# Követelményspecifikáció

Online dokumentum tároló

Szoftverarchitektúrák házi feladat

## Feladatkiírás

A hallgatók feladata egy online felület kialakítása, ahol dokumentumokat lehet tárolni verzió-kezelt módon. A felvitt dokumentumokat lehessen projektekbe, azon belül mappákba rendezni, amelyeknek a hozzáférhetőségét szabályozni lehet. A rendszerbe csak regisztrált felhasználók tudnak adatot felvinni, akiket csoportokba lehet szervezni. Projektet létrehozni az adminisztrátor szerepkörrel rendelkező userek tudnak, illetve ők tudják projekt/mappa szinten beállítani, hogy melyik csoport milyen jogosultságokkal rendelkezik (pl feltöltés, mappa létrehozás, módosítás, törlés stb). A feltöltött dokumentumokhoz lehessen metaadatokat rendelni (név, leírás, absztrakt, téma stb), aminek a köre bővíthető. A metaadatok alapján lehessen dokumentumokat keresni. A dokumentumokhoz lehessen kommenteket fűzni. A rendszer lényeges funkciója, hogy egy dokumentum frissítésekor őrizzük meg a korábbi változatokat is. A rendszer támogassa a dokumentumok zárolását, tehát pl csak zárolt dokumentumot lehet frissíteni és ezt csak az a felhasználó teheti meg, aki a zárat felhelyezte.

## A fejlesztői csapat

A csapat tagjai:

|  |  |  |
| --- | --- | --- |
| név | Neptun | email-cím |
| Angyal Levente | BZGN9J | levi.angyal@gmail.com |
| Szabó Alex | PEONHQ | delanni.alex@gmail.com |

## Részletes feladatleírás

A projekt során egy webalkalmazás elkészítését tűztük ki célul, amely dokumentumok tárolását végzi, ezen felül pedig kezeli a feltöltött dokumentumok különböző verzióit. A dokumentumok tárolásának egységei a mappák, legmagasabb szinten pedig a projektek. Az elkészülő alkalmazás szolgáltatásait a regisztrált felhasználók vehetik igénybe.

Az alkalmazáson belül minden projektnek pontosan egy létrehozója van, ő lesz a Projekt Admin (PA), aki a projekt tekintetében minden jogosultsággal rendelkezik. Ez a szerepkör átruházható egy másik Userre, ezáltal az eredeti PA elveszti PA szerepkörét a projektre vonatkozóan. Ezen kívül megkülönböztetünk Administrator szerepkört. Az Administrator karbantartási célokra használandó, minden projekt és csoport tekintetében minden jogosultsággal rendelkezik. További jogosultságok érhetőek el csoportokon keresztül. Csoportot tetszőleges User létrehozhat, ekkor ő rendelkezik a csoport admin (Group Administrator, GA) jogosultsággal, amit a PA-hoz hasonlóan átruházhat más userre. A PA a hozzá tartozó projekten belül a jogokat rendelhet már létező csoportokhoz vagy egyes felhasználókhoz. Ilyen jogosultságok a dokumentumok feltöltése, módosítása, törlése illetve kommentek fűzése dokumentumokhoz (írási jog, WRITE right), illetve a projekt böngészése, dokumentumok letöltése (olvasási jog, READ right). Dokumentum új verziójának feltöltésekor a régi verzió verzió archiválódik.

## Technikai paraméterek

A fejlesztéshez ASP.NET MVC platformot használunk, mivel az alkalmazás felhasználási területe a web. A dokumentumok meta-információit MS SQL Server-en tároljuk. Az alkalmazás a háromrétegű Model-View-Contoller mintát követi, a Model réteget, amely egy MS SQL adatbázishoz kapcsolódik, az adatelérési osztályok alkotják. A fejlesztés során a Microsoft .NET nyelvek közül a C# nyelvet használjuk az alapvető üzleti logika megírásához, az adatbázis megtervezéséhez T-SQL nyelvet, a felhasználói felülethez pedig HTML és JavaScript nyelveket használunk. A View Engine Razor nyelven lesz megírva. A közös munkához a GitHUB verziókezelő alkalmazás szolgáltatásait vesszük igénybe.

## Szótár

Dokumentum: Tetszőleges fájl az alkalmazás hatáskörén belül.

Feltöltendő dokumentum: A felhasználó által feltöltésre szánt, de még nem feltöltött dokumentum.

Aktív dokumentum: Az adott dokumentum feltöltött és legfrissebb verziója

Archív dokumentum: Már feltöltött dokumentum, amely nem aktív.

Mappa: Tárolási egység, amely mappákat vagy dokumentumokat tartalmazhat.

Projekt: A tárolás legfelső szintje, mappákat és dokumentumokat tartalmazhat.

(Registered) User: Regisztrált felhasználó. Továbbiakban az egyszerűség kedvéért csak User. Az általános szerepkör, amely felvértezhető olvasási és írási jogokkal az egyes projektekhez. Tetszőleges számú jogosultságot kaphat és tetszőleges számú csoportnak lehet a tagja.

Administrator (Admin): Olyan User, aki minden jogosultsággal rendelkezik bármely projektet és csoportot illetően. Rendszer karbantartási célokra használható.

Project Admin: Adott projekt hatáskörében a Project Admin (PA) az a User, aki létrehozta az adott projektet. A PA jog átruházható, de ekkor az eddigi PA elesik a PA jogától. Ezáltal egy projekt esetében csupán egyetlen PA lehet.

Csoport: Userek szervezési egysége. Közös jogosultságok adhatók csoportoknak, amelyek a csoport minden tagjára vonatkoznak.

Csoport Admin: Adott csoport hatáskörében a Csoport Admin (GA) az a User, aki létrehozta az adott csoportot. A GA jog átruházható, de ekkor az eddigi GA elesik a GA jogától. Ezáltal egy csoport esetében csupán egyetlen GA lehet.

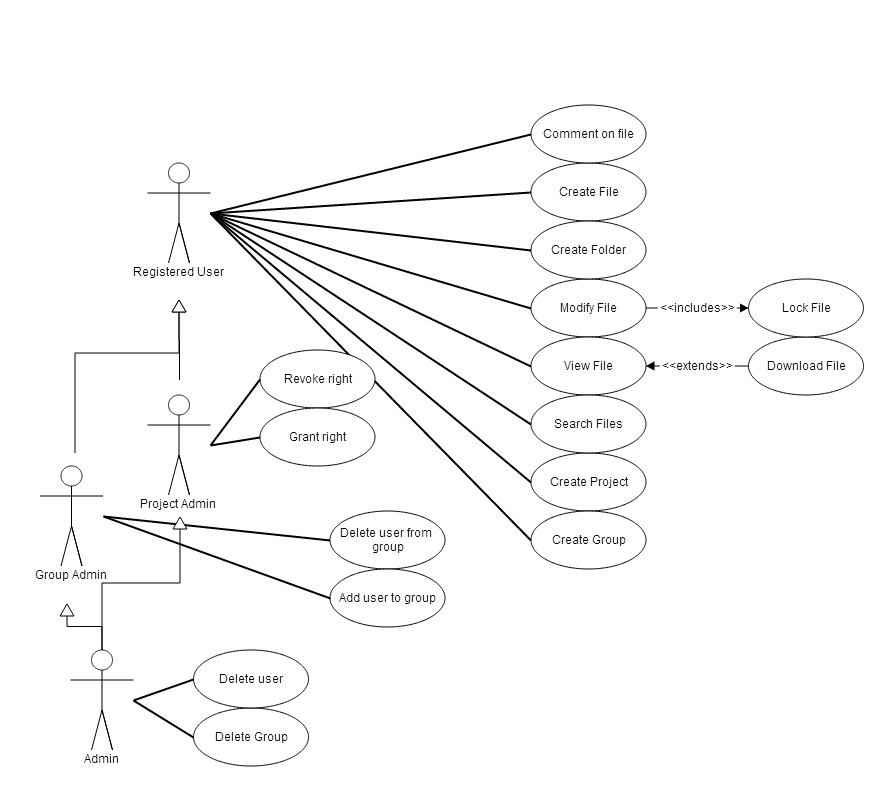
Olvasás jog: Amennyiben a felhasználó rendelkezik ezzel a jogosultsággal egy mappára vagy projektre, akkor jogosult az ebben található összes fájlra, és mappára (és rekurzívan lefelé) böngészési és fájl letöltési jogosultságokkal.

Írás jog: Az olvasási jog kiterjesztése. Amennyiben a felhasználó rendelkezik ezzel a jogosultsággal egy mappára vagy projektre, akkor jogosult az összes ebben található fájl és mappa (és rekurzívan lefelé) böngészésére, letöltésére, törlésére, átnevezésére, metaadatok módosítására, írási zár megszerzésére és fájl feltöltésére.

Metaadat: A dokumentum verziójához, állapotához és tartalmához kapcsolódó információk.

## Essential use-case diagram

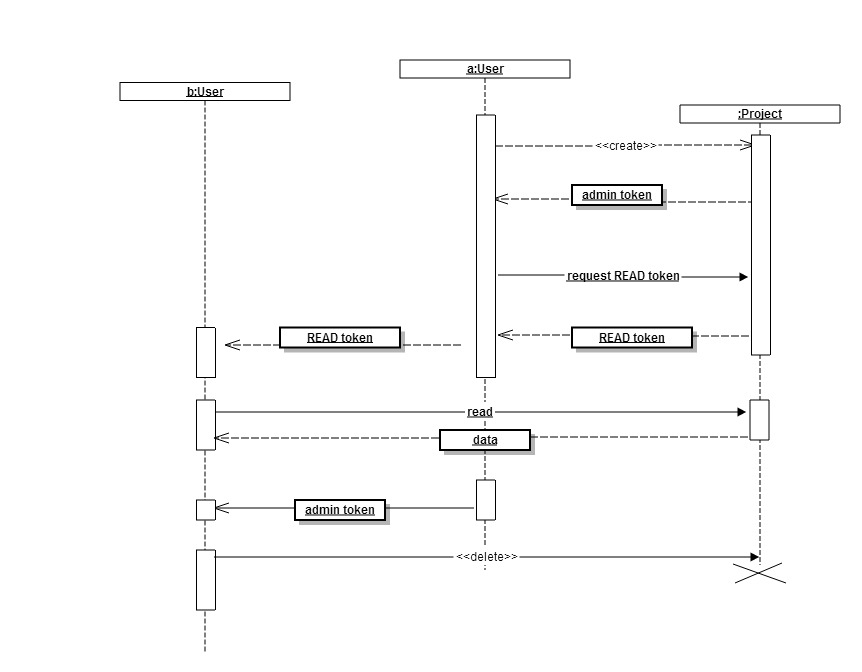
A következő diagram ábrázolja a felhasználói szerepköröket, és e szerepkörökhöz tartozó funkciókat.



A diagramon látható, hogy az alap felhasználó már a legtöbb funkciót eléri, ha az adott projektet és annak egy mappájára tekintettel rendelkezik minden joggal.

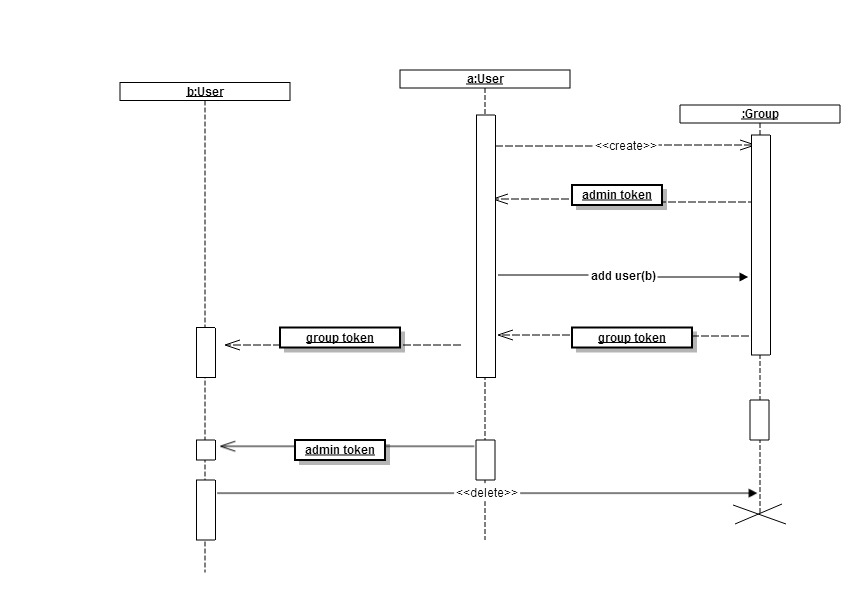
## Életciklus diagramok

A következő Sequence Diagram szemlélteti egy projekt életciklusának egy részét, a fontosabb állapotokkal:



Az ábrán látható egy (a) felhasználó által létrehozott projekt. Ez a projekt létrehozáskor az adminisztrációs jogokat a létrehozónak adja, majd ezzel (a) a projekt egy részére jogosultságot ad egy (b) felhasználónak. (b) ezzel a jogosultsággal sikeresen olvas a projektből. Ez után (a) átruházza az adminisztrációs jogot a (b) felhasználónak, ez után (b) törli a projektet.

Hasonló ábra illusztrálja magyarázat nélkül a csoportok életciklusának fontosabb részeit:



## A felhasználói felület kialakítása

A felhasználói felület megtervezése során elsősorban a modern webfejlesztés trendjei előtt fejet hajtva a kliens felé toltuk a hangsúlyt, szemben a buta vékonyklienssel. Az okos vékonykliens rengeteg előnnyel rendelkezik, többek közt a jobb felhasználói élmény, a dinamikusság, a sebességbeli előnyök, és gyakran a használhatóság is jobb.

A tervezés során úgy döntöttünk, amit lehet, és érdemes, a kliensre juttatva, neki megfelelő módon később ott dolgozunk fel. A böngészőben futó logika nem fogja túlságosan terhelni a processzort, és a sok felhasználós rendszerekben is jobb, ha mentesítjük a szervert a felesleges feldolgozás alól. Hasonló kategóriába esik az oldal adat függvényében történő dinamikus renderelése. Nem végeztünk méréseket, nem is igazán olvastunk utána, hogy mekkora teljesítménybeli javulást eredményez a szerver mentesítése a HTML renderelés alól, de elképzelhető, hogy egy összetett oldal, gyakran történő lekérdezgetés miatt sokat is spórolhatunk egy hasonló, éles rendszerben.

A használt kliens oldali kód a Javascript, és ehhez két jelentősebb framework-öt használtunk. Az egyik a sokak által ismert jQuery, a másik a google által fejlesztett AngularJS, a feltörekvő „MVW” (model, view, whatever) keretrendszer. A jQuery indokoltsága a popularitása, jól ismertsége, és jó használhatósága, bár kevés dolog volt, amire éppen kellett használni, és ne lehetett volna nyers javascript kóddal megoldani, mégis sokat segített, némi időt spórolt. Továbbá a jQuery-re is épít az Angular, tehát egyébként is szükséges volt ezt a függőséget beiktatni. Az Angular pedig a vastagkliensekben megszokott adatkötés, struktúra, aszinkronitás és esemény relaying miatt volt indokolt. A következőkben a jQuery bemutatására nem térek ki, és az Angular-t az elkészített munkán keresztül mutatom be.

Tudok arról, hogy az ASP.NET MVC is jól használható aszinkron könyvtárakkal, és adatkötést segítő könyvtárakkal, és ezek közül nem az Angular a leginkább használt, de némi előzetes tapasztalatom ehhez a könyvtárhoz volt, és szerettem volna még jobban megismerni, amit sikerült a projekt során.

### Mi az az AngularJS?



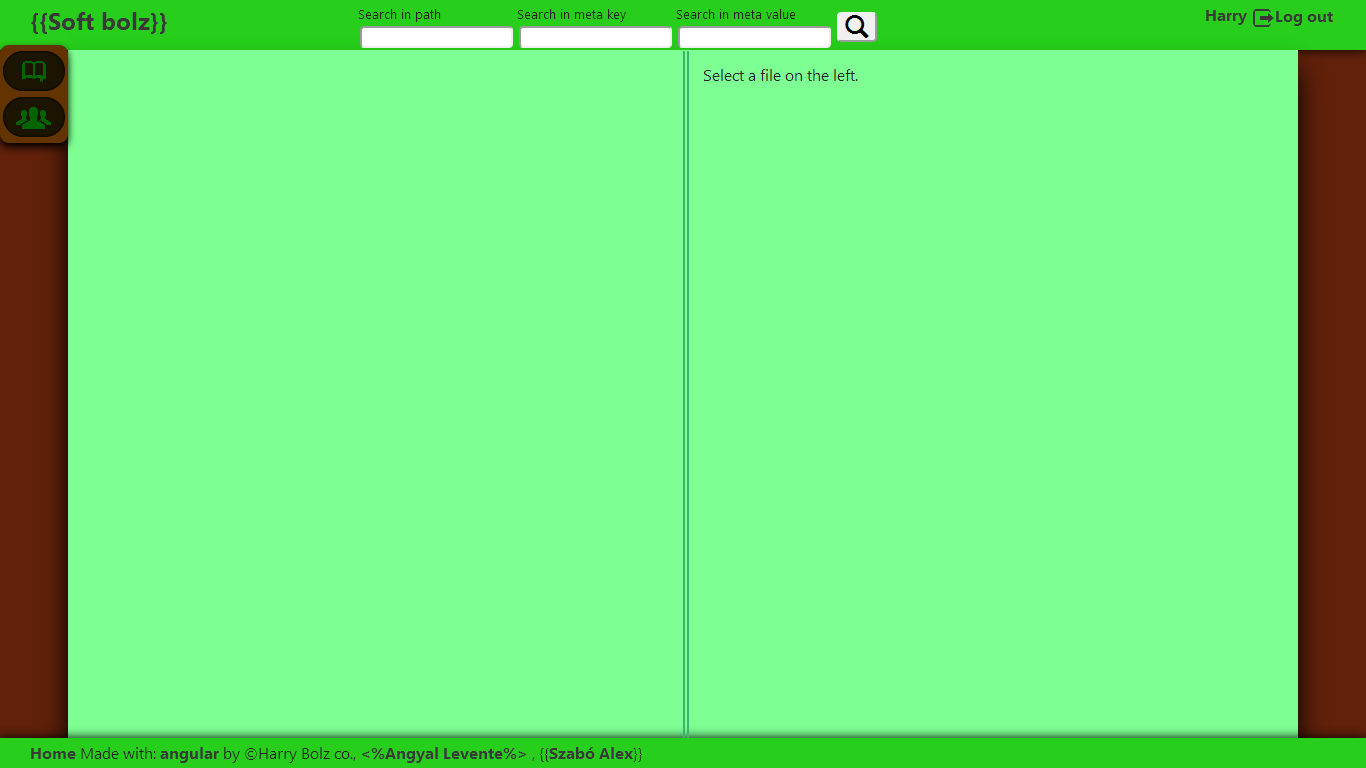
A [weboldal](http://www.angularjs.org)ról idézve: *HTML is great for declaring static documents, but it falters when we try to use it for declaring dynamic views in web-applications. AngularJS lets you extend HTML vocabulary for your application. The resulting environment is extraordinarily expressive, readable, and quick to develop.*

Számomra főleg az adatkötés, a saját kontrollok, és a struktúrára való törekvés marad leginkább jelentős pont. Az általuk említett főbb előnyök a nagy kifejezőképesség, olvashatóság és a gyors fejlesztés. Ezek az egyik adunak emlegetett direktívák jellemzői. A direktívákkal, majd később látható, hogyan lehet saját vezérlőket a jQuery widget megoldásával szemben, inkább a XAML-es megoldásokhoz hasonlóan készíteni.

### Az oldal és az alkalmazás váza

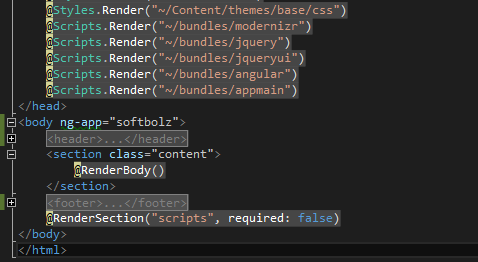
Végeredményben az oldal egy részét az MVC render engine hozza létre, de nem igazán használtuk semmilyen adat megjelenítésére, szimplán az egyik statikus HTML keretbe rendereltetünk egy másik statikus (a renderelt scriptektől és css sablonoktól eltekintve) HTML oldalt.

Az 1. ábrán látható kép az oldal alap elrendezését hivatott bemutatni. Az oldal 2 kisebb és egy nagyobb részre osztható, amely 3 további részre oszlik. A képen az első hármas felosztás vízszintesen osztja fel az oldal struktúráját. A két kisebb rész a felső és alsó navigációs sáv. A felsőn a keresőkön kívül a megszokott homelink és a login linkek találhatók, az alsó navigációs sávon pedig némi placeholder jellegű információ.

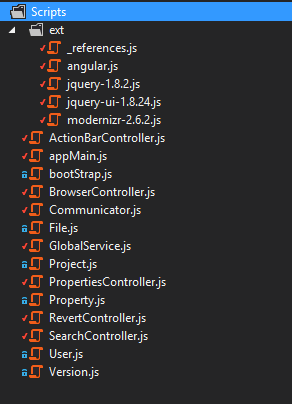


1. A képen látható az oldal első bejelentkezés utáni látképe

A 2. ábrán jól látszik ez a hármas felosztás: a body egy header, egy footer és egy section részből áll össze, ebből a section tartalma a Home kontroller Index.cshtml-jében található, vázát később szemléltetem.

  
2. A kódban jól körvonalazódik a hármas felosztás

A @RenderBody() mögött mindig az aktuális View található, tehát ez a váz közös az összes oldalon, csak a belső tartalom változik. Tulajdonképpen a mi oldalfelépítésünk mindvégig nagyjából egy View-t használ, a Home kontroller Index actionjéhez tartozót. Nem indokolt másik használata, mert kevésszer töltődik valójában újra az oldal, és akkor is gyakran ugyanaz az egy oldal, és még csak új mögöttes adat sem kerül renderelésre.

Az említett másik hármas felosztás az 1. ábrán a középső sáv függőleges felosztásával keletkezik. A bal oldali vékony kis sáv az „Action Bar”, a középső a „Browser panel” és a jobb oldali rész a „properties panel”. Ezek tulajdonképpen a később említett Angular kontrollerek vázai, mindháromhoz külön kontroller tartozik.

Javascript oldalról a kód négy részre osztható, ebből a kisebb részek: az **egyéb** kategória (olal méretezéssel, apróbb besorolhatatlan logikákkal) és a **keretrendszerek-könyvtárak**, a nagyobb és említésre méltóbb részek pedig: a minimális logikával rendelkező **modell osztályok** kódjai, és a fő **alkalmazáslogika** kódja (angular kontrollerek, direktívák, szolgáltatások). Az első kettő valóban nem sok további említést érdemel, a használt osztálykönyvtárak kódja, a 3. ábrán látható **ext** mappában van elkülönítve a saját kódtól.

A korábban említett felhasználói felület réteg, tehát, ha úgy nézzük további három rétegre osztható a **MVW**-nek megfelelően a kliensben.

* A M, mint Model a kevés logikával rendelkező osztályok rétege, pl a File osztály, ami a dokumentumokat reprezentálja a többi javascript kódban.
* A V mint View a HTML kód/markup (és az őt formázó CSS), aminek releváns része csak az Views/Home/Index.cshtml-ben található, itt vannak leírva a sablonok a saját vezérlőkhöz, és itt vannak a mögöttes adat kötésre váró elemek, minden ami valójában megjelenik.
* A W mint Whatever, ami gyakorlatilag a kontrollereket, direktíva kódot, és kontrollereket jelent. Például a BrowserController kódja, amely a browser panel viselkedését írja le.

### Az alkalmazás működése

A következőkben az AngularJS fő építőköveinek bemutatásával egy-egy alkalmazásbeli példán keresztül szemléltetem az alkalmazás felépítését. A fennmaradó, nem említett példák értelmezése említés szinten marad, de a kód részleteiben nem kívánok elmerülni.

#### A modul

Az Angular (a továbbiakban angular, vagy ng) keretrendszer lehetőséget ad alkalmazások definiálására (más néven modul) amelyek a különböző alkalmazás profilokat hivatottak kiszolgálni, ez a klasszikus oldalszervezésben több html oldal összességéből állt volna össze, amelyek egy cél érdekében élnek, tehát a weboldal egy apektusát fedik le.

A mi esetünkben egy modul-t készítettem, mert igazából egy oldal van ami kiszolgálja a használat 90%-át, és egyedül ide kell ng logika, és az inter-modul kommunikáció nehezítette volna az implementációt. Mivel pedig egy modult definiáltam, ezt a body szintjén tettem, így minden ng logika minden oldalra kiterjed (mert a body valamely mélyebb szinten lévő tartalma van újra-újra renderelve a *RenderBody* hívásokban), legfeljebb nincs használva.

Modult a következő egyszerű módon definiálhatunk:

var NGMAIN = angular.module("softbolz", function () {

// Configure if needed

});

A modul a javascript kódban *NGMAIN* néven fut, utalva a modul jellegre a nagy betűkkel, később ezen a modulon tudunk definiálni kontrollert, és egyebeket. Az angular module függvénye használható így, két paraméterrel, ekkor az első paraméter a modul neve, amelyet a 2. ábrán látható *ng-app=’softbolz’* attribútum köt a HTML-hez. A második paraméter egy függvény, amit egyszer futtat le az ng (javascript kódbetöltésenként), amivel az injektorokat, és egyéb mágikus részleteket tudunk testreszabni, de ez nem lényeges.

A modul definiál egy úgynevezett $rootScope-ot, amely az alkalmazás egészére ad egy környezetet. Ez nagyjából analóg a XAML-es leírókban megadható DataContext attribútummal, ugyanis a gyermek elemekben (html-vázban tekintve) ez a kontextus felülírható, azonban ugyanúgy a leszármazottak láthatják a szülő kontextusát.

A $rootScope, $scope és egyéb $-al kezdődő változónevek az ng esetében arra utalnak, hogy ezek a függőségek injektáltak, és hogy nem hagyományos változók, ugyanis lényeges hogy ugyanilyen változónéven legyenek (függőség injektálás miatt), szóval óvatosan bánjunk velük, és nyugodtan számítsunk mágiára velük kapcsolatban.

Az elkészült modult, ha a HTML markupban beállítottuk a megfelelő helyen, ezután kontrollerek, direktívák és szolgáltatások készítésére használhatjuk.

#### A kontroller

A kontrollerek az alkalmazás részfeladatainak elvégzésére használható. A kontrollereknek egymástól különböző kontextusuk van, így rendezetten tarthatjuk (mind kódban, mind változók szintjén) az alkalmazást, amely mögöttes logikaként szolgál az egyes UI vezérlőknek.

Az alkalmazásunkban összesen 5 kontrollert definiáltam, rövid magyarázattal:

* **ActionBarController:** Az éppen elvégezhető, releváns műveletekhez biztosít kényelmes hozzáférést. Állandóan frissül attól függően, hogy mit választunk ki, és arra az elemre milyen érvényes műveletek vannak.
* **BrowserController:** A fájlstruktúra böngészésére használható, böngészés közben frissíti az éppen kiválasztott entitást, és szükség esetén frissíti saját tartalmát is.
* **PropertiesController:** A jobb oldali panel vezérlője, a választott entitás részletező nézetének szolgáltat adatot.
* **RevertController:** A verziók áttekintésére, és revertálás végrehajtására használatos apró kontroller.
* **SearchController:** A keresést támogató kontroller.

Ezek határvonala jól érezhető. Akkor kell új kontroller, ha valami más jellegű adatot szeretnénk összefogni, és műveleteket végezni rajtuk, akkor kell új kontrollert bevezetni. Ha valami féle új adatforrás kell, vagy üzleti logikát akarunk kiemelni, arra nem feltétlenül a kontroller a megoldás, lehet szolgáltatást használni.

NGMAIN.controller("RevertController", function ($scope, Communicator, GlobalService) {

$scope.$on("fileSelectionChanged", function () {   
 $scope.populateVersions(GlobalService.selectedFile); });

$scope.populateVersions = function (file) {

if (file && file.versions) {

$scope.versions = file.versions;

$scope.selectedVersion = file.versions[0];

}

}

$scope.versions = [];

});

Példaként egy kisebb kontrollert választottam ki, a verziók visszaállításáért felelőst. A modulon a *.controller()* függvénnyel tudunk új kontrollert definiálni, melynek paraméterezése az egyszerűbb szintaxissal megadott a példában, és itt első paraméterként kell átadnunk a kontroller leendő nevét, amelyet később a HTML-kódban hivatkozhatunk, második paraméterként szintén egy függvényt, ami inicializálja a kontrollert. Ezen függvény paraméterül kaphat tetszőleges számú változót, mint függőséget, a lényeg, hogy az egyszerűbb szintaxis szerint ezeknek olyan változónévvel kell szerepelniük, amilyen néven definiáljuk a kód többi részében. A kontrollerek definiálásának sorrendje lényegtelen, az előbb említett dolog pontosan ezt hivatott szolgálni, ugyanis függőség injektálás van a háttérben. A lényeg, hogy ne alakítsunk ki körkörös függőségeket, a többit az ng intézi.

A kontroller kódjában feliratkozom egy eseményre, ami a *$scope*-omon történhet (vagy annak bármely ősén) „*fileSelectionChanged*” néven, ez jelenti azt, hogy a callback függvényt hajtsuk végre mindig ha változik a kiválasztott fájl a böngészőben. Ekkor a később definiált *populateVersions* függvényt hajtsuk végre a választott fájllal. Ez a függvény tulajdonképpen elkéri a fájl objektumon lévő verziókat, majd azokat egy a *scope* változójában tárolja. Tudniillik a *$scope*-on definiált változókhoz lehet adatot kötni, és a *$scope*-on definiált függvényeket tudjuk hasonlóan hívni a kontrollerhez tartozó vezérlőben. Végeredményben tehát a versions változóhoz kötött elemek tartalma frissülni fog amennyiben változik a kiválasztott fájl, ez pedig a verziókat listázó vezérlő <select> listája:

<div id="revertDialog" style="display: none" ng-controller="RevertController" title="Revert file...">

<div>Select a version:</div>

<select style="width:100%" ng-model="selectedVersion"

ng-options="'version '+v.versionNumber group by v.date for v in versions">

</select>

<input type="hidden" value="**{{**selectedVersion.versionNumber**}}**" />

</div>

Látható hogy a *ng-controller* attribútumban adjuk meg a vezérlőt, tehát innentől lefelé a tag-fában érvényes a fenti $scope kontextusa. A select html elemen lévő két idegenkedő attribútum adja meg gyakorlatilag a dinamikusan változó tartalmat a lenyíló listában. A *ng-model* az adatkötött mezőt jelöli, a *ng-options* pedig egy *select* specifikus direktíva, ami azt mondja, hogy vegyük v-ként a az elemeket a *versions*-ből, csoportosítsuk őket *v.date* szerint, és jelenítsük meg *v.versionNumber*-t, a plusz szöveg előtte csak az olvashatóság miatt került oda. Mivel később ezt a kiválasztott értéket egy jQuery hívásban, teljesen más helyről, és az angular-on kívülről akarom megszerezni, ezért ezt egyenesen beleteszem egy rejtett input mezőbe. A {{}} jelöléssel tudunk jelölni behelyettesítendő értéket. Ha az angular piszkosnak érzi a $scope-ját, akkor végigmegy, feldolgozza annak értékeit, és kicseréli a {{}}, *ng-bind*, *ng-model* és a többi ng vonatkozású frissítendő mezőt a kontroller vázában.

#### A szolgáltatások

A szolgáltatások konstrukciójukban hasonlóak a kontrollerekhez, azonban nem jellemző a kontextus, és emiatt a feldolgozás, változásérzékelés, és akármilyen adatfrissítés sem. Sokkal inkább a közös célok megvalósítására használatos, több vezérlő által. A szolgáltatások a modellekkel hozhatók párhuzamba, ugyanis adatszolgáltatásra, adattárolásra, és üzleti logika implementációjára használhatók leginkább.

E két fő módra van pontosan két példa a kódban:

**Communicator**: Alapvetően a hálózati forgalom bonyolítására készült, de később minden kérés indítással, akcióval és paranccsal kapcsolatos logika implementációja került ide. Itt van implementálva minden kapcsolódás a REST API-val.

**GlobalService**: Az akciók leírása, és több kontrollerből használatos objektumok tárolása a fő feladata.

Mindkét szolgáltatás méretes, ezért csak egy-két részletet emelek ki a Communicator-ból. Az elkészítést, aminek paraméterezése megegyezik a kontrollerével (csak itt nincs értelme a $scope-nak). Egy függvény definíciót, és a visszatérést. Tulajdonképpen ez az egy darab visszatérési érték az amelyet visszakapunk valamikor is egy kontrollerből hivatkoznánk erre a szolgáltatásra, tehát ezen keresztül kell publikálni a függvényeket, amelyek a kívánt üzleti logikát valósítják meg.

NGMAIN.factory("Communicator", function ($http, $q, $rootScope) {

…

var downloadFile = function (file) {

window.open("/Download" + absolute(file.filePath + file.fileName));

};

…

return {

listFolder: listAsync,

performSearch: performSearch,

revert: revertFile,

delete: deleteResource,

uploadFile: uploadFile,

updateMeta: updateMeta,

download: downloadFile,

tryLock: tryLockFile,

unlock: unlockFile,

createProject: createProject,

createFolder: createFolder,

createGroup: createGroup,

notifyFS: notifyFS

}; });

#### A direktívák

Személy szerint az egyik kedvencem az angular-ben. Bár a kódban csak elemek generálására használtam, ennél sokkal szerteágazóbb lehet felhasználásuk.

A direktívák segítségével tudunk saját attribútumokat és xml elemeket definiálni, amelyek a kódban kifejtődnek a megadott sablonra. Hasonlóak a felhasználói vezérlőkhöz, mert saját kinézetük van, és lehet hozzájuk saját scope-ot is csatolni.

Négy direktívát definiáltam, mindet az *appMain.js*-ben, mert rövidségük miatt nem érdemes szétszórni őket egy-egy fájlba. A hozzájuk tartozó sablon kódot a *templateUrl* megadásával a HTML-be írhattam, így ott láthatjuk definiálva, ahol használva is lesz. A 4 direktíva:

folder, file, project: A böngésző panelben megjelenített három féle sor elem váza.

propertyfield: A részletező nézetben megjelenő fájl tulajdonságainak megjelenítésére, szerkesztésére.

Két példán szemléltetem használatukat, ez előbbin a projekt direktíva kódja található. A már ismerős mintában a második paraméter egy olyan függvény, aminek egy objektumot kell visszaadnia, ami alapján az elem elkészülhet, ennek az objektumnak a kulcsai konfigurálják a direktívát. Például, a *scope* kulcs alatti objektum építi fel az elem kontextusát, a példakódban pl. átveszem érték szerint a *projectname* és *ownername* attribútumokba írt értékeket; a *restrict:”E”-*vel elemekre korlátozom a direktíva használatát, és a *templateUrl*-el megmondom a markupot, ami leírja.

NGMAIN.directive("project", function () {

return {

scope: {

projectname: "@",

ownername: "@"

},

restrict: "E",

templateUrl: "projectTemplate",

};

});

NGMAIN.directive("propertyfield", function () {

return {

scope: {

property: "="

},

restrict: "E",

templateUrl: "propertyTemplate",

};

});

Az utóbbin hasonló minden, de érték szerint veszem át a property-t, tehát a template-ben köthetek hozzá változókat, és ezek két irányú adatkötés esetén, az eredeti objektumon változtathatnak.

A hozzá tartozó HTML pl így néz ki:

<script type="text/ng-template" id="propertyTemplate">

<div class='property'>

<div class='propname' style='font-weight:bold'>**{{**property.propName**}}** : **{{**property.propValue**}}**</div>

<label class='edit'>Property name:</label>

<input class='edit propname' ng-model='property.propName'/>

<label class='edit'>Property value:</label>

<input class='edit propvalue' ng-model='property.propValue' />

</div>

</script>

És használata:

<propertyfield ng-repeat="property in global.selectedFile.properties" property="property"></propertyfield>

Tehát ismételjük meg a propertyfield elemet minden property-re az említett gyűjteményben, és adjuk át ezt a property-t, hogy a fentebb említett rész modellként tudja használni.

#### Összefoglalás

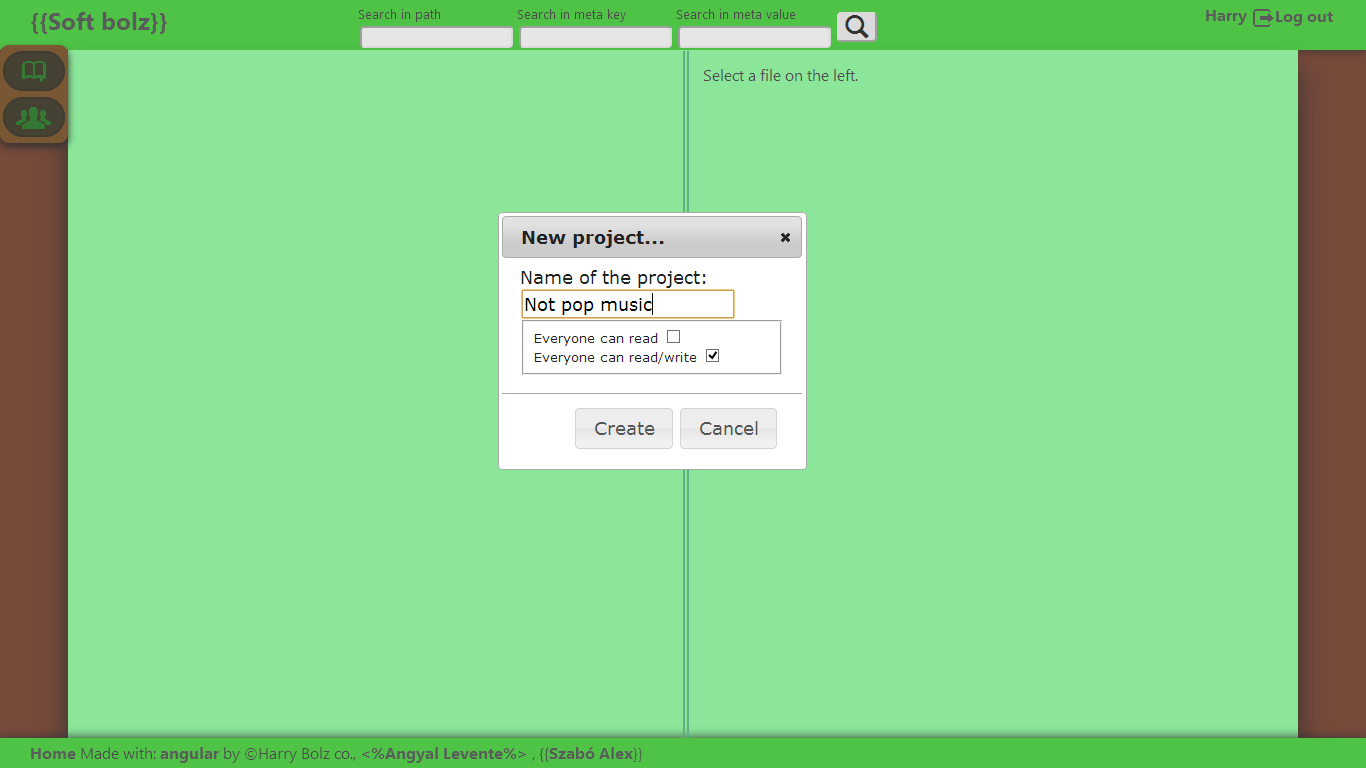
A UI kód a reszponzivitás miatt, a kommunikáció miatt, és egyéb okokból nagyon bonyolult lehet, és a idővel belassul a fejlesztése. Ha szeretnénk távol tartani magunkat a spagetti kódtól, és szeretnénk jól kinéző, jól működő webalkalmazást, és jól szervezett kódot írni érdemes befektetni annyi időt, hogy az ember megtanulja használni valamelyik jól menő MV\* keretrendszert, és ezzel hatékony kliens oldali kódot tudjon írni, hogy minél több dolgot lehessen a kliensre bízni, és a szervert egy biztonságos adatszolgáltatóként lehessen használni. Természetesen nem tudtam teljes mértékben elmagyarázni az alkalmazás működését, de ha sikerült valamit megragadnom az Angular szellemületéből, akkor a kódom olvasható, és jól értelmezhető maradt annak komplexitása ellenére.

### Az alkalmazás használata

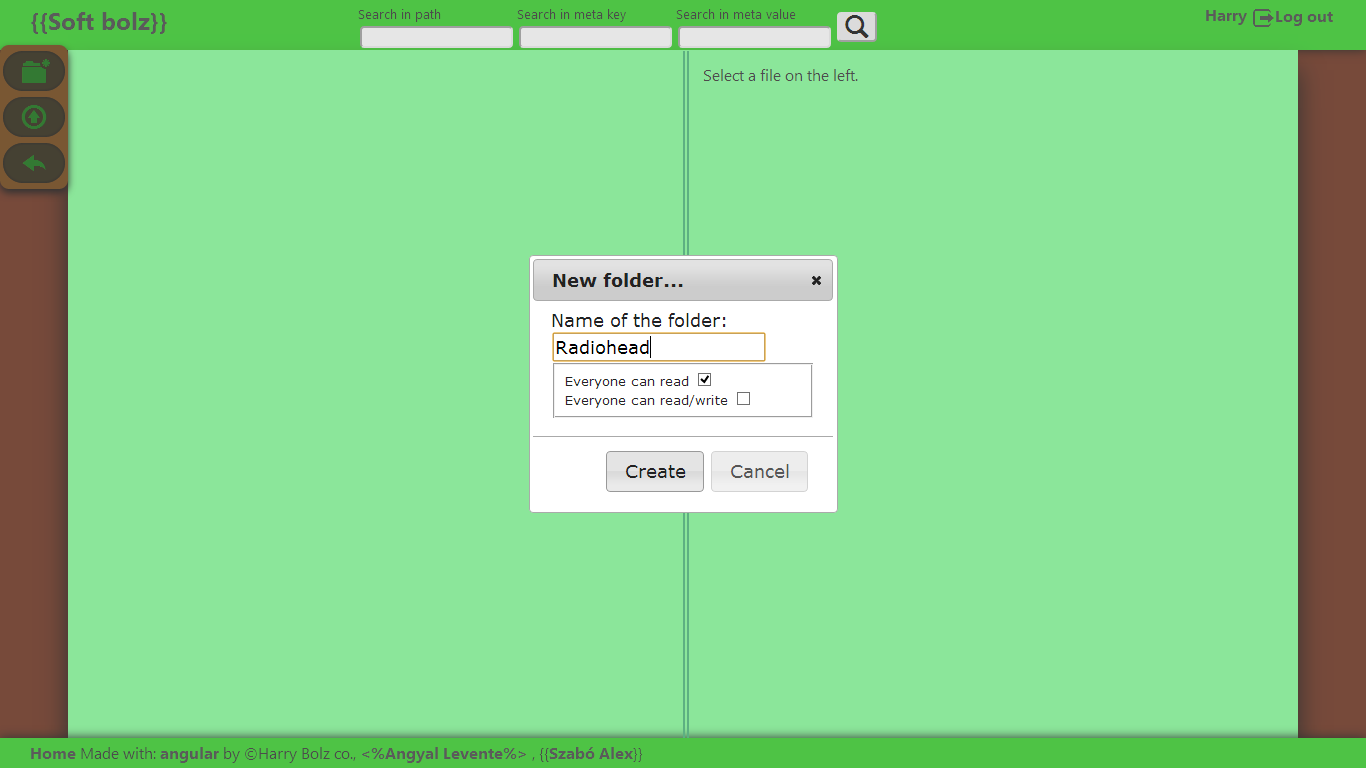
A következőkben röviden áttekintést adok az alkalmazás használatáról. Sajnos néhány feature még nem működik, ettől függetlenül ezeket is megemlítem.

A regisztráció kicsit lepusztult kinézetű, de magától értetődő, szóval azt a lépést kihagyom.

Előttünk tehát a kezdőkép. Baloldalon az Action Bar-on válasszuk ki a könyv jelet. Itt beírhatjuk a nevét a létrehozandó projektnek. A készítéskor választható jogosultságok gyakorlatilag a publikus írás, publikus olvasás, vagy a teljes kézi jogosultságvezérlést szabályozzák. Kattintsunk a create gombra.

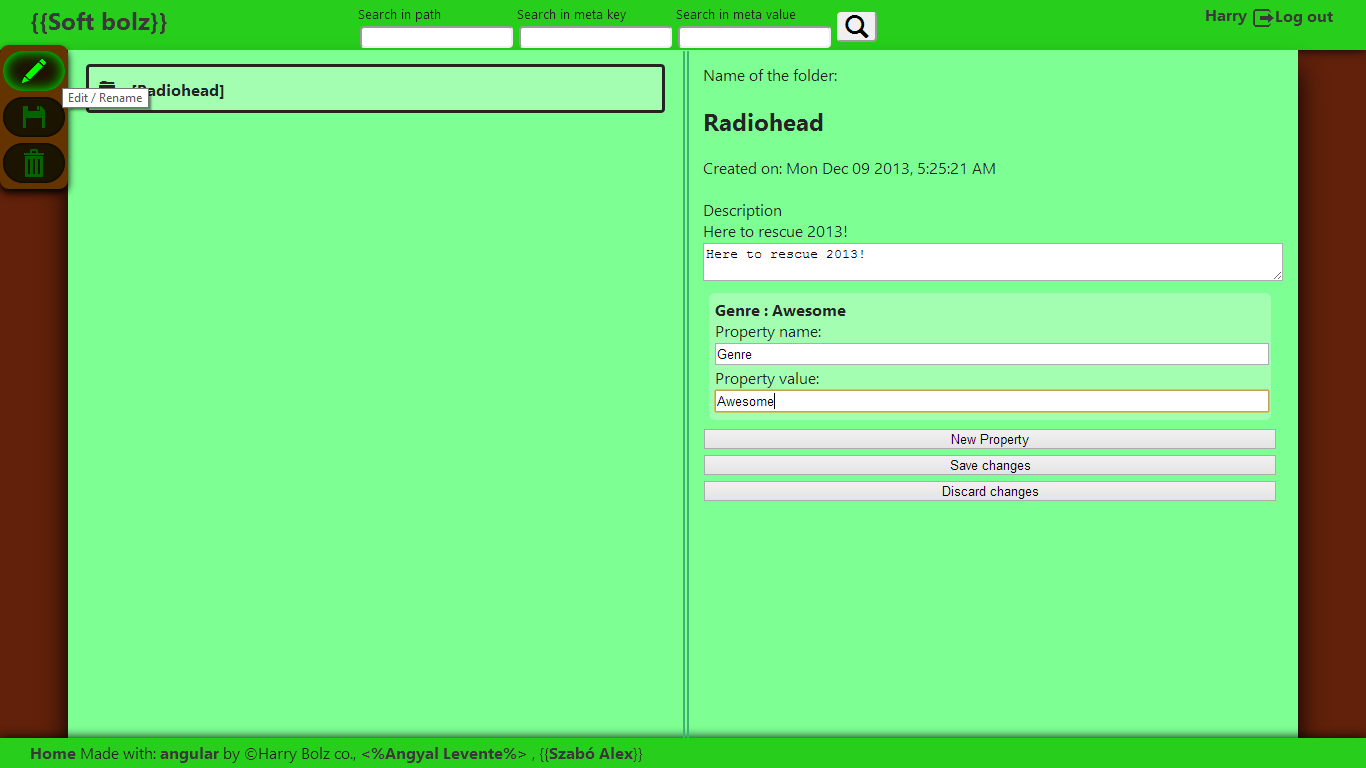


A készített projekten kattintva, kiválaszthatjuk, ekkor megjelennek oldalt a lehetőségeink a projekttel kapcsolatban, és ha ismét rákattintunk a projektbe lépünk. A projekten belül csinálhatunk mappákat, vagy tölthetünk fel fájlokat. Csináljunk tehát itt is egy mappát:



Amiután a mappa elkészült, válasszuk ki ismét, és a bal oldali menüből a ceruza jellel tudjuk szerkeszthetővé tenni a fájljaink metainformációit.

Írjunk valami megjegyzést, esetleg adjunk hozzá metainformációt. Ekkor a jobb oldali „Save changes” vagy a bal oldali floppylemezzel tudjuk érvényesíteni.

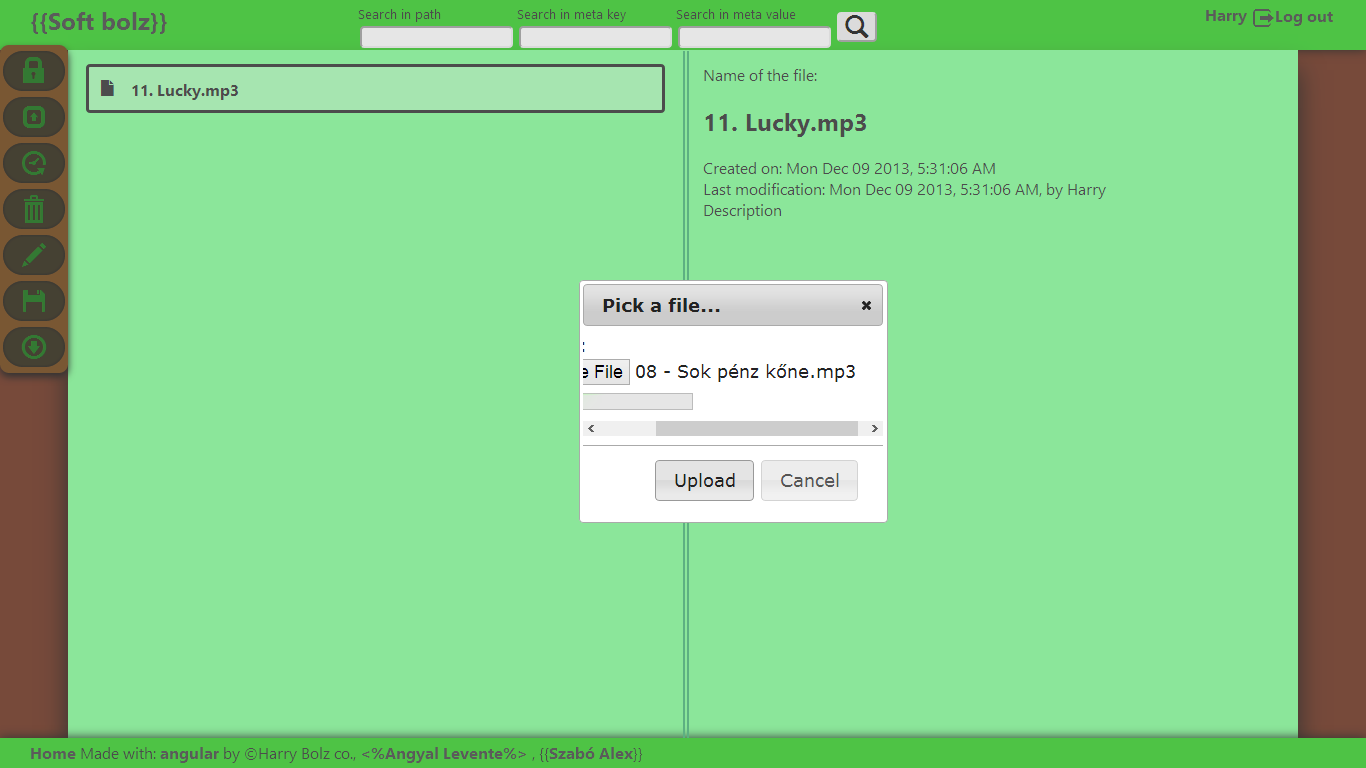


Ha üres területre kattintunk, újra lehetőségünk nyílik mappát csinálni, vagy fájlt feltölteni. Lépjünk be az új mappánkba, és töltsünk fel egy fájlt a fájl feltöltő ikonnal. Ha ez sikeres, akkor a böngészőben megjelenik a fájlunk. Ezt kiválasztva lehetőségünk nyílik zárolni a fájlt, hogy tudjunk rajta módosítást végezni.

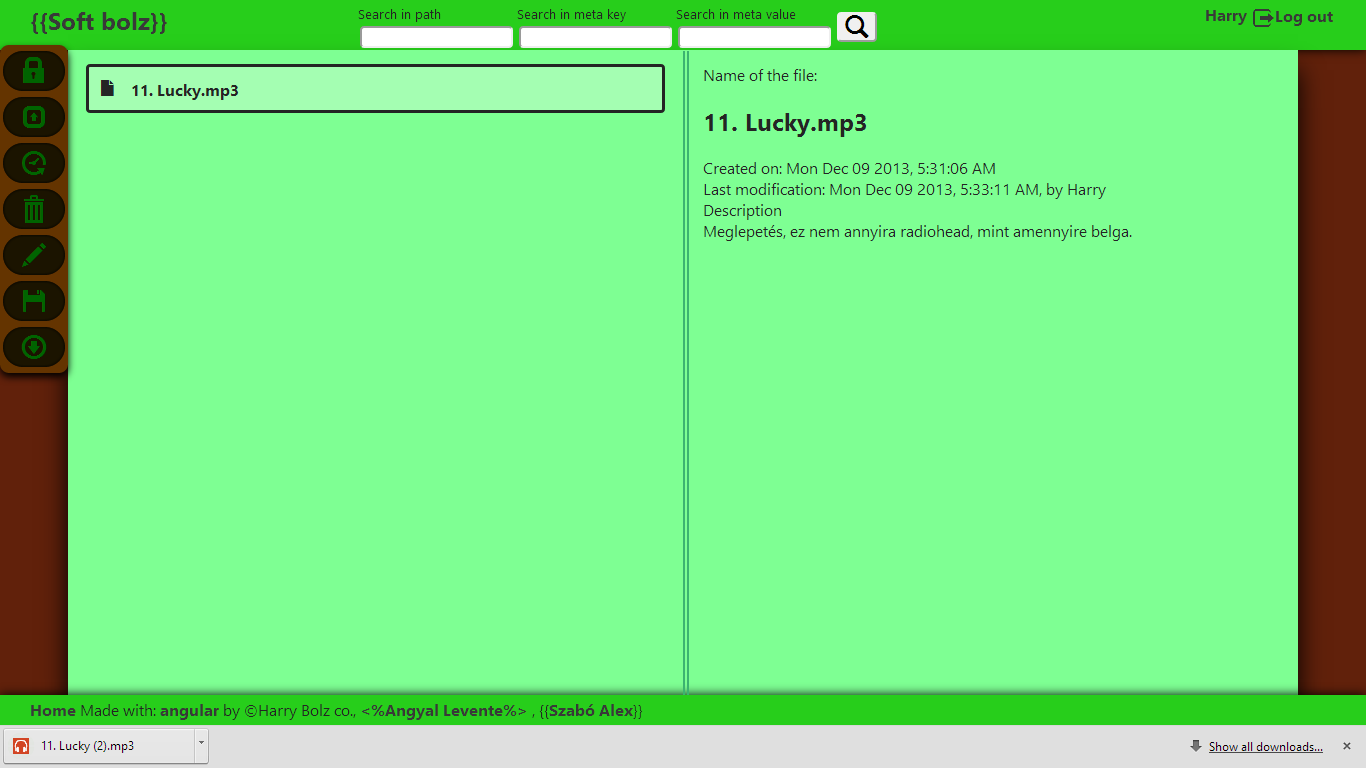


Megszerzett zárral tudunk csak új verziót feltölteni a fájlból, vagy törölni a fájlt. Igazából egy kiskapuval - ha a mappán írási jogunk van – bármikor törölhetjük más fájljait. Ez egy újragondolandó pont a programban.

Ha már nálunk a zár, ragadjuk meg az alkalmat is, és töltsünk fel egy új verziót a fájlból, ami lehet más nevű fájl is, mert átnevezésre kerül.

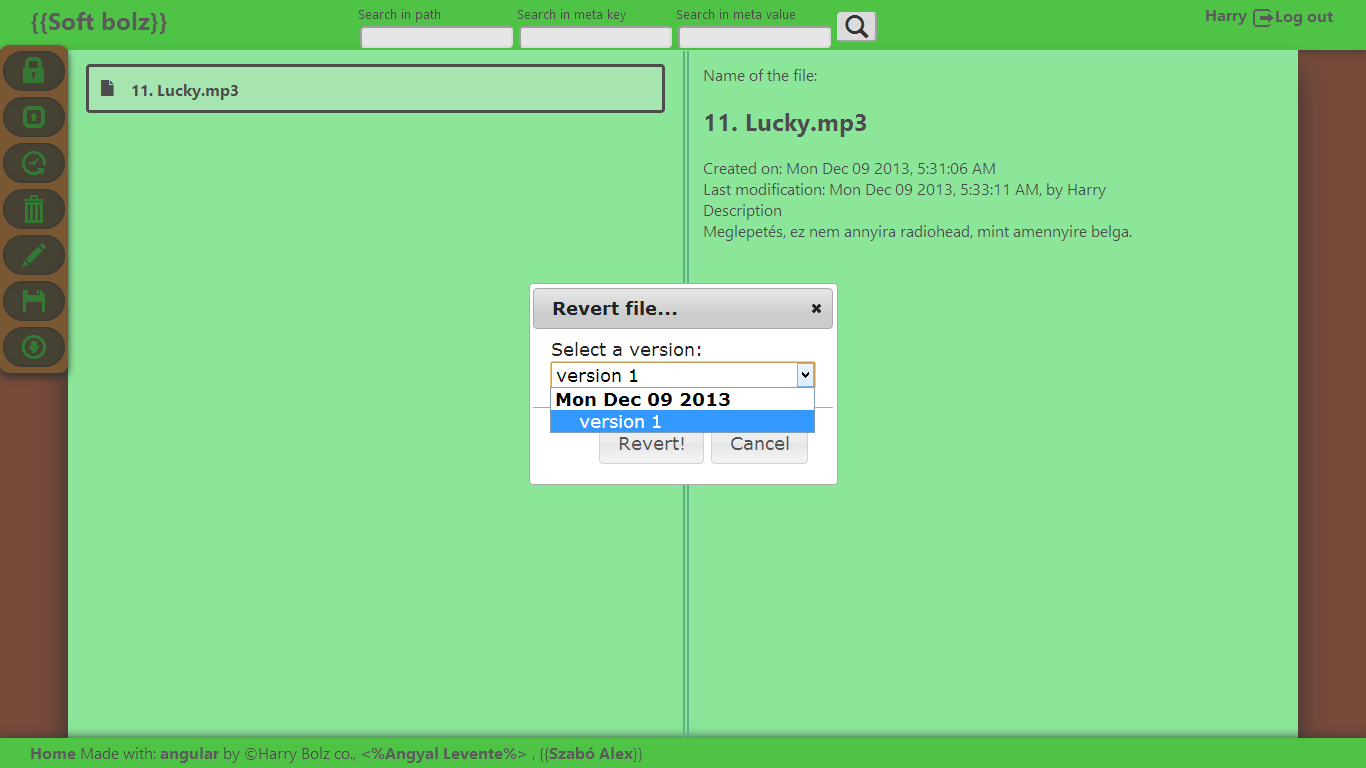


Töltsük le a bal oldali menüben adódó legalsó opcióval a fájlt, és énekeljük együtt a Belgával (bár a Radiohead dalának álcázza magát), hogy Sok pénz kőne…, majd ha végeztünk térjünk vissza a feladathoz. Írjunk megjegyzést a fájlhoz, hogy más ne tévessze össze a két együttest.



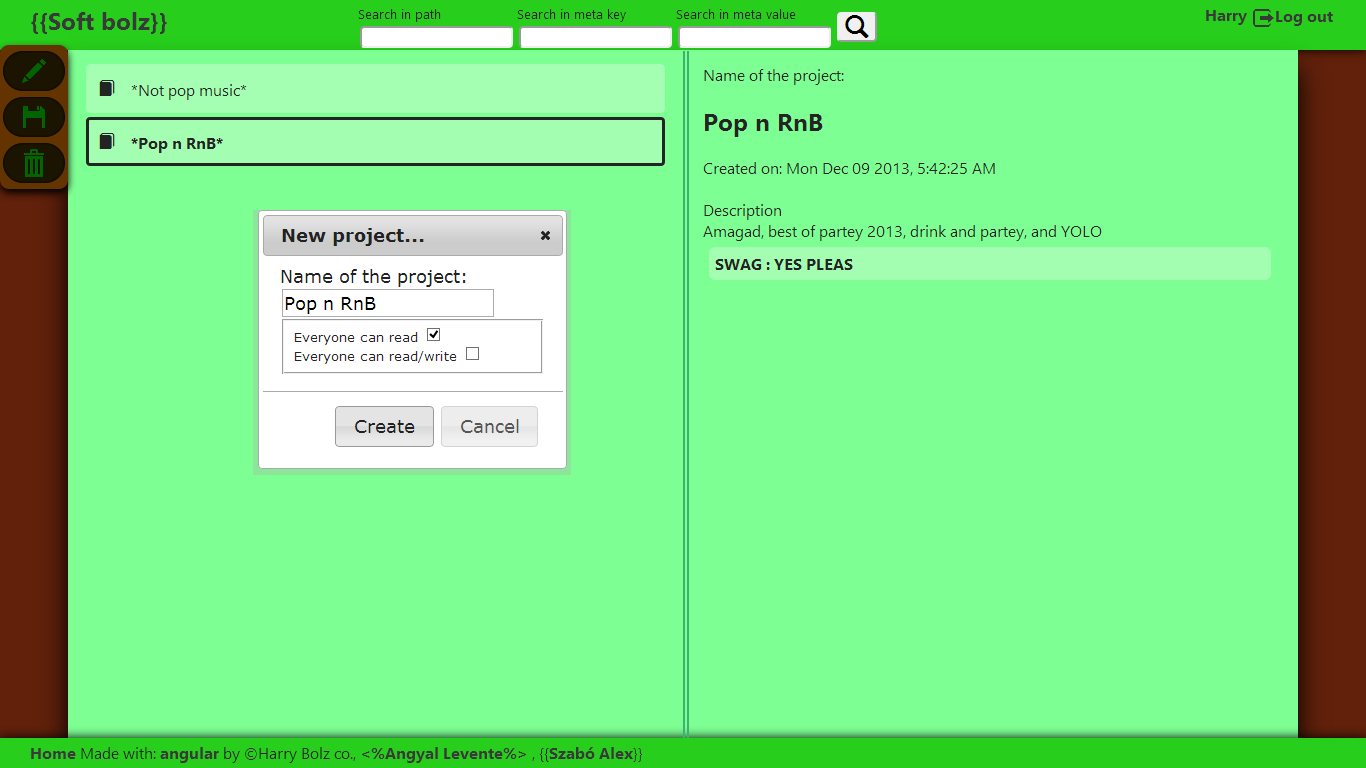
Igazából a megjegyzés nem elég. Az emberek nem szeretnek olvasni, szóval nem fogják elolvasni, letöltik a Radioheadet, és egy óvatlan szoftverhasználó már hívja is a Fókuszt, hogy beszámoljon a szenzációról, miszerint a Radiohead magyarul énekel.

Inkább csináljuk vissza a változtatást. Ehhez, ha még nálunk van a zár, a bal oldali menüből válasszuk ki a verziózó opciót, és válasszuk ki az első fájl verziót, és állítsuk vissza azt.



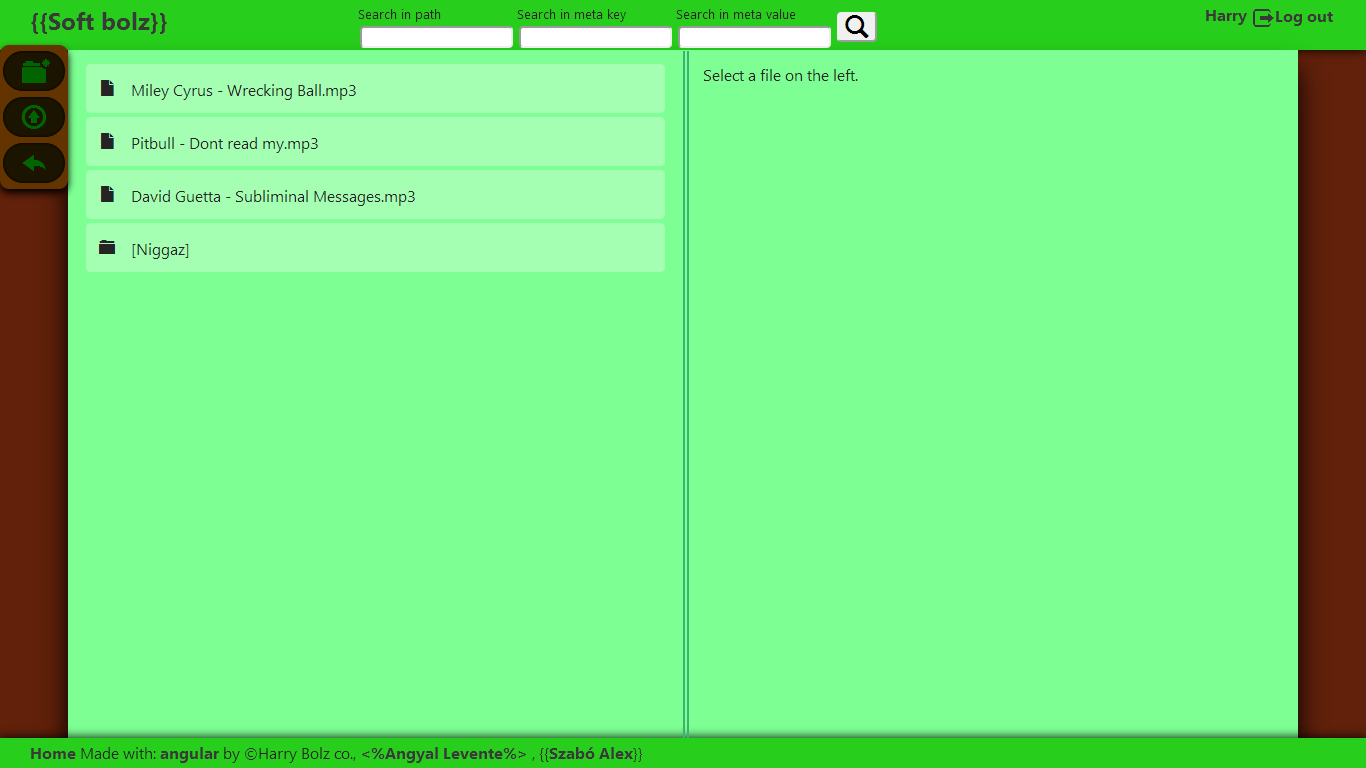
Ekkor letöltés után ismét angolul kell, hogy énekeljenek a dalban.

Csináljunk egy másik mappát is, ahova a „többi” dalt tehetjük.



Bár beállíthatjuk a mappákra és a projektekre a jogosultságokat. Ha kijelentkezünk, látjuk, hogy nem tökéletes valami. Az a probléma, hogy jelen implementációban a virtuális fájlstruktúrában lefelé még nem tudunk jogot adni, mert csak készítéskor van lehetőség jogot kiosztani, készítéskor pedig nincs még senki, aki leszármazott. Ezt javítani tervezzük.

Adjunk hozzá fájlokat az új mappához, és lépjünk vissza egy szinttel, hogy használjuk a keresést (természetesen ezen a szinten is működik).



A keresőbe beírhatunk egy útvonalat, amellyel olyan fájlokat listázhatunk ki, amelyek tartalmazzák a kifejezést. VAGY viszonnyal a metainformációk közti keresés eredménye is hozzájön az eredményekhez. Metainformációk esetén akkor keresünk, ha legalább valamelyik mező ki van töltve, és ha mindkettő ki van töltve, akkor azok a fájlok listázódnak, amelyikek kulcsa a kulcsot, értéke az értéket tartalmazza.



A főoldalon létrehozhatunk továbbá felhasználói csoportokat is, de a felhasználók felvétele csoportokba, a csoportok managementje, és a jogosultságok managementje már nem igazán fért bele az időbe. Igazság szerint így is kicsit túlhaladtunk az előírt ajánlott időkereten. Ettől függetlenül a rendszer jól használható egy fő részére, így biztosított a teljes jogosultság mindenhez, amit az az egy fő létrehoz.