Technická dokumentácia pre modul Cord.Web

Verzia 16.4.2016

Tabuľka 1. Autori

Autor	Rola
Peter Vrana	

Tabuľka 2. História zmien

Verzia	Dátum	Autor	Popis
1.0	3.3.2016	Peter Vrana	Vytvorenie dokumentu
1.1	3.3.2016	Peter Vrana	Autorizacia api volaní
1.2	16.4.2016	Peter Halaš	Presun jednej triedy do Core.Mvc
1.3	15.5.2016	Ivan Beňovic	Presunutie konfiguracii do Admin. portalu
1.4	15.5.2016	Ivan Beňovic	Git konfigurácie repozitárov

Obsah

1 Úvod

V dokumente sú opísané zmeny modulu Cord.Web, v rámci tímového projektu DevActs, ktorý je jedným z mnoho modulov, ktoré sú súčasťou architektúry projektu PerConIK.

2 Analýza

Modul UserActivity.Web slúži ako prístupový bod k repozitárom, vetvám a komitom, umožňuje rôzne variácie vizualizácie komponentov, vypočítava nad pridanými repozitármi rôzne štatistiky ako napr. pachy v kóde. Tento modul zahŕňa samostatný autentizačný a autorizačný mechanizmus, ktorý je potrebné refaktorovať. Na autentifikáciu a autorizáciu sa využije REST rozhranie poskytované Administračným portálom, avšak časť funkcionality spojená s autorizáciu ostne zachovaná.

Z vyššie uvedeného popisu vyplýva, že tento modul vyžaduje z hľadiska bezpečnosti prístupu k citlivým funkciám nasledujúce úpravy:

- Zmena prihlasovania
- Overenie používateľa voči centrálnej používateľskej databáze, ktorá je súčasťou modulu AdministrationPortal, ktorý bol vytvorený v rámci tohto tímového projektu (Autentifikácia)
- Autorizácia používateľa – taktiež prebieha za pomoci administračného portálu (Volaním Rest služby), v prípade, že ja používateľ v roli „admin“ pre tento modul, potom je úspešne autorizovaný ak je v roli „read/reader“ potom vstupuje do procesu autorizácie lokálna kontrola práv používateľov k repozitárom, ak je v roli „deny“ automaticky je mu zablokovaný prístup k repozitárom.
- Autorizácia API volaní, neautorizované requesty na API nebudú akceptované a vyústia do not authorized chyby
- Prístupové práva reprezentované prostredníctvom bitovej enumerácie

3 Návrh riešenia

V nasledujúcej časti bude opísané, ako boli navrhnuté jednotlivé úpravy, ktoré boli opísané v časti analýza.

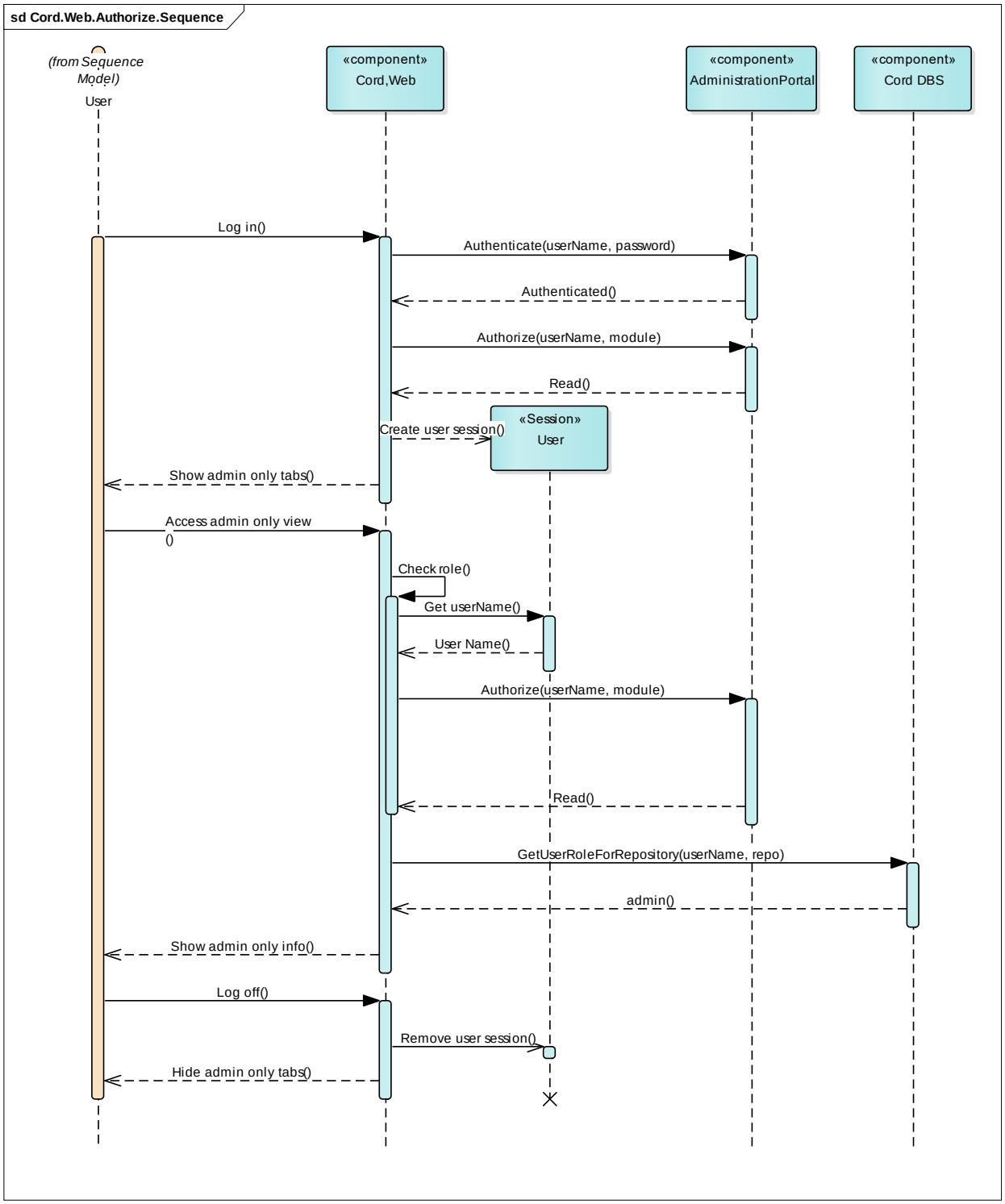
3.1 Návrh prihlasovania a autentifikácie

Pri návrhu prihlasovania do webového portálu Cord.Web bolo nutné brať do úvahy, že tento webový portál je súčasťou architektúry, ktorá vyžaduje autentifikáciu a autorizáciu na rôznych miestach naprieč rôznymi modulmi. Bolo potrebné rozhodnúť akým spôsobom bude prebiehať registrácia používateľov. Najvhodnejším riešením bolo implementovať centrálnu správu používateľov, ktorú budú využívať všetky moduly, čím sa zabezpečí zníženie množstva uložených dát, množstvo kódu potrebného na registráciu a overenie, ... Preto bolo prihlasovanie v rámci tohto modulu navrhnuté tak, aby sa využívalo vytvorené Rest rozhranie pre autentifikáciu a autorizáciu, ktoré sa nachádza v module AdministrationPortal. Proces prihlásenia bol navrhnutý tak, že používateľ zadá prihlasovacie meno a heslo a následne sa vytvorí request a volá sa príslušná Rest metóda. Na základe odpovede sa zistí či je používateľ autentifikovaný, v prípade neúspechu je súčasťou odpovede aj chybová správa, ktorá hovorí o tom, kde nastala chyba v procese autentifikácie.

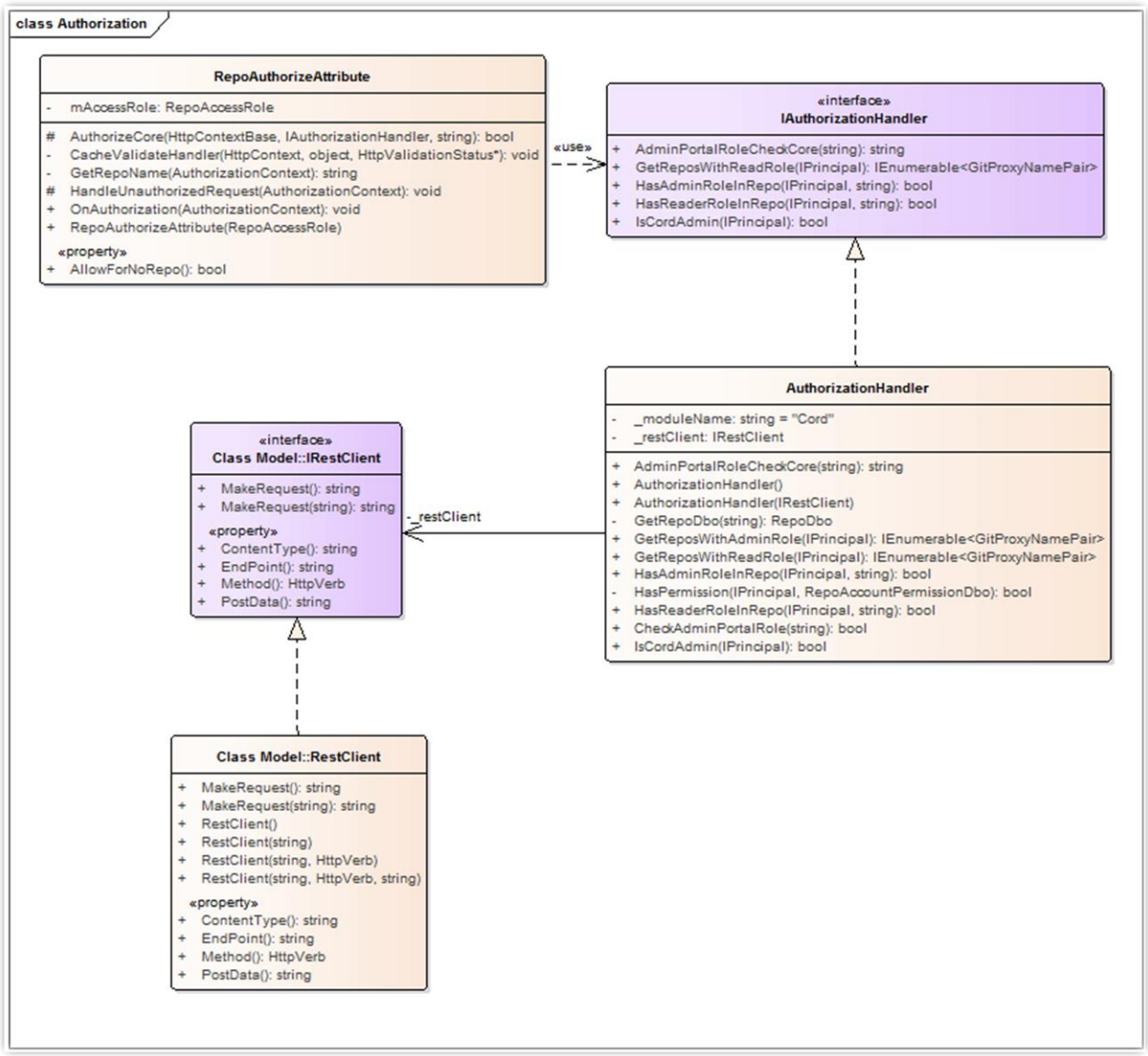
3.2 Návrh autorizácie

Autorizácia prebieha podobne ako autentifikácia pomocou volania príslušnej Rest metódy modulu AdministrationPortal. Vstupom tejto Rest metódy je meno aktuálne prihláseného používateľa a názov modulu – CORD, Metóda vracia v odpovedi rolu aktuálne prihláseného používateľa k modulu CORD, ktorá bola nastavená administrátorom Administračného portálu. V prípade, že je používateľ v roli admin autorizácia prebehla úspešne a používateľ môže pristupovať k zabezpečeným funkciám tohto modulu, v prípade, že je používateľ v roli „read“, vstupuje do procesu autorizácie kontrola lokálnych rolí nastavených v tomto module pre jednotlivé repozitáre, v prípade, že je používateľ v roli „denied“ autorizácia prebehla neúspešne. Dôležitým rozhodnutím pri navrhovaní autorizácie bolo určiť, ako často resp. v akom momente má prebiehať autorizácia. V prípade, že by sme sa rozhodli navrhnúť, aby autorizácia prebiehala iba v čase prihlasovania používateľa na portál, mohlo by dôjsť k neželanej situácii, kedy by administrátor Administračného portálu zmenil používateľskú rolu prihlásenému používateľovi a on by nestratil prístup k citlivým funkciám až do doby kým by sa odhlásil. Preto sme sa rozhodli pre druhú alternatívu, kedy autorizácia používateľa prebieha pri každej požiadavke k citlivým funkciám systému, toto riešenie sa nám zdá v súčasnosti vhodnejšie, pretože sa dokáže jednoducho vysporiadať s popísaným problémom, avšak môže sa stať, že dôjde k úpravám tohto riešenia, v prípade, že by výkonnostne nedosahovalo dobré výsledky. Na obrázku 1 sa nachádza sekvenčný diagram navrhnutého riešenia interakcie Cord.Web komponentu a Administračného portálu. Sekvenčný diagram však zachytáva iba hlavnú cestu vykonávania, kedy je daný používateľ v roli

readera a zároveň nenastala žiadna chyba počas autorizácie a autentifikácie. A má v databáze Cordu uložené admin práva k danému repozitáru.



Obr. 3.2. A Diagram interakcie komponentov Cord.Web a Administartion Portal počas autentifikácie a autorizácie (Model je uložený v zdieľanom úložisku EA pre DevActs)

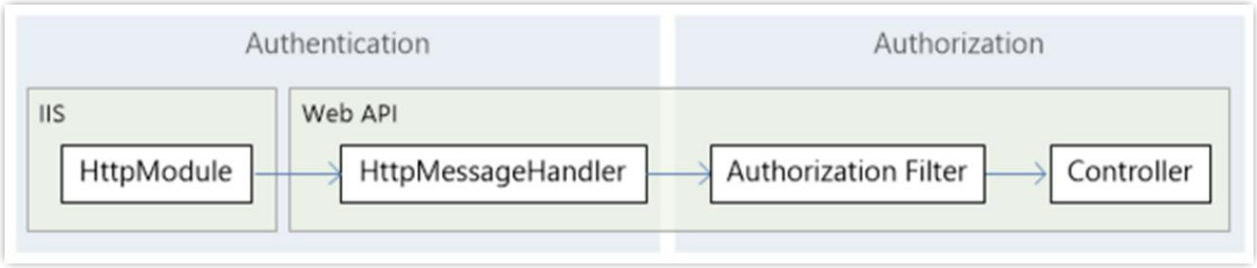


Obr. 3.2. B Class diagram – pridané časti v Cord.Web (Model je uložený v zdieľanom úložisku EA pre DevActs)

Na obrázku 2 ja zachytený navrhnutý diagram tried, ktorý zachytáva triedy potrebné pre realizáciu autorizácie a autentifikácie v rámci Cord. Rozhranie IRestClient bolo navrhnuté pre zvýšenie testovateľnosti pridanej funkcionality.

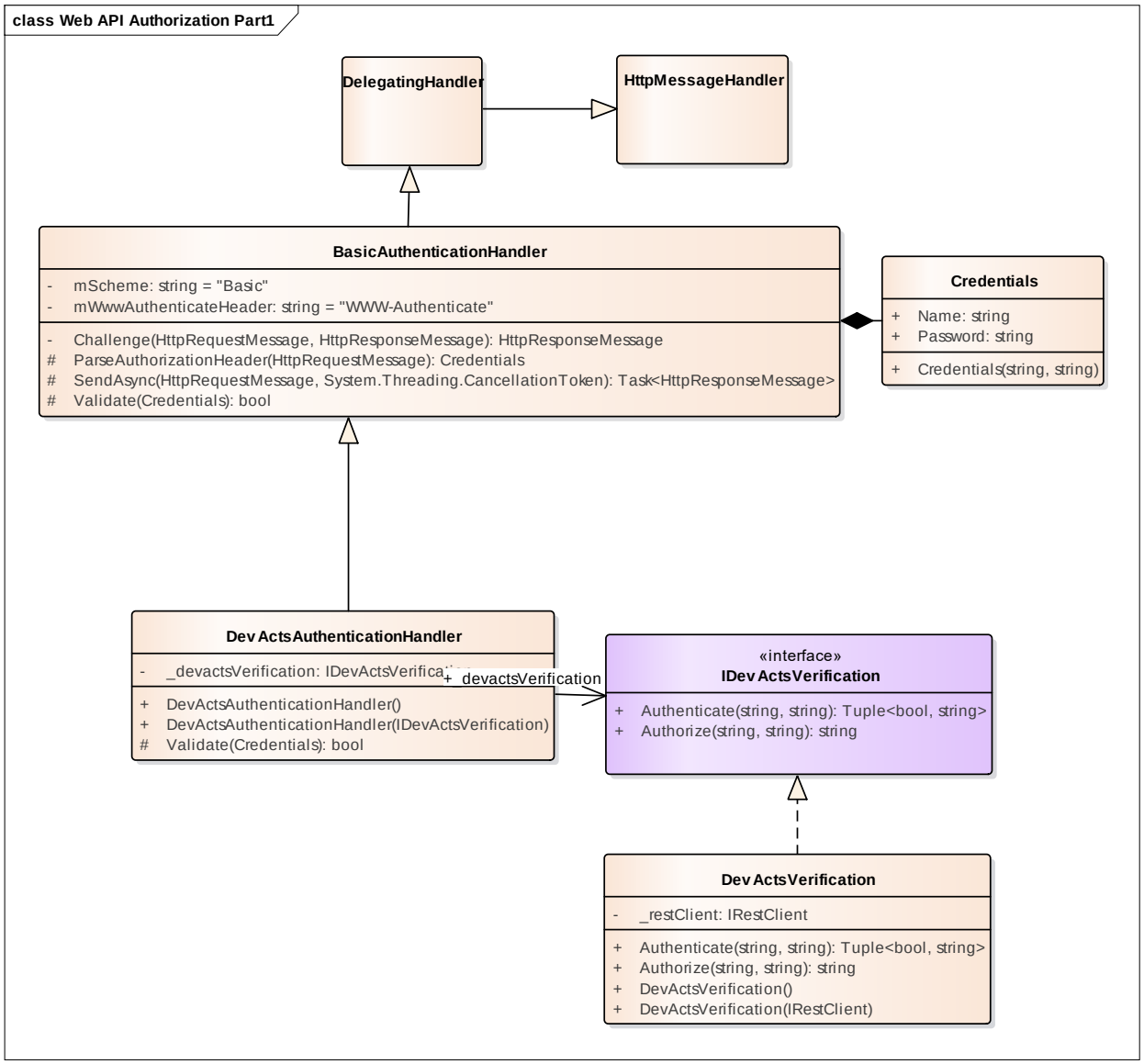
3.3 Návrh autorizácie API

Na autorizáciu API volaní sú použité štandardné mechanizmy, ktoré poskytuje technológia .Net WEB API a teda autorizačný MessageHandler, ktorý je súčasťou autorizačnej pipeline obrázok 3.3 A, tento handler z hlavičky HTTP požiadavky prečíta meno a heslo používateľa, ktoré autentifikuje voči centrálnemu portálu. Vzhľadom na to že v rámci Architektúry projektu PerConIK sa tento handler používa na viacerých miestach už bola vytvorená jeho základná implementácia z ktorej bude *DevActsMessageHandler* dediť a prekoná metódu Validate, v ktorej dochádza k samotnej autentifikácii na základe mena a hesla, V prípade, že autentifikácia prebehne úspešne do *Thread.CurrentPrincipal* a *Http.Current.User* je priradený nový generický principál s daným menom, ktoré sa neskôr využíva na autorizáciu. K samotnej autorizácii dochádza až v triede *ApiRepoAuthorizeAttribute*, ktorá funguje analogicky ako *RepoAuthorizeAttribue*, ktorá bola zdokumentovaná v predchádzajúcej kapitole.

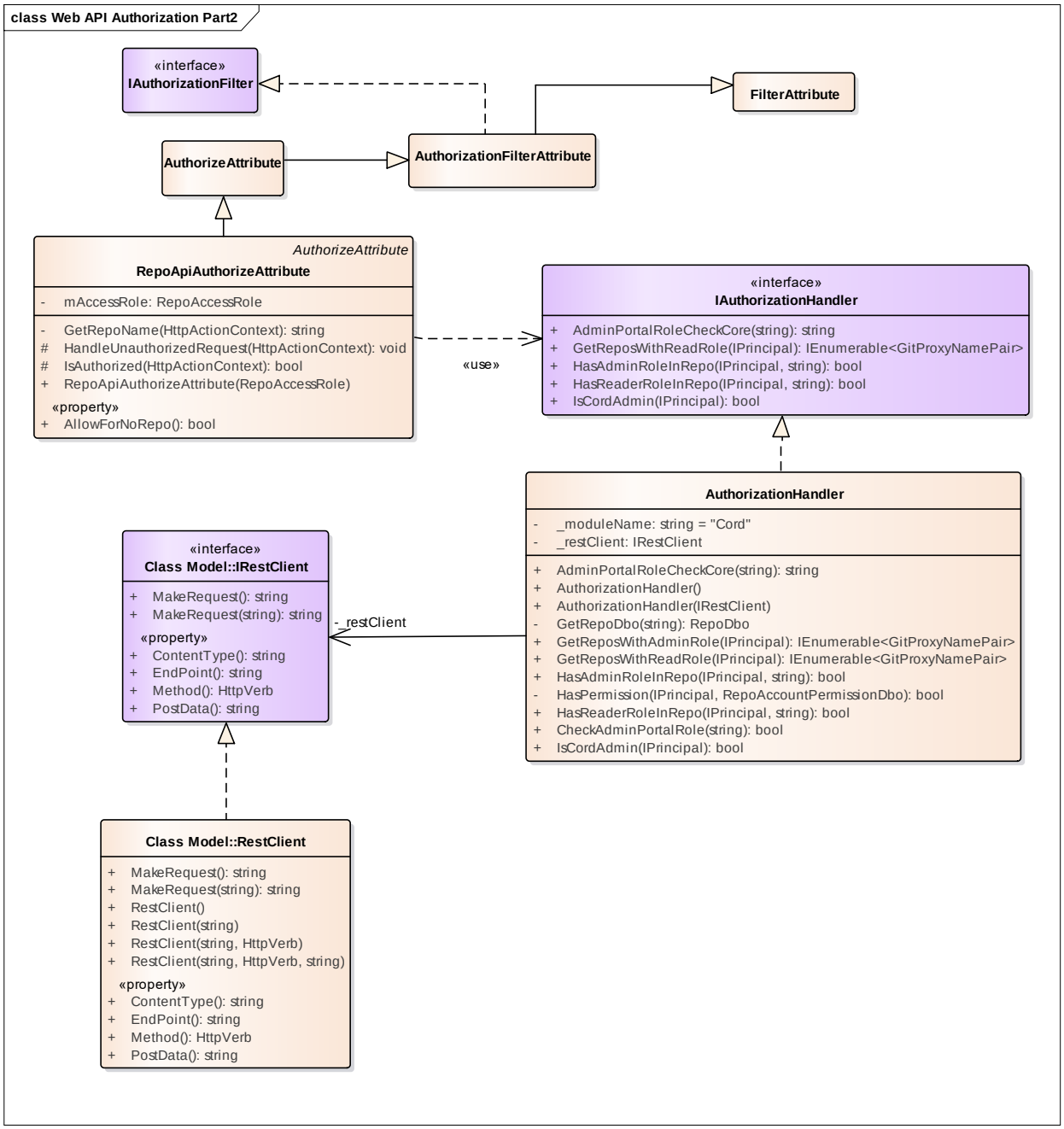


Obrázok 3.3 A Autorizačná pipeline Web API

Na obrázku 3.3 B je zachytená prvá časť diagramu tried pre autorizáciu. Triedy *DelegatingHandler* a *HttpMessageHandler* obsahujú mnohé metódy a vlastnosti, ktoré nie sú na obrázku zachytené, tieto triedy sú súčasťou .Net frameworku a boli pridané, na diagram, aby bolo jasné, ktorú časť pipeliny z obrázka 3.3 A tvorí daná hierarchia tried. Podobne je to s triedami *AuthorizationAttribute*, *AuthorizationFilterAttribute*, *FilterAttribute* a rozhraním *IAuthorizationFilter* z obrázka 3.3 C.



Obr. 3.3 B Web API diagram tried autorizačná pipeline 1



Obr. 3.3 C Web API diagram tried autorizačná pipeline 2

3.4 Návrh efektívnej reprezentácie prístupových rolí

Keďže prístupové roly sa získavajú prostredníctvom rest volania administračného portálu, ktoré na výstupe vracia stringovú reprezentáciu roly je potrebné navrhnúť efektívnejšiu reprezentáciu, prostredníctvom ktorej by sme sa vyhli nepríjemným chybám spojeným so stringami najmä preklepom a zároveň dokázali využívať automatické dopĺňanie vývojového prostredia (intellisense pre VS) pri manipulácii s používateľskými právami, najvhodnejšou reprezentáciou je bitová enumerácia, ktorá umožňuje vytvárať práva kombinovaním viacerých prvkov enumerácie a zároveň zjednodušuje overovanie práv. Rozhodli sme sa využiť nasledovné práva:

None = 0,
Reader = 1,
Writer = 2,
Admin = Reader | Writer

Rola Writer je iba virtuálna rola ale dôležitá pretože zjednodušuje overenie práv. Príkladom je overenie či je používateľ v roli „reader“ pri stringovej reprezentácii by sme museli explicitne overiť či je používateľ v roli „reader“ alebo „writer“ enum reprezentácia nám umožňuje vykonať nasledujúce overenie:

(userAccessRole & accessRole) == accessRole;

Daný zápis má nasledujúcu slovnú reprezentáciu: ak právo používateľa v bitovom súčine s požadovaným právom vracia požadované právo potom je používateľ oprávnený pre vykonanie funkcie. Zapísané matematicky pre jednotlivé roly:

(00 & 1) == 0 - používateľ je v roli None požaduje sa Read
(01 & 1) == 1 - používateľ je v roli Read požaduje sa Read
(11 & 1) == 1 - používateľ je v roli Admin požaduje sa Read

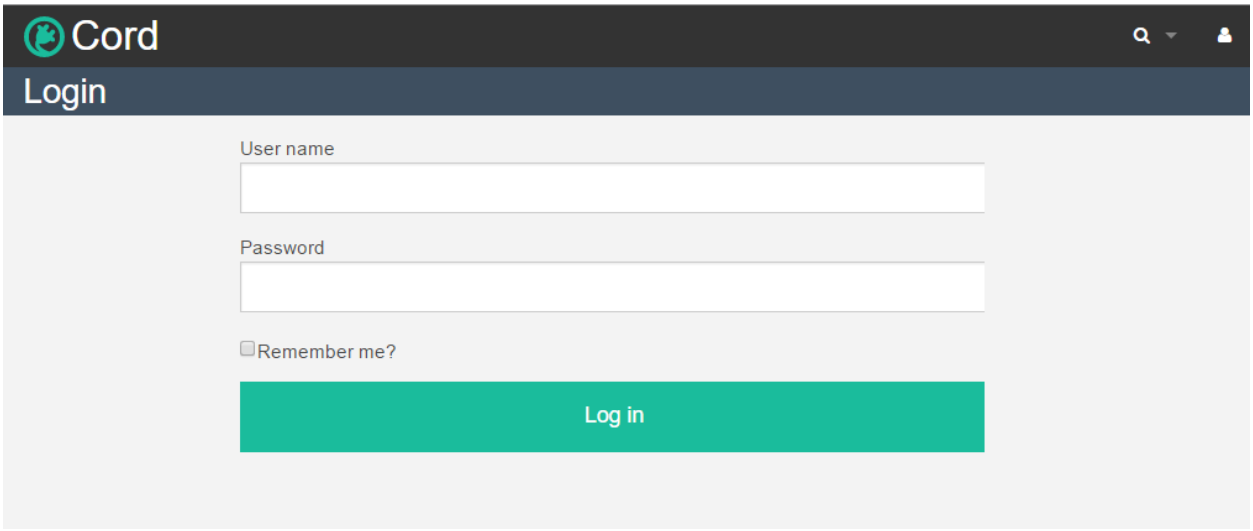
Ako je vidieť výsledok autentifikácie je 1 aj pre rolu read aj pre admin.

4 Implementácia

Implementácia navrhnutej funkcionality bola realizovaná podľa návrhu v nasledujúcej kapitole bude detailnejšie opísané niektoré dôležité implementačné detaily a rozhodnutia, ktoré k nim viedli.

4.1 Implementácia prihlasovania a autentifikácie

Autentifikácia bola implementovaná tak, ako bolo navrhnuté. Pre prihlasovanie sme sa rozhodli využiť FormsAuthentication vzhľadom na to, že takýto typ prihlasovania sa využíva už v iných webových projektoch v rámci architektúry PerConIK napr. v CodeReview webovom projekte, chceli sme zachovať konzistentnosť v rámci architektúry. Pre dočasné ukladanie údajov o prihlásenom používateľovi sme využili Session, ktorá je súčasťou .Net MVC frameworku, čo značne uľahčilo implementáciu celkovej funkcionality. Na obrázku 3 sa nachádza formulár pre prihlásenie do systému.



Obr. 4.1 Prihlasovací formulár

4.2 Implementácia autorizácie

Pre implementáciu autorizácie bolo potrebné doplniť a upraviť funkcionalitu triedy *AuthorizationHandler* a upraviť *IAuthorizationHandler*. Pre zavádzanie tejto triedy sa používa IOC (angl. Inversion of control container) *WindsorInstaller*, čo spôsobovalo spočiatku problémy so *Session*, tá neostávala perzistentná na vyriešenie tohto problému sme využili *SessionRouteHandler* a IoC kontajner nemohol zavádzať triedu ako singelton ale transient. Ako vyplýva z diagramu tried v návrhu *RepoAuthorizationAttribute* už využíval rozhranie

IAuthorizationHandler a preto sa zmeny v triede *AuthorizationHandler* priamo dotkli autorizácie. Kľúčová pridaná funkcionalita sa nachádza na obrázku 4.2. Ako je vidieť princíp autorizácie ostal konzistentný s prístupom v ostatných moduloch a rozšírený o ďalšiu funkcionalitu potrebnú pre zabezpečenie modulu Cord.

```
public bool CheckAdminPortalRole(string role)
{
    if(role == "reader")
    {
        role = "read";
    }

    if (HttpContextManager.Current.User.Identity.IsAuthenticated)
    {
        var user = HttpContextManager.Current.Session["User"] as IUser;

        if (user == null) return false;

        var userRole = AdminPortalRoleCheckCore(user.Name);

        if (string.IsNullOrEmpty(userRole)) return false;

        // if user role changed update Session["User"] with new role
        if (userRole != user.Role)
        {
            user.Role = userRole;
            HttpContextManager.Current.Session["User"] = user;
        }

        if (userRole == role || userRole == "admin") return true;
    }
    return false;
}
```

Obr. 4.2 Ukážka kódu triedy *AuthorizationHandler* slúžiaceho na autorizáciu voči admin portálu

4.3 Implementácia autorizácie Web API

Autorizácia bola implementovaná tak ako je opísané v návrhu na obrázku 4.3 sa nachádza ukážka metódy Validate, ktorá bola prekonaná zvyšok funkcionality bol prebratý z nadtriedy. Trieda *DevActsAuthenticationHandler* sa nachádza v Core.Mvc.Security.

```
protected override bool Validate(Credentials credentials)
{
    return _devactsVerification.Authenticate(credentials.Name, credentials.Password).Item1;
}
```

Obr. 4.3 Metóda Validate v DevActsAuthorizationHandler

4.4 Implementácia efektívnej reprezentácie prístupových rolí

Implementácia prebehla podľa návrhu na obrázku 4.4 možno vidieť funkciu ktorá vykonáva mapovanie stringovej reprezentácie roly na enumeráciu a priamo vykonáva overenie voči poskytnutej roli.

```
public static bool CheckPermissionFromString(this AccessRole acessRole, string role)
{
    switch (role)
    {
        case "denied":
            return acessRole.CheckPermission(AccessRole.None);
        case "read":
            return acessRole.CheckPermission(AccessRole.Reader);
        case "admin":
            return acessRole.CheckPermission(AccessRole.Admin);
        default:
            throw new NotSupportedException();
    }
}

public static bool CheckPermission(this AccessRole accessRole, AccessRole userAccessRole)
{
    return (userAccessRole & accessRole) == accessRole;
}
```

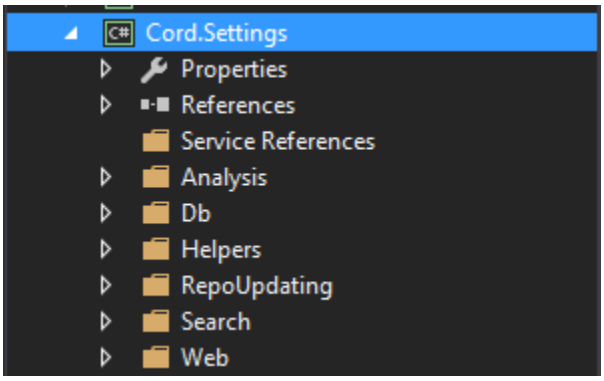
Obr. 4.4 Implementácia mapovania stringovej reprezentácie roly na enumeráciu

5 Konfigurácia Cord modulov

5.1 Implementácia

Aby bol Cord modul jednoducho konfigurovateľný, bolo nutné presunúť konfigurácie z konfiguračných XML (app.config/web.config) na nový systém, ktorý bude konfigurácie získavať z administračného portálu.
Aby boli tieto konfgurácie zjednotené, vytvoril sa knižničný projekt s názvom ***Cord.Settings***.

Štruktúra tohoto projektu je tvorená adresármi, kde v každom koreňovom adresáry nájdeme konfigurácie pre knžnice, ktoré Cord využíva. Ukážku môžeme vidieť na obrázku č. 5.1.



Obr. 5.1 Ukážka štruktúry projektu Cord.Settings

V týchto adresároch sa nachádzajú triedy, ktoré dedia od triedy BaseSettings zadefinovanej v knižnici Core.CentralServices. Dokumentáciu k tejto triede nájdete v dokumentácie ku modulu Core.

Každá trieda je namodelovná ako singleton, kde pre každú konfiguračnú hodnotu obsahuje privátnu premennú, ktorá bude držať hodnotu konfigurácie. Na začiatku, je táto hodnota nastavená na null, kde pri prvom zavolaní prislúchajúcej property sa zistí jej hodnota cez dopyt na REST rozhranie Administračneho portálu. Opakovaný dopyt na danú property už nespôsobí opätovný dopyt, pretože premenná už nebude null a v takom prípade nám metóda GetValue<T> vráti naspäť tú istú hodnotu. Toto cachovanie nám pomôže znižiť traffic load pri volaní konfigurácií. Na obrázku č. 5.2 môžeme vidieť ukážku konfiguračnej triedy.

```
6 references | Ivan, 53 days ago | 1 author, 2 changes
public class GlobalDbConfig : BaseSettings
{
    private static GlobalDbConfig instance = null;

    1 reference | Ivan, 64 days ago | 1 author, 1 change
    protected GlobalDbConfig(IRestClient client) : base(client) { }

    1 reference | Ivan, 64 days ago | 1 author, 1 change
    public static GlobalDbConfig GetInstance()
    {
        if (instance == null)
        {
            instance = new GlobalDbConfig(new RestClient());
        }
        return instance;
    }

    private ConfigProperty<bool> _enableRepoDboCache = null;

    2 references | 1/1 passing | Ivan, 53 days ago | 1 author, 2 changes
    public bool EnableRepoDboCache
    {
        get
        {
            _enableRepoDboCache = GetValue<bool>(_enableRepoDboCache, "Db.EnableRepoDboCache", ModuleNameEnum.Cord);
            return _enableRepoDboCache.Value;
        }
    }
}
```

Obr. 5.2 Ukážka konfiguračnej triedy pre Cord.Db knižnicu.

5.2 Testovanie konfigurácií

Všetky konfiguračné hodnoty vo všetkých konfiguračných triedach sú otestované v module ***Cord.Test***, kde štruktúra umistnenia testovacích tried a adresárov zodpovedá umiestneniu v knižnici ***Cord.Settings***.

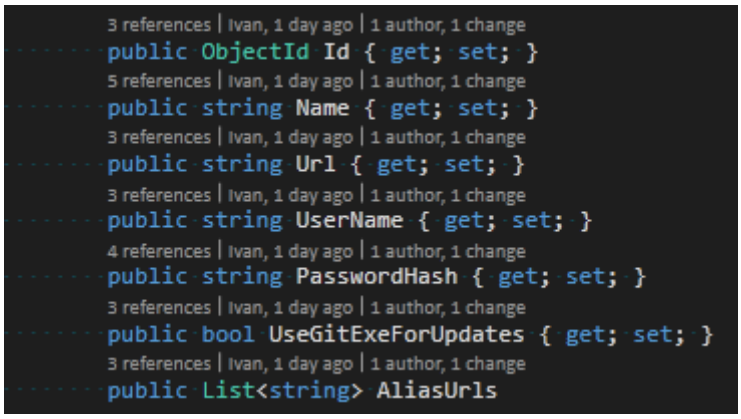
Pre jednoduchšie testovanie bola implementovaná generická templatová trieda, ktorá od ktorej dedí väčšina testovacích tried.

6 Nastavenia Git repozitárov pre Cord

Pôvodne využíval Cord na konfiguráciu git repozitárov XML súbor s názvom config.xml, ktorý sa deserializoval pri štarte aplikácie a získali sa z neho konfigurácie. Keďže bol tento spôsob nepraktický, bolo implementované grafické rozhranie pre prístup, a XML bolo transformované do MongoDb databázy.

6.1 Implementácia Db API

Pre uloženie konfigurácií bola vytvorená kolekcia s názvom “GitRepoSettings“ na ktorej bol pridaný unique index na názov repozitára. Štruktúra bola zachovaná ako v pôvodnom XML dokumente, jej implementáciu môžeme vidieť na obrázku č. 6.1.



Obr. 6.1 Ukážka triedy (GitRepoSettingDbo), ktorá reflektuje štruktúru kolekcie v databáze.

Na manipuláciu s touto kolekciou boli vytvorené metódy :

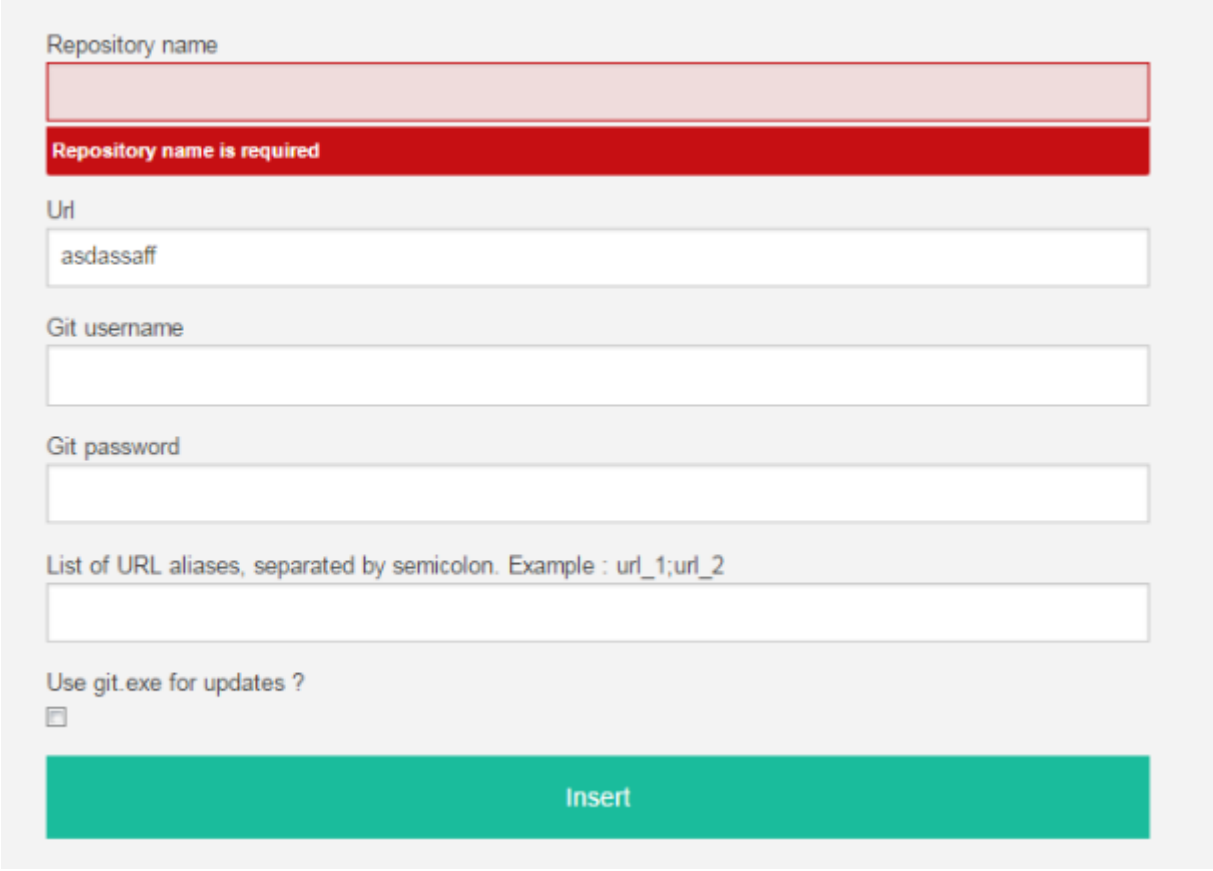
```
1. public string SaveGitRepoConfig(GitRepoSettingDbo repoConfig)
2. public void RemoveGitRepoConfig(string repoConfigId)
```

Prvá metóda updatuje alebo vkladá novú git konfiguráciu repozitára do databázy, kde druhá ju umožnuje vymazať.

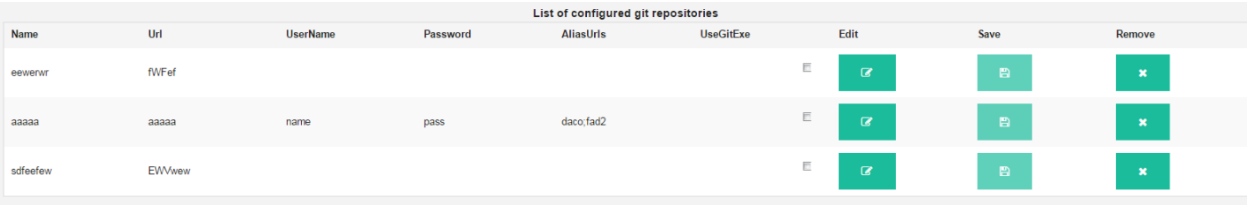
6.2 Implementácia grafického rozhrania

Pre grafické rozhranie vznikol nový view v adresáry GitReposConfig. Jeho prislúchajúci controller, z ktorého sa volajú a validujú operácie nad konfiguráciami sa nazýva GitReposConfigController. Tento view je autorizovaný a teda prísupný iba administrátorom Cordu po prihlásení.

Implementáciu grafického rozhrania môžeme vidieť na obrázku č. 6.2 a 6.3. Ako jedinné povinné hodnoty, ktoré musia byť zadané pri vložení alebo updatnutí záznamu sú hodnoty pre názov repozitára a jeho Url. Ostatné parametre sú nepovinné.



Obr. 6.2 Ukážka GUI pre vloženie novej konfigurácie git repozitára.



Obr. 6.3 Ukážka vykreslenia zoznamu konfigurácií pre git repozitáre (pre každý riadok tabuľky je povolená operácia [EDIT, UPDATE, DELETE])

7 Testovanie

Boli vytvorené testy pre overenie správnej funkcionality triedy *AuthorizationHandler*, na testovanie sa využíva framework *Moq* a unit testový projekt s názvom *Cord.Web.Test*. Všetky testy prebehli v jednej testovacej metóde *AdminPortalRoleCheckCoreTest* v triede *AuthorizationHandlerTest*. Pre testovanie WEB API autorizácie bola vytvorená trieda *DevActsAuthenticationHandlerTest*, ktorý sa nachádza v projekte Core.Test.