Technická dokumentácia pre modul UserActivity.Web

Verzia 14.11.2015

Tabuľka 1. Autori

Autor	Rola
Peter Vrana	

Tabuľka 2. História zmien

Verzia	Dátum	Autor	Popis
1.0	14.11.2015	Peter Vrana	Vytvorenie dokumentu
1.1	24.11.2015	Peter Halaš	Upravenie dokumentu
1.2	29.11.2015	Peter Vrana	Úprava dokumentu – refactoring presunutie autentifikácie a autorizácie do Core.CentrakServices
1.3	30.3.2016	Peter Vrana	Úprava volaní služieb na autentifikáciu a autorizáciu

Obsah

1 Úvod

V dokumente sú opísané zmeny modulu UserActivity.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 informačný web portál na ktorom môže používateľ získať bližšie detaily o projekte PerConIK. Na tejto stránke sa nachádzajú manuály ako si nainštalovať klientskú aplikáciu a rozšírenia vývojových prostredí či webových klientov. Súčasťou stránky je aj referenčný manuál k projektu PerConIK. Problémom tohto portálu je, že neautentifikovaný a neautorizovaný používateľ má voľný prístup k citlivým funkciám systému ako napríklad: monitorovanie stavu používateľov alebo generovanie reportov o zaznamenaných aktivitách.

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

- Zavedenie prihlasovania v rámci webového portálu
- 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)
- Zmena rozhrania – skrývanie kariet, ku ktorým má prístup iba používateľ s rolou administrátor, pridanie potrebných prvkov pre prihlasovanie používateľov
- Vystavená Rest služba, ktorá ukladá udalosti z klientskej aplikácie do databázy, nijakým spôsobom neautorizuje ani neautentifikuje používateľa, ktorý posiela udalosti.
- Vystavené rest služby, ktoré vracajú udalosti uložené v databáze podľa rôznych filtrov rovnako neautorizujú ani neautentifikujú či používateľ má právo prístupu k týmto údajom.

3 Návrh riešenia

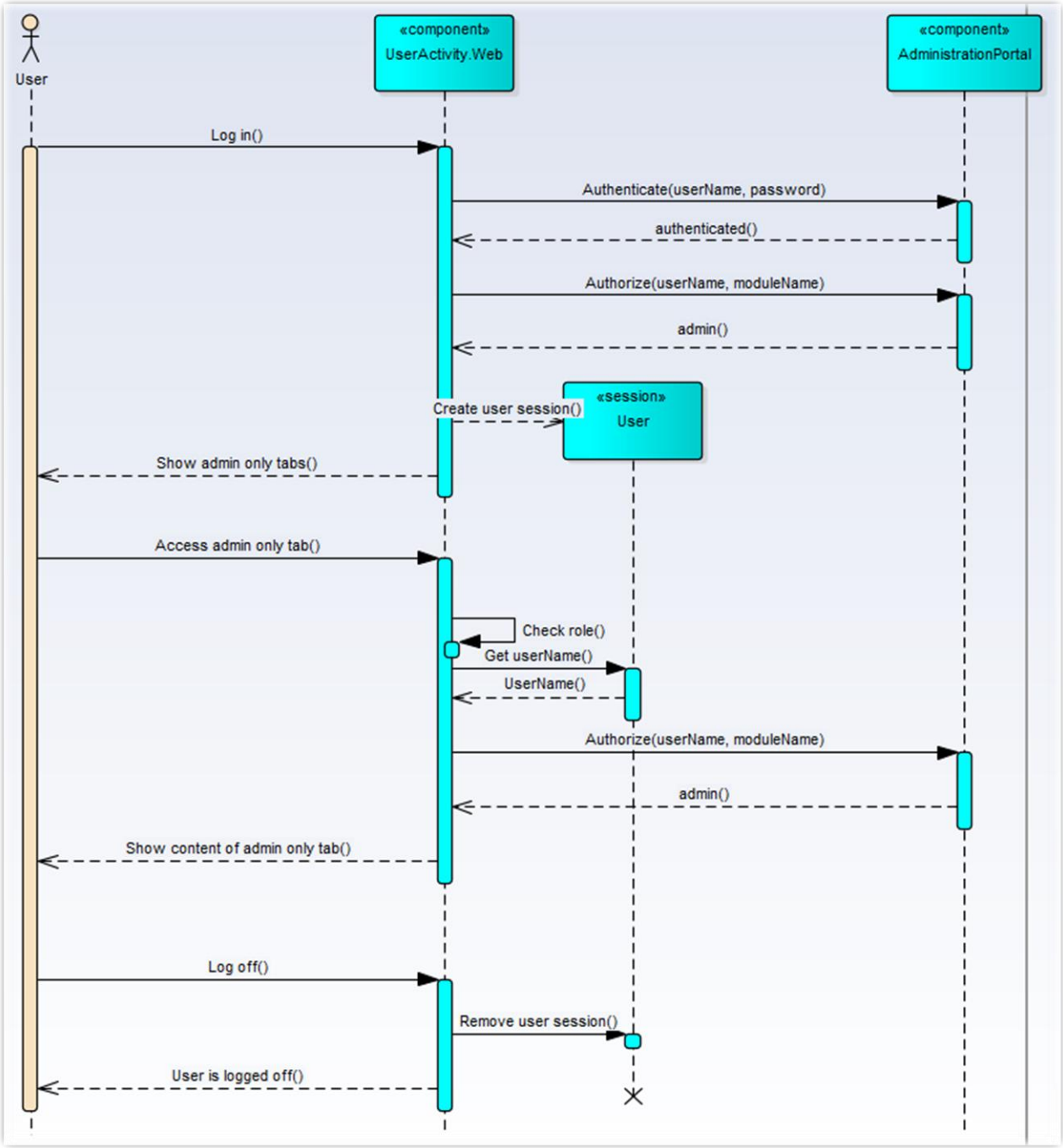
V nasledujúcej časti bude opísané, ako boli navrhnuté jednotlivé úpravy, ktoré boli opísané v časti analýza.

3.1 Návrh prihlasovania a autentifikácie

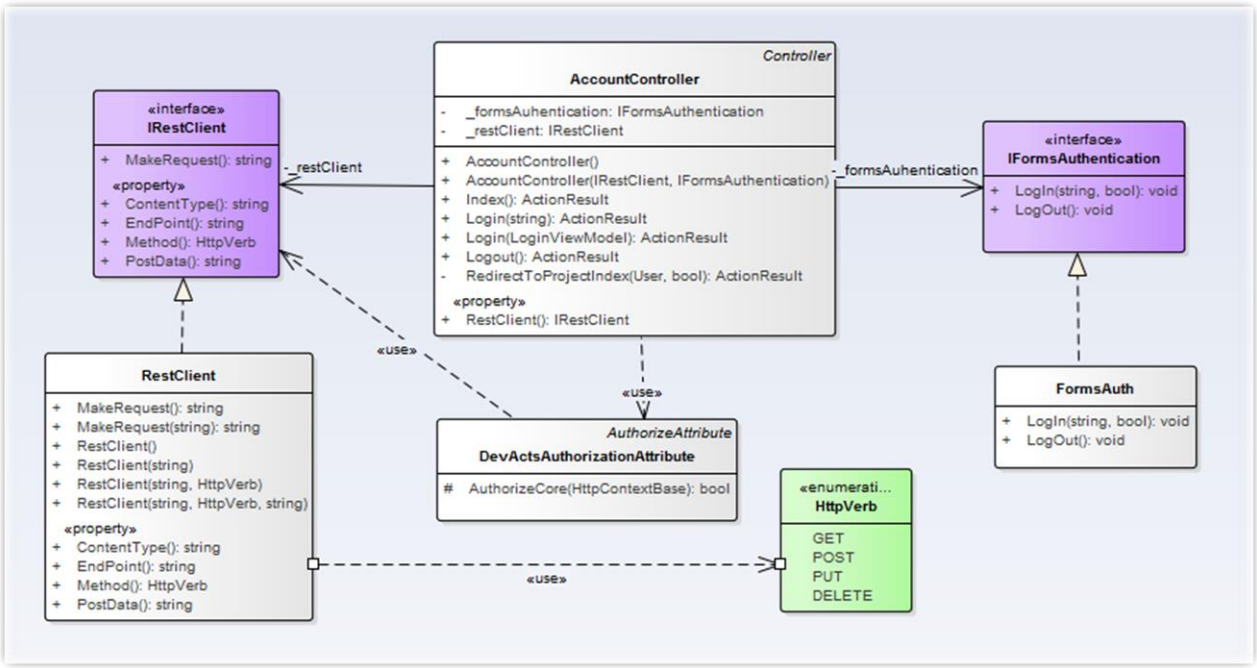
Pri návrhu prihlasovania do webového portálu User Activity bolo nutné brať do úvahy, že tento webový portál je súčasťou architektúry, ktorá vyžaduje autentifikáciu a autorizáciu na rôznych miestach naprieč rôznymi modulmi. Bolo potrebné rozhodnúť akým spôsobom bude prebiehať registrácia používateľov. 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 – UserActivity, Metóda v vracia v odpovedi rolu aktuálne prihláseného používateľa k modulu UserActivity, ktorá bola nastavená administrátorom Administračného portálu. 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 UserActivity.Web komponentu a Administračného portálu. Sekvenčný diagram však zachytáva iba hlavnú cestu vykonávania, kedy je daný používateľ v roli administrátora a zároveň nenastala žiadna chyba počas autorizácie a autentifikácie.



Obr. 1 Diagram interakcie komponentov User Activity a Administartion Portal počas autentifikácie a autorizácie



Obr. 2 Class diagram – pridané časti v UserActivity.Web

Na obrázku 2 ja zachytený navrhnutý diagram tried, ktorý zachytáva triedy potrebné pre realizáciu autorizácie a autentifikácie v rámci UserActivity. Rozhrania IRestClient a IFormsAuthentication boli navrhnuté pre zvýšenie testovateľnosti pridanej funkcionality.

3.3 Návrh zabezpečenia vystavených Rest služieb

Rest službu na ukladanie udalostí z klientskej aplikácie je potrebné zabezpečiť tak, aby:

- Názov používateľa, ktorý sa uvádza v hlavičke *requestu* rovnal názvu používateľa v tele udalosti
- Názov a heslo používateľa sa bude posielať cez hlavičku *requestu* na to určenú.
- Sa používateľa podarilo autentifikovať voči Administračnému portálu
- Používateľ mal akékoľvek iné právo ako „denied“

Ostatné Rest služby, ktoré vracajú informácie o udalostiach je potrebné zabezpečiť tak, aby:

- Sa používateľa podarilo autentifikovať voči Administračnému portálu
- Používateľ mal právo „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. Dizajn tohto formulára nie príliš graficky atraktívny, keď berieme do úvahy súčasné dizajnové trendy, avšak dizajn bol prispôsobený tak, aby bol konzistentný s celkovým dizajnom webového portálu.

Obr. 3 Prihlasovací formulár

4.2 Implementácia autorizácie

Pre implementáciu autorizácie sme sa rozhodli využiť tak isto funkcionalitu, ktorú nám poskytuje .Net framework. Autorizácia je teda implementovaná prostredníctvom autorizačných atribútov, pomocou ktorých sú dekorované všetky chránené funkcie webového portálu, keďže sa nejedná o štandardné využitie týchto atribútov museli sme implementovať vlastné prispôsobenie tridy AuthorizeAttribute a preťažiť metódu AuthorizeCore, ktorá je volaná pri požiadavke pre prístup k chránenej funkcionalite. Na obrázku 4 sa nachádza samotná implementácia tejto metódy. Pri

presune triedy DevActsAuthorizationAttribute do Core.CentralServices došlo k dvom významným zmenám tejto triedy: trieda príma ako parameter v konštruktore názov modulu pre ktorý sa vyžaduje autorizácia a používateľ uložený v Session sa castuje na rozhranie IUser, ktoré má len dve vlastnosti a to meno a rola, trieda User implementuje toto rozhranie.

```
protected override bool AuthorizeCore(HttpContextBase httpContext)
{
    if (httpContext.User.Identity.IsAuthenticated)
    {
        var user = httpContext.Session["User"] as IUser;

        if (user == null) return false;

        // Always use rest service for authorization
        if (_restClient != null)
        {
            _restClient.EndPoint = Settings.Default.DevActsEndpoint + "api/users/authorize";
            _restClient.PostData = new JavaScriptSerializer().Serialize(new AuthorizationRequest() { UserName = user.Name, Module = _moduleName });
            _restClient.Method = HttpVerb.POST;
            _restClient.ContentType = "application/json; charset=UTF-8";
        }
        else
        {
            return false;
        }

        var authorizationResult = _restClient.MakeRequest(Encoding.UTF8);
        var authorizationResponse = new JavaScriptSerializer().Deserialize<AuthorizationResponse>(authorizationResult);

        string userRole = authorizationResponse.Role;

        if (userRole == null) return false;

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

        return Roles.Split(',').Contains(userRole);
    }
    return false;
}
```

Obr. 4 Ukážka kódu zabezpečujúceho overenie používateľa pri každej požiadavke chránenej funkcionality

4.3 Implementácia zabezpečenia rest služieb

Zabezpečenie využíva už existujúce služby pre autentifikáciu a autorizáciu používateľa. Implementované sú tieto metódy:

```
private void AuthentificateUserDevAct()
```

Táto metóda autentifikuje používateľa voči Administračnému portálu, pričom údaje o používateľovi získa z hlavičky *requestu*. Ak nejaký údaj chýba alebo sa používateľa nepodarí autentifikovať vyhodí chybu.

```
private void AuthorizeUserAsAdmin()
```

Táto metóda zistí, aké má používateľ práva pre modul „UserActivity“, pričom údaje o používateľovi získa z hlavičky *requestu*. Ak nejaký údaj chýba alebo používateľ nemá administrátorské právo („admin“) vyhodí chybu.

```
private void AuthorizeUserForPut()
```

Táto metóda zistí, aké má používateľ práva pre modul „UserActivity“, pričom údaje o používateľovi získa z hlavičky *requestu*. Ak nejaký údaj chýba alebo používateľ nemá administrátorské právo („admin“) alebo právo na čítanie („read“) vyhodí chybu.

```
private void CompareNames(string body)
```

Táto metóda porovná používateľské meno v obsahu správy s menom v autorizačnej hlavičke, ak sa nerovnajú vyhodí chybu.

5 Testovanie

Boli vytvorené unit testy, ktoré testujú či sa AccountController správa korektne pri prihlasovaní používateľa a tak isto pri jeho odhlásení z webového portálu. Unit testy k webovému portálu sa nachádzajú v projekte UserActivity.Web.Test. Vytvorené testy pre AccountController sa nachádzajú v testovacej triede s názvom AccountControllerTest.cs.

Boli vytvorené unit testy, ktoré testujú pridané metódy v UserActivityController, teda či správne zabezpečili existujúce Rest služby.