Slovenská technická univerzita

Fakulta informatiky a informačných technológií

Ilkovičova 2, 842 16 Bratislava 4

Prehliadka kódov v tímových projektoch

(CodeReview)

Dokumentácia k inžinierskemu dielu

Tím: Lucky Seven

Vedúci projektu: Ing. Karol Rástočný

Kontakt:*tp.1314.07@gmail.com*

Ak. Rok : 2013/2014

Autori: Bc. Zuzana Grešlíková

Bc. Matej Chlebana

Bc. Tomáš Kepič

Bc. Patrik Oriskó

Bc. Patrik Samuhel

Bc. Michael Scholtz

Bc. Július Skrisa

OBSAH

1 ÚVOD

Tento dokument obsahuje technickú dokumentáciu k projektu CodeReview implementovanom tímom č. 7 na predmete tímový projekt 1 na Fakulte informatiky a informačných technológií STU v Bratislave.

1.1 ŠTRUKTÚRA DOKUMENTU

Vývoj je riadený metódou „scrum“, preto sa dokument sa skladá z kapitol, ktoré opisujú jednotlivé šprinty. V každom šprinte sú opísané funkcie („user stories“), ktoré boli v rámci neho implementované. Jednotlivé funkcie sú rozdelené na časti: Analýza, Návrh, Riešenie, Testovanie.

1.2 ZADANIE PROJEKTU

Úlohou projektu je vývoj systému pre podporu študentských softvérových projektov. Tento informačný systém bude vytvárať prostredie, v ktorom si budú môcť študenti navzájom zdieľať svoje repozitáre zdrojových kódov a navzájom ich prehliadať, vyjadrovať sa k nim. Budú tak mať možnosť rozvíjať svoje schopnosti prostredníctvom efektívnej a priamej spätnej väzby ku kvalite svojich zdrojových kódov a problémom, ktoré identifikujú počas práce na projekte. Tento systém taktiež zjednoduší spoluprácu tímov a odovzdávanie a priebežnú kontrolu projektov a zadaní.

Cieľom projektu je zjednodušiť a zrýchliť vývoj softvérového produktu, taktiež skvalitniť zdrojový kód vzájomnou spätnou väzbou a dobrou komunikáciou medzi vývojármi softvérového produktu. Webový portál by mal byť využívaný ako tímový nástroj kde členovia tímu komunikujú a kontrolujú kvalitu vytvoreného kódu.

2 STANOVENIE CIEĽOV

Cieľom nášho tímového projektu je vytvoriť systém na prehliadku zdrojových kódov. Systém bude vytváraný ako webová aplikácia s využitím technológie ASP.NET MVC 4. Pomocou webového rozhrania bude možné spravovať softvérové projekty. V našom systéme máme v pláne využiť metódu značkovania zdrojových kódov z programu PerConIK. Túto metódu budeme implementovať do webového zohrania, kde bude možné značkovanie priamo vo webovom prehliadači.

Systém by mal slúžiť na zjednodušenie vyhľadávania chýb v kóde, komentovanie kódu a zvýšenie kvality kódu. Značkovanie zdrojového kódu má veľký význam pre obsiahle projekty, v ktorých sa často po dlhšej dobe stráca prehľadnosť.

Cieľom práce v zimnom semestri je vytvorenie funkčného prototypu. Prototyp by mal spĺňať základnú myšlienku zadania ale nemusí obsahovať všetky navrhnuté komponenty pre finálnu verziu projektu. Metódou vývoja SCRUM budeme vytvárať nový systém a postupne pridávať funkcionalitu. Každá iterácia bude obsahovať nové požiadavky na systém, ktoré budú jednotlivý členovia tímu postupne riešiť a tým vytvárať zjednotený systém.

V rámci zimného semestra bolo našim cieľom taktiež zlepšenie tímovej práce a zosúladenie prác jednotlivých členov tímu. Na základe vytvorenia niekoľkých metodík špecifikujeme konkrétne postupy pri práci na rôznych častiach projektu či už sa jedná o písanie kódu, verziovanie, vytváranie dokumentácie a i.

Webová aplikácia by mala obsahovať možnosť prideľovania projektov administrátorom. Následne budú používatelia systému môcť pracovať s jednotlivými pridelenými projektmi. V projekte bude možné zobraziť celkovú štruktúru súborov a entít projektu. Systém dokáže zobraziť rôzne verzie projektu a rozdiely medzi týmito verziami. Pri prehliadke kódov bude následne možné kód komentovať a pridávať značky rôzneho typu a obsahu. Ďalšia funkcionalita bude pribúdať v priebehu vývoja

3 PRED PRVÝM ŠPRINTOM

Prvou úlohou tímu bolo rozdelenie zodpovedností medzi členov. Zodpovednosti (úlohy) boli roztriedené členmi podľa predošlých skúseností a špecifikácie daných úloh.

Keďže náš tím sa zaoberá vývojom webového prostredia pre systém AST-RCS, ktorý spravuje projekty, ich verzie, používateľov, bolo nutné sa oboznámiť s týmto systémom. Systém tvorí základ pre správu dát, ktoré môžu slúžiť na zdokonalenie prehliadok kódu ale aj samotného vývoja softvérového produktu.

Pred prvým šprintom bolo potrebné naštudovať dokumentáciu systému, ako aj oboznámiť sa s cieľmi celého projektu. Tieto úkony boli potrebné na rýchle prispôsobenie členov tímu na diskusie a návrhy, ako aj na rýchlejší nábeh na vývojové tempo v projekte. Hlavným cieľom štúdia systému, bolo porozumieť logike načítavania údajov, ako aj spoznať najefektívnejšie spôsoby implementácie.

Pred prvým šprintom bolo taktiež zabezpečované samotné vývojové prostredie, či už individuálne na počítačoch členov tímu. Kde bolo nutné inštalovať niekoľko softvérov podporujúcich vývoj. Taktiež sa zabezpečoval a vyjednával server, ktorý bol nutný pre začiatok vývoja.

4 SEMIENKO

Číslo šprintu: 1

Začiatok šprintu: 9.10.2013

Koniec šprintu: 23.10.2013

Príbehy:

- **Prihlásenie používateľa AIS loginom**
- **Pridanie informácií o tíme na stránku projektu**
- **Určenie administrátora systému**
- **Priradenie používateľa k projektu**
- **Zobrazenie zoznamu projektov používateľa**
- **Načítanie súborovej štruktúry projektu do stromovej reprezentácie**
- **Načítanie AST štruktúry projektu do stromovej reprezentácie**
- **Prehliadanie stromovej reprezentácie**
- **Zobrazenie zdrojového kódu zvolenej entity z AST-RCS**
- **Zmena hesla používateľa**
- **Registrácia emailu používateľa**
- **Zvýraznenie syntaxe zobrazeného kódu**
- **Zobrazenie zoznamu odovzdaní projektu**

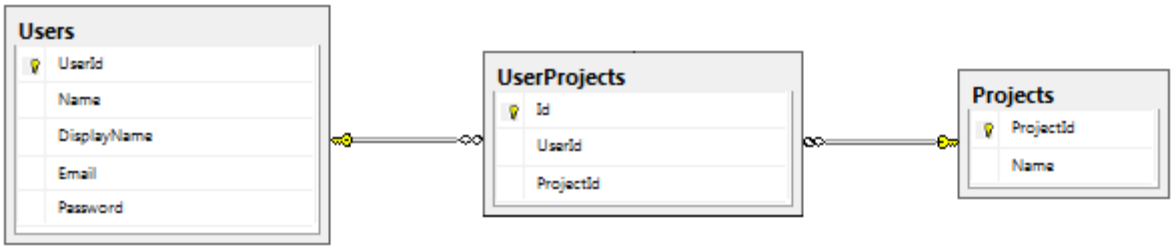
4.1 PRIHLÁSENIE POUŽÍVATEĽA AIS LOGINOM

Používateľ sa môže prihlásiť prostredníctvom *AIS* konta, aby mohli študenti STU používať vyvíjaný systém. Registrácia do systému *CodeReview* prebehne po úspešnom overení používateľa voči systému *AIS*.

Riešenie

Riešenie tohto príbehu malo viacero nadväzujúcich aktivít. Keďže sa jednalo o jednu z prvých úloh, bolo v prvom rade potrebné nainštalovať a nakonfigurovať server, nainštalovať databázový server a vytvoriť databázu. Pre jednoduchší prístup k dátam v databáze sme použili objektovo relačný mapovač *LinqToSQL*.

Dátový model obsahuje tabuľku *Users* a tabuľky pre uloženie projektov.



Obr. č. 1. Databázový model

Users – Tabuľka obsahuje základné údaje o registrovaných používateľoch

- *UserId* – identifikačné číslo používateľa
- *Name* – prihlasovacie meno používateľa (prihlasovacie meno do AIS)
- *DisplayName* – celé meno používateľa, zobrazované na stránke (meno študenta z AIS)
- *Email* – email používateľa (meniteľné, štandardne získané z AIS)
- *Password* – kryptované heslo používateľa

Projects – Tabuľka projektov, čerpaných z AST-RCS

- *ProjectId* – identifikačné číslo projektu čerpané zo systému AST-RCS
- *Name* – názov projektu čerpané zo systému AST-RCS

UserProjects – entitno relačná tabuľka, ktorá obsahuje záznamy o pridelených projektoch používateľom

Na overenie používateľa voči systému *AIS*, používame *OpenLDAP* server. Konfigurácia *AIS OpenLDAP* je nasledovná:

```
<setting name="LDAPEndpoint" serializeAs="String">
  <value>LDAP://ldap.stuba.sk:389/ou=People,dc=stuba,dc=sk</value>
</setting>
<setting name="LDAPUserDN" serializeAs="String">
  <value>uid={0},ou=People,dc=stuba,dc=sk</value>
</setting>
```

Obr. 2. Konfigurácia *OpenLDAP*

Prihlasovanie prebieha prostredníctvom prihlasovacieho formulára cez *http forms authentication*. Používateľ zadáva prihlasovacie meno a heslo. Heslo musí mať minimálnu dĺžku 6 znakov. Spojenie medzi našou aplikáciou a *OpenLDAP* serverom prebieha len v prípade ak databáza ešte neobsahuje meno používateľa, ktorý sa snaží prihlásiť. Po úspešnom overení používateľa cez *AIS OpenLDAP,* sa

vytvorí nový používateľ v internej databáze a viac sa voči *AIS* neoveruje. Heslá sú ukladané v databáze a sú kryptované 128 bitovou hešovacou funkciou *MD5*.

4.2 Pridanie informácií o tíme na stránku projektu

Na stránke systému *CodeReview* bolo potrebné zobraziť informácie o projekte a o našom „*Lucky seven*“ tíme.

Riešenie

Pridali sme novú stránku s názvom *About*. Na stránku sme umiestnili mená vývojárskeho tímu ako aj odkaz na webovú stránku tímu.

Obr. 3. Informácie o tíme

4.3 Určenie administrátora systému

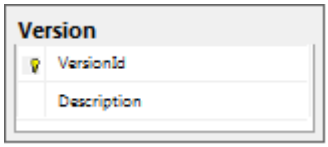
V databáze systému *CodeReview* je možné priradiť používateľovi rolu administrátora, vďaka čomu bude môcť administrátor vykonávať administračné úkony.

Analýza

Museli sme pridať ďalšiu tabuľku na definovanie používateľských rolí. Očakávame, že používatelia budú môcť mať súčasne viacero rolí. Z tohto dôvodu sme museli vytvoriť entitno relačnú tabuľku, ktorá umožní pridelenie ľubovoľného počtu rolí používateľom. Pre zmenu databázového modelu bolo potrebné spraviť zásah do databázovej štruktúry. Aby sme zakaždým nemuseli generovať novú databázu, vytvorili sme migrácie, ktoré umožňujú kontinuálne zmeny v databáze. Migrácie sa aplikujú pre každú zmenu vykonanú nad dátovým modelom.

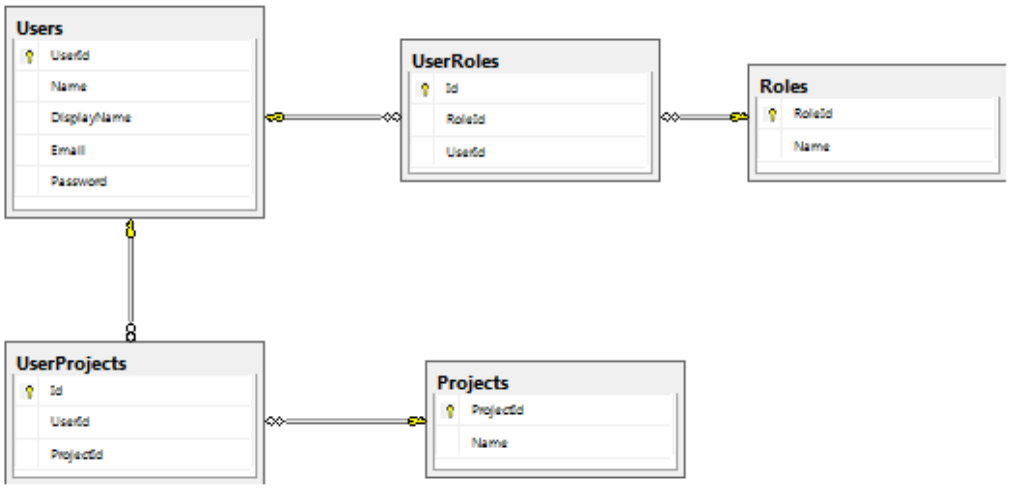
Riešenie

Keďže sme chceli meniť databázu bez silného zásahu do systému, museli sme aplikovať migrácie. Pri spustení stránky, systém načíta migračné scripty a spustí tie, ktoré ešte neboli aplikované. Aby systém vedel či má, alebo nemá aplikovať migráciu, musí si zaznamenávať verziu databázy. Pre zaznamenávanie verzie databázy sme museli vytvoriť novú tabuľku s názvom *Version*. Tabuľka obsahuje *Id* verzie a stručný opis zmien, ktoré boli vykonané v aktuálnej verzií.



Obr. č. 4. Tabuľka verzií

Dátový model v tomto štádiu obsahoval iba tabuľku používateľov s ich projektmi, preto sme pridali do súčasného modelu aj tabuľku používateľských rolí. Pre zmenu modelu sme vytvorili migračný script, ktorým sa vytvorili nové tabuľky a cudzí kľúč na vytvorenie prepojenia s tabuľkou *Users*.



Obr. č.5 Dátový model rozšírený o roly používateľov

Roles – Tabuľka číselníkov, obsahujúca roly používateľov (administrátor, používateľ)

- *RoleId* – identifikačné číslo roly
- *Name* – názov roly

UserRoles – entitno relačná tabuľka, ktorá obsahuje záznamy o roliach používateľov

4.4 PRIRADENIE POUŽÍVATEĽA K PROJEKTU

Nie všetci používatelia majú prístup ku všetkým projektom. Každý používateľ má prístup len ku projektom, ktoré mu administrátor priradí. Preto je nevyhnutné, aby mohol administrátor k projektu priradiť používateľa, čím by používateľ získal prístup k danému projektu a mohol by s ním pracovať.

Návrh

V prvom kroku je potrebné načítať zoznam všetkých projektov a zoznam všetkých používateľov. Tieto dáta je potrebné uložiť do modelu a daný model zobraziť používateľovi v prehľadnom „view“.

Po načítaní údajov sa administrátorovi zobrazí list používateľov a list projektov, v ktorom označí používateľa a projekt, ktorý mu chce priradiť. Po zvolení používateľa a projektu administrátor môže daný výber uložiť. Po uložení sa tieto údaje uložia do databázy.

Riešenie

Projekty sú načítavané z AST-RCS pomocou metódy

```
private List<Project> ReadProjectsFromASTRCS()
            {
                  var projectlist = new List<Project>();

                  using (var client = ServiceClient.AstRcsClient)
                  {
                        var projectsets = client.SearchRcsProjects(new
SearchRcsProjectsRequest());

                        foreach (var rcsproject in projectsets.RcsProjects)
                        {
                              projectlist.Add(new Project()
                              {
                                    Name =
rcsproject.Url.Substring((rcsproject.Url.LastIndexOf("/") + 1)),
                                    ProjectId = rcsproject.Id
                              });
                        }
                  }
                  //Vlozenie projektov z AST-RCS do databazy
                  ProjectMethods.InsertAllProjects(projectlist);

                  return projectlist;
            }
```

V tejto metóde prebehne načítanie projektov a porovnanie s aktuálnou databázou. V prípade, že sú v AST-RCS nové projekty, ktoré neboli zatiaľ do databázy pridané, sú vložené do databázy.

V ďalšom kroku sú z databázy načítaní všetci používatelia s rolou „user“ .

Tieto údaje sa administrátorovi zobrazia v dvoch „listbox-och“. Po uložení ním zadaných údajov je používateľovi s rolou „user“ priradený vybraný projekt a tieto údaje sú uložené do databázy.

4.5 ZOBRAZENIE ZOZNAMU PROJEKTOV POUŽÍVATEĽA

Cieľom bolo umožniť používateľovi prístup ku svojím projektom. Na to, aby mohol používateľ pristúpiť k danému projektu je nevyhnutné mu zobraziť všetky jeho projekty, z ktorých si následne má možnosť zvoliť ten projekt, na ktorom chce aktuálne pracovať.

Návrh

Projekty prihláseného používateľa je potrebné načítať z databázy . Vytvoriť model pre uloženie týchto projektov a zobraziť ich prehľadne používateľovi.

Riešenie

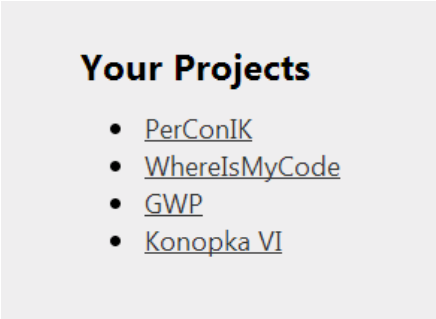
Na načítanie projektov z databázy je použitá metóda

```
public static List<Project> GetUserProjects(User user)
        {
            using (var dc = new CRDataContext())
            {
                return dc.UserProjects.Where(x => x.UserId ==
user.UserId).ToList().Select(x => x.Project).ToList<Project>();
            }
        }
```

Metóda vráti list projektov používateľa „user“. List projektov je vložený do modelu

```
public class UserProjects
  {
    public IList<Project> UserProjectsList { get; set; }
  }
```

A následne sú vo „view“ používateľovi jeho projekty zobrazené, ako na obrázku č. 6.



Obr. č. 6 Zobrazenie projektov používateľa

Testovanie

Testovanie nebolo v tejto fáze riešené pomocou automaticky naprogramovaných testov. Výsledky boli testované manuálne porovnávaním s údajmi v databáze.

4.6 NAČÍTANIE SÚBOROVEJ ŠTRUKTÚRY PROJEKTU DO STROMOVEJ REPREZENTÁCIE

Pre zobrazenie súborov projektu sme sa rozhodli zobrazovať štruktúru súborov v strome. Takáto štruktúra je prehľadná s ľahkým vyhľadávaním.

Analýza

Je potrebné analyzovať prístupy k tvorbe stromovej štruktúry v ASP.NET MVC 4. Analyzujeme spôsoby vytvárania stromu v jazyku C#. Následne analyzujeme metódy zobrazenia stromovej štruktúry v cshtml a css.

Návrh

Podľa analýzy sme navrhli jednoduchú stromovú štruktúru pozostávajúcu s uzlov, ktoré obsahujú informácie o jednotlivých uzloch a zoznam referencii na ďalšiu úroveň uzlov.

Riešenie

Pre reprezentáciu uzla bola vytvorená trieda EntityNode.

Zo systému AST-RCS boli pomocou servisu SearchFiles zaznamenané cesty ku všetkým súborom požadovaného projektu. Z ciest k súborom bola vytvorená stromová štruktúra súborov projektu.

Objekt stromu bol následné zobrazený ako zoznam na stránke. Strom sa menil podľa zadaného čísla projektu.

Testovanie

Zobrazená stromová štruktúra bola porovnávaná s reálnou štruktúrou súborov v niekoľkých projektoch. Zobrazené štruktúry sa zhodovali.

4.7 NAČÍTANIE AST ŠTRUKTÚRY PROJEKTU DO STROMOVEJ REPREZENTÁCIE

V tomto príbehu je nutné načítať kódové entity zo systému AST-RCS, tak aby ich bolo možné zobraziť používateľovi v stromovej štruktúre. Je takisto potrebné načítať podrobné údaje o týchto kódových entitách ako názov, typ a iné údaje, ktoré by mohli používateľa zaujímať.

Analýza

V prvom rade je nutné analyzovanie možností systému AST-RCS, z ktorého je nutné načítať údaje. Z tohto systému je potrebné nahrať údaje o všetkých kódových entitách daného projektu. Vstupom do tejto funkcie je projekt, ktorého entity chceme načítať. K dispozícií sú servisy, ktoré poskytujú údaje z databázy AST-RCS a mnoho vyhľadávacích metód, ktoré je možné použiť.

Metódy servisu súvisiace s kódovými entitami:

GetCodeEntity() – vráti najnovšiu entitu a informácie o nej podľa jej ID.
GetCodeEntityHistory() – vráti predchodcov entity podľa jej ID a intervalu.
SerachCodeEntity() – hľadá entity podľa rôznych kritérií a vráti pole ID vyhovujúcich entít.

Návrh

Pri riešení tohto problému je nutné efektívne využiť metódy poskytované systémom AST-RCS a údaje zoradiť do stromovej štruktúry.

Pre načítanie všetkých entít daného projektu je možné využiť funkciu *SerachCodeEntity(),* kde je možné podľa identifikátora projektu vyhľadať všetky projektové kódové entity. Metóda vráti zoznam identifikátorov entít.

Načítané údaje je neskôr možné využiť v ďalšej funkcií (*GetCodeEntity()*), ktorá načíta podrobné údaje o každej entite podľa identifikátora. Podľa týchto podrobných údajov bude možné zostrojiť stromovú štruktúru.

Riešenie

V riešení sú využité navrhnuté metódy zo systému AST-RCS. Po zavolaní týchto metód sú načítané všetky kódové entity s veľkým množstvom atribútov. Na zostrojenie stromovej štruktúry sú potrebné nasledovné atribúty:

- *ID – identifikátor entity v rámci systému AST RCS.*
- *ParentEntityId – identifikátor rodiča kódovej entity.*
- *EntityType – identifikátor typu entity.*

Po načítaní podrobných údajov o každej entite do triedy *EntityNode* sú údaje spracované do dátovej štruktúry *Dictionary<int, EntityNode>,* tak aby podľa identifikátorov bolo možné pristupovať k jednotlivým entitám.

Iteráciou cez túto štruktúru program vyplní údaje o potomkoch entít v štruktúre EntityNode. Týmto krokom sa vytvorí stromová štruktúra. Metóda, ktorá načítava tieto údaje vráti koreň („root“) tohto stromu na ďalšie spracovanie.

Atribúty EntityNode:

```
public int ID { get; set; }
public int VersionID { get; set; }
public int? ParentID { get; set; }
public string Type { get; set; }
public string Name { get; set; }
public List<EntityNode> Children;
```

Testovanie

Testovanie tohto príbehu je testované spolu s príbehom, ktorý zobrazuje tieto údaje na stránke. Je vytvorený automatický jednotkový test, ktorý sa spúšťa spolu s nasadzovaním kódu.

4.8 PREHLIADANIE STROMOVEJ REPREZENTÁCIE

V tomto príbehu bolo potrebné zabezpečiť vizuálne zobrazenie stromovej štruktúry získanej z verziovacieho systému AST-RCS a reprezentovať ju vizuálne a prehľadne vo webovom prehliadači používateľa. Konkrétne ide o stromové štruktúry, ktoré reprezentujú súborovú a entitnú štruktúru projektu samotného.

Analýza

Ako prvé bolo potrebné zvážiť, akým spôsobom by mohol byť daný strom vizualizovaný vo webovom prehliadači. Keďže ide o súborovú resp. entitnú štruktúru, ako prvé nás napadlo využitie HTML značky neusporiadaný zoznam ("<ul>"). Táto značka má možnosť v sebe obsahovať vnorené neusporiadané zoznamy a takýmto spôsobom reprezentovať stromovú štruktúru, pričom samotné úrovne sú potom jednotne odsadzované. Ďalšou možnosťou by bolo vytvorenie vlastného systému, ktorý by sa musel špeciálne naštýlovať. Vzhľadom na komplexnosť takéhoto systému, ktorý by iba imitoval niečo čo už existuje, bol použitý prvý nápad využívajúci neusporiadaný zoznam.

Návrh

Na zabezpečenie tejto funkcionality bolo potrebné navrhnúť dve samostatné časti.

Sú to:

1. Kontrolér - ovláda logiku; načítanie stromu podľa zvoleného typu
2. View - viaže sa na kontrolér; vizualizuje výsledky

Samotný kontrolér bude využívať kód, ktorý napísali dvaja iný členovia tímu. Tento kód sa stará o načítanie údajov z AST-RCS a vytvorenie dátovej štruktúry. Následne vráti odkaz na koreň stromu, ktorý reprezentuje získané dáta vo vopred definovanej podobe.

Riešenie

Ako prvý bol vytvorený kontrolér s názvom *TreeViewController*. Tento kontrolér prijíma niekoľko parametrov a na základe nich načítava korektný typ údajov o zvolenom projekte. Sú to parametre:

1. type (Integer) - typ stromu; súbory alebo entity
2. projectId (Integer) - identifikačné číslo zvoleného projektu

Typy stromov sú definované v súbore *TreeTypesEnum*. Momentálne existujú dva typy:

- *TreeTypesEnum.CodeEntities* - kódové entity
- *TreeTypesEnum.ProjectFileEntities* - súborové entity

Na základe zvoleného typu je zavolaná príslušná metóda zo statickej triedy *LoadEntities*:

- *LoadEntities.LoadCodeEntities()* - pre kódové entity
- *LoadEntities.LoadFileSystemEntities()* - pre súborové entity

Po prijatí stromovej štruktúry je zavolané View, do ktorého sa celá štruktúra pošle.

Vo View je stromová štruktúra vizualizovaná pomocou vnorených neusporiadaných zoznamov. Keďže je to pomerne komplexná dátová štruktúra, vo View je ešte definovaná pomocná funkcia. Táto funkcia rekurzívne vykreslí strom na základe danej dátovej štruktúry.

Testovanie
Za predpokladu, že kód na načítanie dátovej štruktúry poskytnutý mojimi tímovými kolegami je korektný, bolo potrebné overiť, či prebieha korektne aj samotné vykresľovanie v prehliadači. Overovanie prebiehalo hlavne manuálne, kedy sa porovnávala získaná vizuálna reprezentácia s tou, ktorá sa nachádzala vo verziovacom systéme. Bolo potrebné overiť, či jednotlivé úrovne boli korektne odsadené. Pri overovaní a testovaní finálnej verzie neboli nájdené problémy.

4.9 Zobrazenie zdrojového kódu zvolenej entity z AST-RCS

V tomto príbehu bolo potrebné zabezpečiť zobrazenie zdrojového kódu zvolenej entity z verziovacieho systému AST-RCS pre prehliadanie obsahu danej entity používateľom.

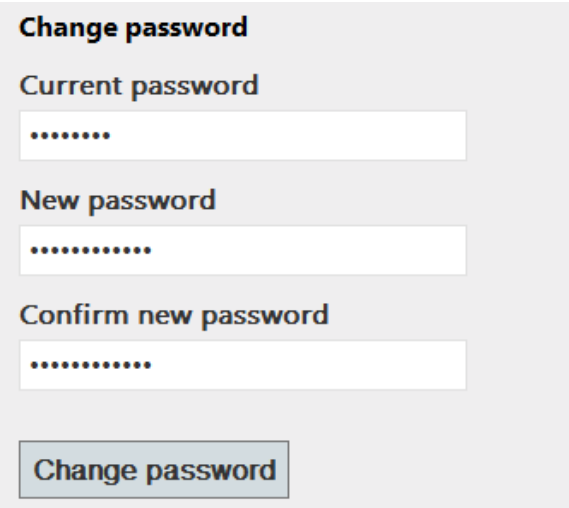
Riešenie

Na zabezpečenie požadovanej funkcionality bola implementovaný kontrolér *FileRequestController*, ktorý slúži na načítanie obsahu zvolenej entity. Tento kontrolér prijíma presný identifikátor verzie (*versionId*) zvolenej entity, na základe ktorej spraví dopyt pre získavanie obsahu entity zo systému AST-RCS. Po načítaní obsahu entity sa zavolá prislúchajúci View, kde je reprezentovaný zdrojový kód entity.

4.10 Zmena hesla používateľa

Používateľ si môže zmeniť svoje heslo do systému, aby zvýšil bezpečnosť svojho konta.

Riešenie
Pre zmenu hesla bolo potrebné vytvoriť formulár na zmenu hesla do systému. Formulár sa zobrazí po kliknutí na meno aktuálne prihláseného používateľa.



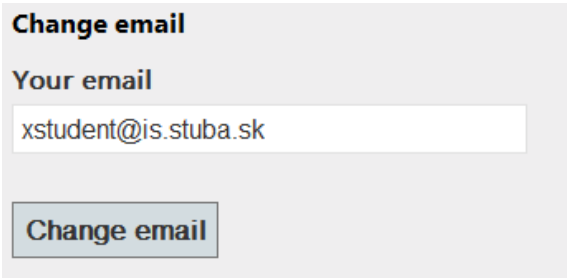
Obr. č. 7 Formulár na zmenu hesla používateľa

4.11 Registrácia emailu používateľa

Používateľ si zaregistruje email, aby mohol v budúcnosti prijímať notifikácie.

Riešenie
Registrácia emailu používateľa prebieha pri registrácií. Email používateľa získame spolu s ostatnými dátami zo servera *AIS OpenLDAP*. Tento email uložíme do databázy k používateľovi. Ďalšou úlohou

bolo vytvoriť formulár na zmenu emailu používateľa. Formulár sa zobrazí po kliknutí na meno aktuálne prihláseného používateľa.



Obr. č. 8 Formulár na zmenu emailu používateľa

4.12 Zvýraznenie syntaxe zobrazeného kódu

Úlohou pri tomto príbehu bolo zlepšiť prehľadnosť a uľahčiť čitateľnosť zobrazeného zdrojového kódu zvolenej entity. Na zabezpečenie spomenutého bolo potrebné vytvoriť riešenie na zvýraznenie zdrojového kódu entity.

Analýza

Skúmané boli možné riešenia zvýraznenia zdrojových kódov pre webové stránky. Tieto riešenia sú zvyčajne implementované v podobe JavaScript a CSS súborov, ktoré sa začlenia do HTML kódu danej stránky. Úlohou bolo nájsť najprijateľnejšie riešenie pre náš projekt.

Návrh

Po analýze problematiky bol vybratý JavaScript modul *google-code-prettify*, ktorý umožňuje definovanie vlastných tém pre zvýraznenie zdrojových kódov, číslovanie riadkov zobrazeného kódu, a podporuje zvýraznenie zdrojových kódov pre viaceré programovacie jazyky. Na obrázku č. 9 je

```
<script src="https://google-code-
prettify.googlecode.com/svn/loader/run_prettify.js">
</script>
<pre class="prettyprint linenums:1">@codeContent</pre>
```

znázornená jeho použitie v súbore .cshtml.

Obr. č. 9 použitie modulu *google-code-prettify* v súbore .cshtml

Riešenie

Implementácia príbehu bola spravená podľa návrhu. Aplikovanie zvýraznenia zdrojových kódov zvolenej entity pri prehliadnutí jej obsahu bolo zahrnuté do „View“ kontroléra *FileRequest*. Podobným spôsobom sa da aplikovať zvýraznenie aj pre ďalšie zobrazenia zdrojového kódu, ktoré môžu pribudnúť v nasledujúcich šprintoch.

Testovanie

Testovanie príbehu prebiehalo manuálne pre rôzne obsahy zvolených entít. V každom prípade sa aplikovalo zvýraznenie obsahu danej entity, test skončilo úspešne.

4.13 Zobrazenie zoznamu odovzdaní projektu

Hlavnou úlohou tohto príbehu bolo zobrazenie odovzdaní projektu, tzv. changesetov. Tie slúžia na to, aby si mohol používateľ pozrieť, kedy nastali zmeny v programe – kto a kedy uložil aktuálnu verziu programu na server.

Analýza

Jednotlivé odovzdania programov sú uložené na serveri AST-RCS. Pomocou volania rôznych služieb môžeme zo servera získať zoznam changesetov. Aby sme mohli volať služby, potrebujeme mať najprv vytvoreného klienta, pomocou ktorého budeme získavať údaje. Existuje viacero typov klientov. My použijeme klienta pre pripojenie k AstRcs službám a User službám, pričom medzi AstRcs službami sú volania na získanie changesetov a pri User službách volania pre získanie mena (loginu) používateľa pomocou jeho ID na serveri.

Návrh

V prvom kroku vytvoríme dvoch klientov na pripojenie k serveru, aby sme mohli využívať jeho služby. V druhom kroku získame zoznam konkrétnych changesetov. Tie získame pomocou volania služby v AstRcsService. Pri volaní služby máme možnosť určiť za koľko posledných dní chceme dostať zoznam, tj. maximálne aké staré majú byť changesety. Funkcia vracia iba zoznam ID (identifikačných čísiel changesetov), preto zoznam prejdeme a necháme si vytiahnuť konkrétny changeset s podrobnými informáciami. Informácie následne uložíme do modelu obsahujúceho list changesetov a ten vypíšeme zatiaľ v jednoduchej internetovej stránke.

Riešenie

Najprv sme vytvorili *ChangesetController.cs* s príslušným View.

Pri implementácií sme použili dvoch klientov: *RcsServiceClient* a *UserServiceClient.* Po napojení s RCS klientom sme pomocou služby *SearchChangesets* vyhľadali zoznam ID changesetov, ktoré sa zmenili počas nami posledných zadaných dní. Zdrojový kód volania môžeme vidieť nižšie.

```
// vyhladanie changesetov
var changesets = rcsClient.SearchChangesets(new SearchChangesetsRequest()
{
        TimestampFrom = DateTime.Now.AddDays(-21),
        TimestampFromSpecified = true
});
```

Zoznam ID hodnôt sme prešli v cykle a vyhľadali sme informácie o konkrétnom changesete. Na pomoc pre ukladanie changesetov sme vytvorili model, ktorý obsahuje list skladajúci sa z changesetov a userov, ktorý ho vykonali. Usera (používateľa) ukladáme osobitne z dôvodu, že v informáciách o changesete je iba ID používateľa. Informácie o používateľovi s konkrétnym ID sme získali pomocou služby v UserService s názvom GetUser . Jedna položka listu modelu sa tak skladá z dvoch prvkov a to *ChangesetDto* a *UserDto*, ktoré sú modelmi v AST-RCS. Výsledný model po

spracovaní všetkých nájdených changesetov sme poslali do príslušného View a zobrazili ako jednoduchú tabuľku pomocou jazyka HTML.

Testovanie

V tejto fáze sme testovanie ešte neriešili pomocou naprogramovaných testov. Výsledky, ktoré sme získali, sme porovnávali so zoznamom changesetov, ktorý môžeme nájsť na našom verziovacom a tímovom serveri TFS. Pri testovaní sme pozorovali jeden problém, a tým bolo zobrazovanie changesetov rôznych tímov a nemožnosť filtrácie pri vyhľadávaní.

5 KORIENOK

Číslo šprintu: 2

Začiatok šprintu: 23.10.2013

Koniec šprintu: 6.11.2013

Príbehy:

- **Zobrazenie zoznamu projektov používateľa z novej verzie AST-RCS**
- **Rozbaľovateľná stromová štruktúra**
- **Priradenie používateľa k projektu z novej verzie AST-RCS**
- **Zobrazenie stromovej štruktúry vedľa zobrazenia zdrojového kódu**
- **Zobrazenie grafu odovzdaní projektu**
- **Zobrazenie zoznamu zmien vykonaných vo zvolenom odovzdaní**
- **Zobrazenie zdrojového kódu zmenenej entity v odovzdaní so zvýraznenými zmenami**
- **Načítanie histórie súborovej entity z AST-RCS**
- **Načítanie histórie entity zdrojového kódu z AST-RCS**
- **Zobrazenie histórie entity vo forme tabuľky**
- **Porovnanie zdrojového kódu dvoch zvolených verzií entity z AST-RCS**
- **Načítanie súborovej štruktúry projektu do stromovej reprezentácie pre novú verziu AST-RCS**
- **Načítanie AST štruktúry projektu do stromovej reprezentácie pre novú verziu AST-RCS**
- **Zobrazenie zdrojového kódu zvolenej entity z novej verzie AST-RCS**
- **Zobrazenie zoznamu odovzdaní projektu v novej verzii AST-RCS**
- **Úspešné prihlásenie s neplatným heslom**

5.1 ZOBRAZENIE ZOZNAMU PROJEKTOV POUŽÍVATEĽA Z NOVEJ VERZIE AST-RCS

Pri prechode na novú verziu AST-RCS boli zmenené prístupy k načítaniu údajov z databázy. Podľa novej dokumentácie bolo potrebné upraviť metódy načítania údajov.

Riešenie

V novej verzií AST-RCS bolo pre načítanie potrebné zmeniť klienta. V starej verzii prebiehalo načítavanie projektov nasledovne

```
    var projectsets = c.SearchRcsProjects(new SearchRcsProjectsRequest());

    foreach (var projectid in projectsets.ProjectIds)
    {
        var projectinfo = c.GetRcsProject(new GetRcsProjectRequest()
        {
            RcsProjectId = projectid
        });
    }
    c.Close();
}
```

Pri prechode na novú verziu bolo potrebné kód upraviť na

```
using (var client = ServiceClient.AstRcsClient)
{
    var projectsets = client.SearchRcsProjects(new SearchRcsProjectsRequest());
}
```

5.2 ROZBAĽOVATEĽNÁ STROMOVÁ ŠTRUKTÚRA

V tomto príbehu bolo potrebné zabezpečiť modifikácia zobrazovanej stromovej štruktúry a to takým spôsobom, že všetky uzly stromu sa dajú kliknutím jednoducho zbaliť a rozbaliť. Pri veľkých projektoch to zabezpečí rýchlejšiu a prehľadnejšiu prácu zo stromovou vizualizáciou.

Analýza

Je potrebné zvážiť, akým spôsobom je možné docieliť interakciu v okne prehliadača pri ktorej sa priamo mení obsah zobrazovaného dokumentu. Najrozumnejšie riešenie spočíva v použití JavaScriptu, ktorý po kliknutí na uzol stromu zabezpečí, aby sa jeho potomok skryl prípadne odokryl. Je vhodné zvážiť, či nebude dobré použiť nejakú dostupnú JavaScriptovú knižnicu, ktorá by nám prácu uľahčila, napr. jQuery.

Návrh

Návrh riešenia spočíval v napísaní jednoduchého JavaScriptového kódu, ktorý po kliknutí na daný uzol vykoná akciu, pri ktorej skryje, resp. odokryje svojich potomkov. Pri riešení bola použítá aj JavaScriptová knižnica jQuery, ktorá je priamo podporovaná v nami použitom frameworku MVC.

Riešenie

Pri riešení bol najprv mierne upravený spôsob vykresľovania stromu. Táto úprava nemala vplyv na vizuálnu stránku vykresleného stromu, ale mierne zjednodušila samotné zbaľovanie a rozbaľovanie

stromovej štruktúry. Okrem toho bola všetkým uzlom (okrem listov) pridaná v HTML kóde trieda "toggle". Na základe tejto triedy vie JavaScriptový kód interagovať s uzlami stromu. Samotný JavaScriptový kód je vďaka použitie jQuery knižnice triviálny (obr. č. 10).

```
$(".toggle").click(function () {
        $(this).next().slideToggle();
});
```

Obr. č. 10 Ukážka kódu zabezpečujúceho interakciu so stromom

Testovanie

Testovanie prebiehalo jednoducho základným overením, či sa daný strom rozbaľuje a zbaľuje korektne. Testovaný strom bol porovnávaný s plne rozbalenou verziou stromu.

5.3 PRIRADENIE POUŽÍVATEĽA K PROJEKTU Z NOVEJ VERZIE AST-RCS

Pri prechode na novú verziu AST-RCS boli zmenené prístupy k načítaniu údajov z databázy. Podľa novej dokumentácie bolo potrebné upraviť metódy načítania údajov.

Riešenie

V novej verzii AST-RCS bolo potrebné upraviť načítanie projektov, ktoré je bližšie opísané v bode 5. 1.

5.4 ZOBRAZENIE STROMOVEJ ŠTRUKTÚRY VEDĽA ZOBRAZENIA ZDROJOVÉHO KÓDU

Cieľom tohto príbehu je zabezpečiť, že ak si používateľ klikne v súborovom strome na názov súboru, obsah tohto súboru sa mu zobrazí v druhej časti okna.

Analýza

Je potrebné zistiť, akým spôsobom zabezpečíme, aby sme vedľa seba mali dve separované časti, strom a obsah súboru v rámci jednej obrazovky. Ďalej je potrebné zabezpečiť, aby keď používateľ klikne na názov súboru, obsah tohto súboru sa mu zobrazí na obrazovke bez toho aby nastalo obnovenie stránky.

Návrh

Je potrebné zabezpečiť skombinovanie dvoch samostatných kódov, ktoré už boli vytvorené. Zobrazovanie stromov som riešil už v predchádzajúcom šprinte. Zobrazovanie súborov a zvýraznenie syntaxe riešil v minulom šprinte kolega s tímu. Návrh teda pozostáva s kombinácie týchto dvoch riešení. Pre získavanie obsahu zvoleného súboru bude potrebné použiť AJAXové asynchrónne volania, vďaka ktorým vieme získavať nový obsah a ním aktualizovať dokument bez jeho obnovenia v prehliadači.

Riešenie

Samotné riešenie pozostávalo z opätovného využitia čiastkových riešení, ktoré boli vytvorené v minulom šprinte. Vedľa stromovej štruktúry je vytvorená nová sekcia použitím HTML značky * .* Táto sekcia je pri prvom načítaní prázdna. Po kliknutí na ľubovoľný názov v strome sa načíta obsah zvoleného súboru a spomínaná sekcia sa vyplní. Riešenie obnovuje obsah stránky bez jej reálneho opätovného načítania. O túto funkcionalitu sa stará pomocná AJAX metóda, ktorá je priamo súčasťou MVC. Pri použití takejto metódy nie je potrebné písať vlastný JavaScriptový kód (obr. č. 11).

```
@Ajax.ActionLink(n.Name, "FileRequest",
new RouteValueDictionary {{ "versionId", n.VersionID }, { "fileName", n.Name }},
new AjaxOptions { HttpMethod = "GET", UpdateTargetId = "file-view" })
```

Obr. č. 11 Ukážka pomocnej AJAX metódy

Testovanie
Testovanie bolo v tomto prípade priamočiare. Bolo potrebné overiť, či po kliknutí na názov súboru sa upraví zobrazovaný dokument. Zobrazované súbory boli vo finálnej verzii vždy načítané a zobrazené korektne.

5.5 ZOBRAZENIE GRAFU ODOVZDANÍ PROJEKTU

Graf odovzdaní projektu má slúžiť používateľovi ako prehľad o odovzdaniach („changesetoch“) projektu. Jednotlivé odovzdania v grafe predstavujú body grafu, ktorý sa podľa vlastností projektu môže rozvetvovať. Vetvenie môže spôsobiť prípad, keď z jednej verzie projektu vzniknú dve paralelné. Na úrovni bodov grafu je umiestnená informácia o jednotlivých verziách.

Analýza
Keďže sa má graf zobrazovať na web stránke, ďalej budú analyzované možnosti zobrazenie grafu v jazyku JavaScript prípadne jeho rozšírenia JQuery.

Existuje mnoho knižníc na zobrazovanie grafových štruktúr a stromov, ktoré zobrazujú obsah pomocou HTML5 elementu canvas. Canvas slúži na zobrazovanie grafických údajov na webe. Obsahuje teda grafickú knižnicu, ktoré ponúka funkcie na vytvorenie grafického obsahu.

Analyzované boli niektoré JavaScript knižnice, ktoré využívali HTML5 canvas:

- Sigma.js
- Processing.js
- jit.js
- D3.js

Údaje pre graf je nutné načítať zo systému AST-RCS, tieto údaje by mali obsahovať informácie o jednotlivých odovzdaniach („changesetoch“) projektu. Vzorom k tejto operácií bol analyzovaný príbeh - **Zobrazenie zoznamu odovzdaní projektu.** Tento príbeh poskytuje základné funkcie k načítaniu potrebných údajov. Tieto údaje je potrebné zotriediť do stromu tak aby mohol byť zobrazený.

Návrh
Po analýze jednotlivých knižníc sa ako najefektívnejšia možnosť ukázala nepoužiť ani jednu z nich a využiť základné funkcie HTML5 canvas:

Znazornenie bodu:

```
ctx.arc(x, y, 10, 0, 2 * Math.PI);
ctx.fill();
```

Znázornenie spojenia:

```
ctx.moveTo(x, y);
ctx.lineTo(xx, yy);
```

Načítanie dát do stromu môže byť uskutočnené iteráciou poľa a vyplnenie potomkov jednotlivých entít.

Riešenie
Jednotlivé načítané „changesety“ na zobrazenie boli usporiadané do stromu pomocou triedy:

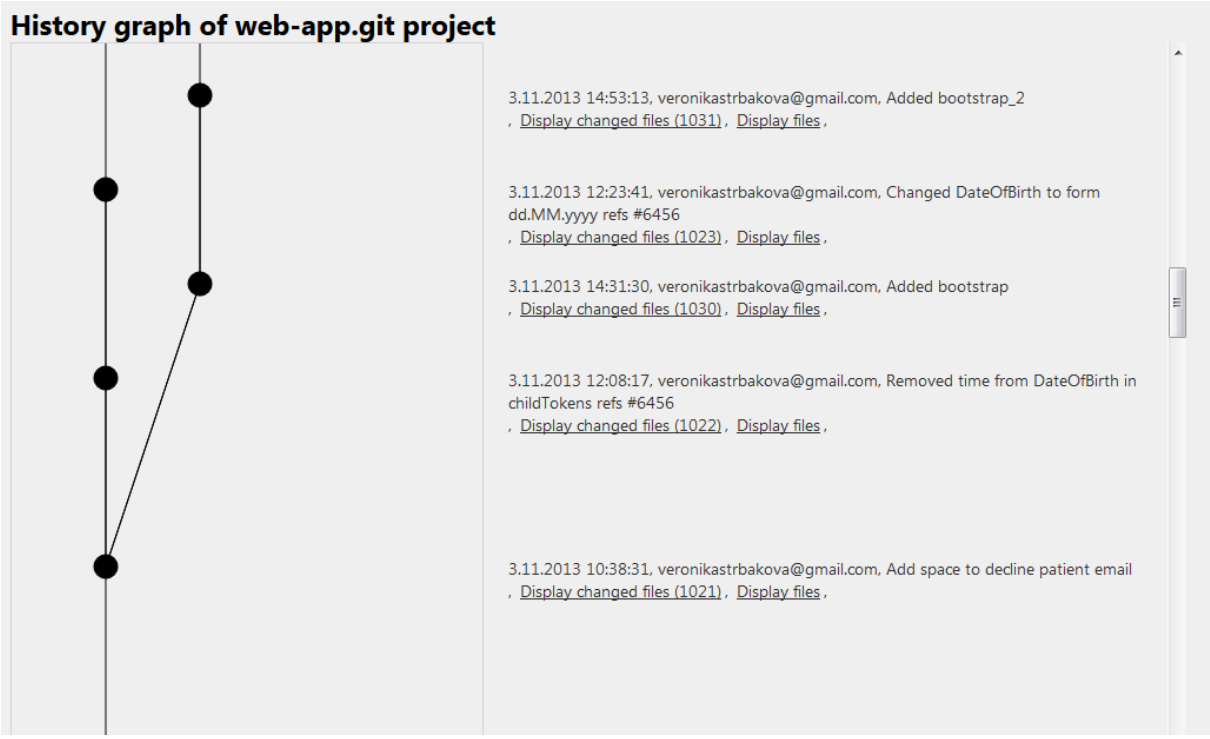
```
public class ChangesetModel
{
        public int ID { get; set; }
        public string IdInRcs { get; set; }
        public int? AncestorID1 { get; set; }
        public int? AncestorID2 { get; set; }
        public int CommiterID { get; set; }
        public string CommiterLogin { get; set; }
        public string TimeStamp { get; set; }
        public string Message { get; set; }
        public int ProjId { get; set; }
        public int?[] Children = new int?[2] { null, null };
}
```

Model bol prenesený a zobrazený na používateľovu stranu do formátu JSON vďaka funkcií:

```
var model = @Html.Raw(Json.Encode(Model));
```

Táto transformácia je zadefinovaná v príslušnom „View“ , ktorý zobrazuje stránku s grafom. Po tom ako sa načíta stránka sa spustí algoritmus vykresľovania grafu v jazyku JavaScript. Algoritmus rekurzívne prechádza vytvorený strom a podľa vetvenia rozmiestňuje body tak aby ku každému na jeho pravej strane prislúchal daný popis.

Canvas sa rozširuje spolu s rozširovaním grafu a je možné skrolovať v rámci takto vytvoreného grafu, vďaka vnoreniu elementu „canvas“ do elementu „div“.



Obr. č. 12 Ukážka grafu zmien v projekte

Testovanie

Na testovanie tohto príbehu je vytvorený automatický jednotkový test, ktorý sa spúšťa spolu s nasadzovaním kódu.

5.6 ZOBRAZENIE ZOZNAMU ZMIEN VYKONANÝCH VO ZVOLENOM ODOVZDANÍ

Pri vytváraní projektu vznikajú stále novšie verzie projektu. V týchto verziách je niekedy potrebné zobraziť zmeny, ktoré v zvolenom odovzdaní nastali. V tejto požiadavke budeme vytvárať metódu pre zobrazenie zmien vo zvolenom odovzdaní oproti predchádzajúcej verzii.

Analýza

Analyzovali sme prístupy k zisťovaniu rozdielov v zdrojových kódoch. Nástroj Diff slúži na zobrazenie rozdielov dvoch súborov. Pre jazyk C# existuje knižnica DiffModel, ktorá umožňuje porovnanie súborov s rôznym výstupným zobrazením.

Návrh

Pomocou knižnice DiffModel implementujeme metódu porovnávania súborov. Podľa výstupu označíme pridané, zmazané a zmenené riadky v zdrojovom kóde. Označené riadky zobrazíme v používateľskom rozhraní.

Riešenie

Implementovali sme metódu, ktorá pomocou knižnice DiffModel zistí rozdiely v dvoch verziách zdrojových kódov súborových entít.

```
var EntityDifferences = changesInSelectedCommit(changesetId);
```

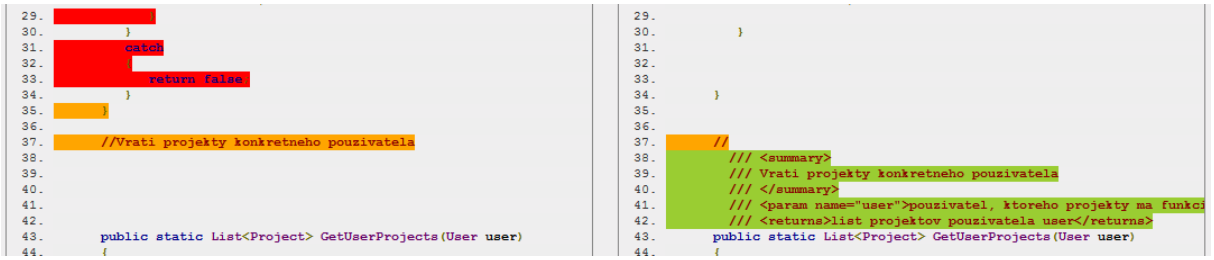
Metóda označuje riadky v kóde, na ktorých nastala zmena. V prípade zmazania súboru alebo vzniku nového súboru sa zmeny nezisťujú.

```
var d = new Differ();

var inlineBuilder = new SideBySideDiffBuilder(d);
var result = inlineBuilder.BuildDiffModel(oldFileContent.Content, newFileContent.Content);
foreach (var line in result.OldText.Lines)
{
        actualLine = "";
        switch (line.Type)
        {
                case DiffPlex.DiffBuilder.Model.ChangeType.Inserted:
                        actualLine += "+";
                        break;
                case DiffPlex.DiffBuilder.Model.ChangeType.Deleted:
                        actualLine += "-";
                        break;
                case DiffPlex.DiffBuilder.Model.ChangeType.Modified:
                        actualLine += "*";
                        break;
        }
        actualLine += line.Text + '\n';
        oldFileChanges += actualLine;
}
```

Program prečíta všetky riadky starej verzie súboru a označí zmeny. Rovnaký postup opakuje s novou verziou súboru.

Zobrazenie v používateľskom rozhraní obsahuje porovnané zdrojové kódy s farebným označením zmenených riadkov. V prípade zmazaného alebo novo vzniknutého súboru sa zobrazuje iba zdrojový kód súboru.



Obr. č. 23 Príklad zobrazenia zmien v súbore

Testovanie

Pri testovaní sme porovnávali nami zobrazené zmeny v našom projekte so zobrazenými zmenami v systéme TFS.

5.7 ZOBRAZENIE ZDROJOVÉHO KÓDU ZMENENEJ ENTITY V ODOVZDANÍ SO ZVÝRAZNENÝMI ZMENAMI

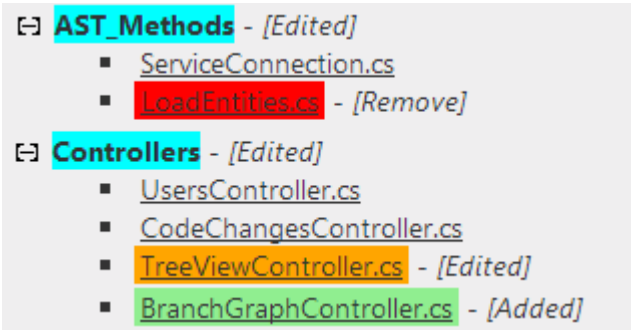
Zobrazenie zdrojového kódu zmenenej entity vo zvolenom odovzdaní. Úlohou bolo graficky zobraziť zmeny v stromovej štruktúre projektu zo zvoleného odovzdania z predchádzajúceho stavu.

Riešenie

Farebne odlíšené zmeny v graficky zobrazenej stromovej štruktúre projektu. Súbory sa zobrazujú s príznakmi a farebne rozlíšené podľa typu zmeny. Adresáre, v ktorých sú súbory uložené, farebne rozlišujeme spolu s príznakom (napr. *[Edited]*). Rozlišujeme tieto tri zmeny:

Farba	Príznak	Popis
Oranžová	[Edited]	Upravený súbor
Zelená	[Added]	Pridaný nový súbor
Červená	[Remove]	Vymazaný súbor

Pre zobrazenie vykonaných zmien (všetky 3 typy zmeny súboru) v súboroch adresára, je adresár rozlíšený bledo modrou farbou.



Obr. č. 14 Farebne rozlíšené zobrazenie súborov

5.8 NAČÍTANIE HISTÓRIE SÚBOROVEJ ENTITY Z AST-RCS

Používateľ potrebuje vedieť informácie o danej súborovej entite. Kto na nej pracoval, kedy na nej vykonával zmeny a opis toho, aké zmeny na danej súborovej entite vykonal.

Návrh

Údaje je potrebné načítať z AST-RCS na základe Id súborovej entity. Tieto údaje sa uložia do modelu.

Riešenie

Tieto údaje načítame z AST RCS pomocou

```
var projectsets = c.GetFileChangesets(new GetFileChangesetsRequest()
	{
		EntityId = entityId
	});
```

Kde c je `AstRcsClient` Tieto údaje sú následne uložené do modelu, ktorý je zobrazovaný používateľovi .

Testovanie

V tejto fáze testovanie nebolo riešené pomocou automatických naprogramovaných testov. Výsledky boli testované pre známu súborovú entitu.

5.9 NAČÍTANIE HISTÓRIE ENTITY ZDROJOVÉHO KÓDU Z AST-RCS

Našou úlohou je zistiť a vypísať, kedy nastali zmeny v kódovej entite, ktorou môže byť napríklad trieda alebo metóda, čiže nejaký menší prvok súboru. Tieto zmeny získame ako zoznam odovzdaní projektu – changesetov, kedy bola daná entita nejako upravovaná.

Analýza

Prvou otázkou, ktorá sa vyskytla pri analýze entity v AST-RCS je určenie jej ID. Entita môže mať 2 druhy ID:

- VersionID – ID konkrétnej inštancie entity (verzie)
- EntityID – ID, pomocou ktorého je identifikovaná entita bez ohľadu na jej verziu, čiže akoby nejaké globálne ID entity v projekte

Pri spracovaní určitej entity bude vhodnejšie ju najprv identifikovať podľa *VersionID*. Pomocou tohto čísla je potom možné získať *EntityID*. S ním dokážeme vytiahnuť z AST-RCS pomocou použitia správnej služby zoznam changesetov, kedy bola menená.

Návrh

Pri volaní metódy na získanie histórie entity odovzdáme parameter *VersionID*. Pomocou neho vyhľadáme konkrétnu entitu a jej changesety. Tie zaznamenáme do novovytvoreného modelu a následne pošleme do View, ktoré bude slúžiť na vhodný výpis histórie entity.

Riešenie

Pre implementáciu sme vytvorili metódu *CodeEntityHistory* nachádzajúcu sa v súbore *CodeChangesController.cs*.

Prvým krokom bolo klasicky vytvorenie klienta umožňujúceho prístup k službám v AST-RCS. Pomocou služby *GetCodeEntity* a parametra *VersionID* sme získali popis konkrétnej metódy, ktorý obsahuje aj názov, typ a *IdentityID* entity. Tieto údaje sme zaznamenali do modelu. Model obsahuje názov, typ entity a zoznam changesetov, kedy bola entita menená. Z informácií o entite sme však ešte nezískali konkrétne changesety. Na to sme použili službu GetCodeEntityChangeset. Získaným zoznamom *changesetov* sme naplnili list v modeli. Takto sme dospeli k naplneniu nášho modelu a môžeme ho ďalej poslať na zobrazenie. Na to nadväzuje User Story s názvom Zobrazenie histórie entity vo forme tabuľky.

Testovanie

Pri testovaní sme skúmali, či sa správne zobrazí výsledok a či existuje ošetrenie, ak by sa entita s daným *VersionID* nenašla. Napísali sme test, ktorý overuje, či bolo po zavolaní našej metódy následne zobrazené View.

5.10 Zobrazenie histórie entity vo forme tabuľky

Ako už názov napovedá, jedno z vhodných zobrazení histórie entity je formou tabuľky. Preto vytvoríme také View, ktoré bude spĺňať naše požiadavky.

Analýza

Ako vstup dostávame model, ktorý obsahuje meno, typ a zoznam changesetov pre danú entitu. Tento musíme spracovať a vhodne vypísať. Na tvorbu tabuľky použijeme HTML a CSS kód.

Riešenie

V súbore *EntityHistory.cshtml* sme si definovali HTML kód pre zobrazenie tabuľky. Jej vlastnosti sú doplnené v CSS súbore *site.css*. Tým sa zobrazí prehľadná tabuľka, s vyznačenými čiarami a formátovaním. Tá obsahuje výpis listu changesetov. V modeli, ktorý bol poslaný do View, sme mali ešte názov a typ entity, ktoré sme vypísali nad tabuľku pre doplnenie informácií.

Testovanie

Testovanie prebiehalo hlavne vizuálne, či daná tabuľka spĺňa v dostatočnej miere jednoduchý a prehľadný graficky dizajn. Ako sme už spomínali pri príbehu Načítanie histórie entity zdrojového kódu z AST-RCS, bol napísaný test ktorý kontroluje či sa zobrazilo toto View. Ošetrili sme prípad, že ak prišiel na vstup prázdny model, oznámili sme používateľovi, že zadal nevhodný parameter.

5.11 Porovnanie zdrojového kódu dvoch zvolených verzií entity z AST-RCS

V tomto príbehu bolo potrebné umožniť porovnanie zdrojového kódu dvoch zvolených verzií entity.

Bolo potrebné umožniť používateľovi systému prezeranie históriu verzií zvolenej kódovej entity tým, že si ľubovoľne zvolí dve verzie danej entity, ktorých obsah chce porovnať a následne sa mu zobrazia vedľa seba so zvýraznenými zmenami.

Analýza

Bola analyzovaná možnosť získania histórie verzií zvolenej entity z systému AST-RCS. Na základe týchto poznatkov, bol vytvorený návrh pre riešenie príbehu.

Návrh

Na riešenie tohto príbehu je potrebné vytvoriť kontrolér, ktorý spracuje údaje o verziach zvolenej entity. Údaje o verziách entity je vhodné ukladať, preto je vhodné vytvoriť si model, ktorý obsahuje údaje ako, *versionId* (id verzie entity)*, commiter* (osoba, ktorý pridal entity do repozitára)*, commitId* (id odovzdania do repozitára) a *date* (dátum odovzdania). Tieto údaje sa majú zobraziť aj pri prezerania histórií zvolenej entity. Na zvýraznenie zmien sa má použiť knižnica *DiffBuilder.*

Riešenie

Počas implementácie bol vytvorený kontrolér *EntityVersionController.* Kontrollér obsahuje dve základné metódy. Metódy kontroléra:

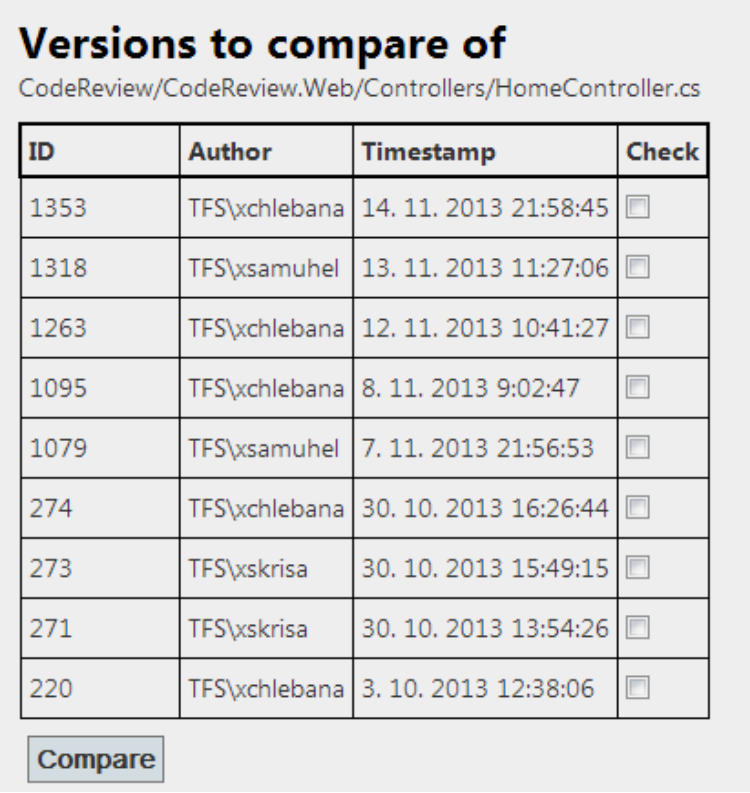
- *Index()* - na základe *versionId* získa históriu zvolenej verzie, údaje o verziách sú ukladané v modeli *EntityVersionModel* (obrázok č. 15)
- *FilesCompare()* – porovnáva dve zvolené verzie entity, zmeny v dvoch verziách sa zvýraznia

```
public class EntityVersionModel
    {
        public List<EntityVersion> entityVersionList { get; set; }
    }

    public class EntityVersion
    {
        public int commitId { get; set; }
        public int versionId { get; set; }
        public string commiter { get; set; }
        public DateTime date { get; set; }
    }
```

Obr. č. 15 *EntityVersionModel*

Naplnení model je ďalej reprezentovaná vo príslušnom View, kde sa zobrazí história verzií zvolenej entity s možnosťou výberu dvoch na porovnanie(Obrázok č. 16).



Obr. č. 16 Zobrazenie histórií verzie zvolenej entity

5.12 Načítanie súborovej štruktúry projektu do stromovej reprezentácie pre novú verziu AST-RCS

Nová verzia AST-RCS obsahuje zmenené prístupy k načítaniu údajov z databázy. Podľa novej dokumentácie sa metódy načítania údajov upravia.

Riešenie

V novej verzií AST-RCS bolo potrebné načítať najprv všetky changesety(zoznam zmien). Podľa zvoleného id changesetu bolo následné možné načítať súbory zvolenej verzie projektu. Do uzlu v strome pribudli údaje o VersionId (id verzie) súboru. Podľa tohto ID je následné možné zobraziť detaily zvolenej verzie súboru.

5.13 Načítanie AST štruktúry projektu do stromovej reprezentácie pre novú verziu AST-RCS

Nová verzia AST-RCS priniesla zmeny v metódach, ktoré zasiahli príbeh: Načítanie AST štruktúry projektu do stromovej reprezentácie. Preto bolo nutné pozmeniť štruktúru načítavania údajov z tohto systému.

Analýza

Zmeny v systéme zasiahli hlavne správu verzií kódových entít. Preto je v novej verzií nutné v prvom rade zadefinovať, z ktorého „changesetu“ alebo verzie chce systém načítať kódové entity. Nová verzia použitej metódy *SearchCodeEntity()* už neumožňuje načítať kódové entity podľa identifikátoru projektu, ale práve podľa „changesetu“.

Cieľom tejto metódy bolo načítať kódové entity poslednej verzie daného projektu. Na načítanie „changesetov“ služia v novej verzií metódy:

SearchChangesets() – vráti zoznam „changesetov“, ktoré vyhovujú kritériám

GetChangesets() – vráti údaje o danom „changesete“ podľa identifikátoru

Návrh

Keďže vstupom do metódy ostáva identifikátor daného projektu, je treba načítať identifikátor posledného „changesetu“ tak, aby boli načítané najnovšie kódové entity. Tento identifikátor je už možné použiť v rovnakých metódach, ktoré boli použité v prvej verzií.

Riešenie

Vo finálnom riešení nenastala vážnejšia zmena. Zmena nastala iba v načítavaní údajov zo systému AST-RCS.

Testovanie

Testovanie je prevedené rovnako ako v prechádzajúcej verzií.

5.14 Zobrazenie zdrojového kódu zvolenej entity z novej verzie AST-RCS

Úlohou bola zmeniť už vytvorené funkcionality nášho systému pre prácu s novou verziou systému AST-RCS.

Riešenie

Zmeny boli vykonané v kontroléri *FileRequestController*. Opravili sa volania služieb AST-RCS na volania nových služieb. Nebola pôsobená výrazná zmena funkcionality spomenutého kontroléra.

5.15 Zobrazenie zoznamu odovzdaní projektu v novej verzii AST-RCS

Po prvom šprinte sa zmenila štruktúra služieb v AST-RCS. Preto je nevyhnutná úprava doterajších riešení. Našou úlohou je upraviť získavanie zoznamu odovzdaní projektu.

Analýza

Hlavným rozdielom voči doterajšiemu riešeniu je, že všetky potrebné služby sú aktuálne pohromade a nepotrebujeme vytvárať viacero klientov. S pribúdajúcimi changesetmi a testovaním sme si tiež všimli, že je potrebné ošetriť stránkovanie, pretože naraz sa dá získať iba 100 changesetov. Novinkou je tiež vyhľadanie changesetov podľa ID projektu, ktoré nám v predchádzajúcej verzii chýbalo.

Riešenie

Pri implementácií sme upravili program podľa analyzovaných nedostatkov a zmien.

V prvom rade sme použili iba jedného klienta na pripojenie k serveru a zodpovedajúcim službám. Po pripojení sme vyhľadali changesety, pričom sme zadali parameter, ktorým je ID projektu a nie čas vytvorenia, ako to bolo predtým. Ako výsledok sme nedostali zoznam ID changesetov ale už pole changesetov zložených z prvkov *ChangesetDto*. V tomto modeli nastala tiež zmena. Nepotrebujeme už zisťovať login používateľa a hľadať ho osobitne, model changesetu obsahuje prvok *Commiter* s potrebnými informáciami. Preto sme upravili aj náš model, do ktorého ukladáme changesety. Skladá sa z podobnej štruktúry ako model ChangesetDto na serveri, avšak informácie si kopírujeme do vlastného modelu. Je to z dôvodu, že ak by nastala niekedy v budúcnosti opäť zmena na serveri, tak nemusíme byť viazaný na konkrétny serverový formát, ale máme vlastný. Informácie tak uložíme do listu pozostávajúceho z nášho modelu a ten posielame do View, kde sa zobrazí v prehľadnej tabuľke. Poslednou doplnenou funkcionalitou je stránkovanie. To je pridané z dôvodu, že môžeme získať maximálne 100 changesetov na jedno hľadanie na serveri, preto ho musíme urobiť viac krát podľa počtu stránok.

Testovanie

Testovanie prebiehalo ako v predošlej verzii, pomocou TFS servera. Rozdielom však je filtrovanie podľa ID projektu, čiže pri zadaní ID nášho projektu výpis presne zodpovedal výpisu na našom serveri. Stránkovaním sme zabezpečili, že sa zobrazili aj posledné vykonané changesety, ktoré sa pri testovaní bez neho nezobrazovali, keďže ich počet prekročil 100.

5.16 Úspešné prihlásenie s neplatným heslom

Metóda, ktorá sa stará o prihlasovanie obsahovala chybu, ktorá umožnila používateľovi prihlásiť sa s neplatným heslom.

Riešenie

Metódu, ktorá kontrolovala, či používateľ existuje v systéme mala návratovú hodnotu existujúceho používateľa aj keď nezadal správne prihlasovacie meno. Riešením bolo vrátenie hodnoty *NULL.*

6 STONKA

Číslo šprintu: 3

Začiatok šprintu: 6.11.2013

Koniec šprintu: 20.11.2013

Príbehy:

- **Vytvorenie vizuálu stránky**
- **Vytvorenie kontextového menu pre položky**
- **Kontrola prístupu k dátam**
- **Filtrácia histórie odovzdaní**
- **Stránkovanie histórie odovzdaní**
- **Priradenie projektov k používateľovi**
- **Priradenie používateľov k projektu**
- **Určenie administrátorov systému**
- **Definovanie aktívnych odkazov v kontextovom menu**
- **Úplné zobrazenie číslovania riadkov**
- **Vizualizácia zmeny v strome**
- **Vizualizácia rozbalenia uzla stromu**
- **Vytvorenie atribútov na autentifikáciu pre vytvorený model**
- **Prázdne riadky na začiatku a konci kódu**

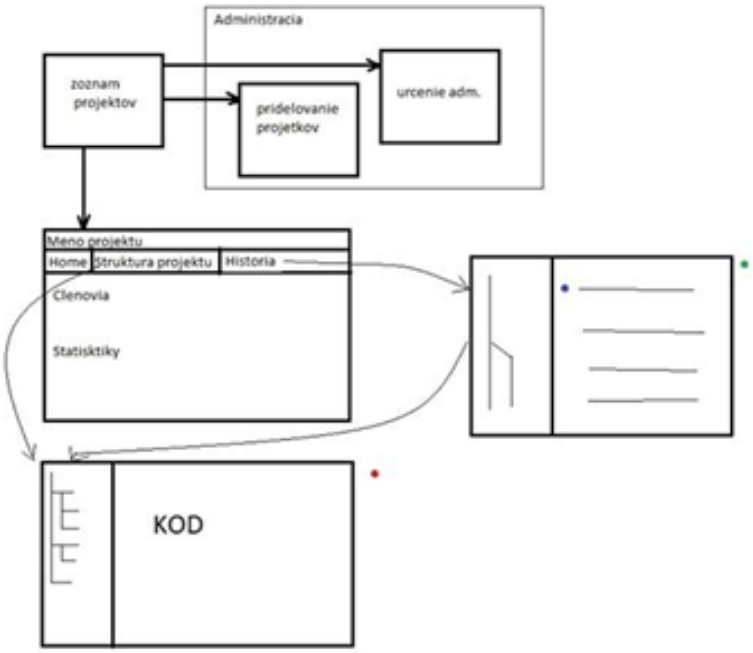
6.1 VYTVORENIE VIZUÁLU STRÁNKY

Je potrebné vytvoriť jednotné používateľské rozhranie, v ktorom sa môže používateľ jednoducho orientovať. Na navigáciu je potrebné vytvoriť menu projektu, ktoré bude obsahovať kategorizované odkazy na jednotlivé časti zobrazenia projektu.

Analýza

Analyzovali sme prístupy vytvárania jednoduchého rozbaľovacieho menu pomocou HTML a CSS.

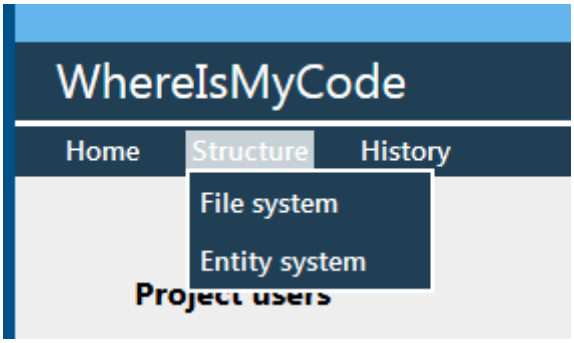
Návrh



Obr. č. 17 Návrh navigácie na stránke

Navrhli sme zobrazenie stránky projektu. Menu projektu je umiestnené v hornej časti stránky. Jednotlivé odkazy menu majú prepojenie na zobrazenie konkrétnych komponentov.

Riešenie



Obr. č. 3 Zobrazenie na stránke

V jazyku HTML a pomocou štýlov CSS sme vytvorili jednoduché používateľské menu. Jednotlivé položky menu:

- Home – informácie o zvolenom projekte, zoznam používateľom, ktorým je projekt pridelený, neskôr pribudnú štatistické údaje

- Structure – obsahuje štruktúru projektu
    - File system – strom súborov v projekte
    - Entity system – strom jednotlivých entít v projekte
- History – zobrazenie histórie zmien v projekte

6.2 VYTVORENIE KONTEXTOVÉHO MENU PRE POLOŽKY

Cieľom tohto príbehu je vytvorenie jednoduchého kontextového menu pre položky v strome, ktoré nám poskytne ďalšie možnosti, akými s nimi môžeme pracovať.

Analýza

V tomto príbehu je potrebné zistiť, akým spôsobom budeme reprezentovať samotné menu, kde v dokumente sa bude nachádzať, akým spôsobom zabezpečíme vizuálnu úpravu dokumentu a zobrazovanie menu, a zároveň ako to bude celé vyzerať.

Návrh

Návrh riešenia sa skladá z troch hlavných častí.

- HTML kód - Bolo potrebné upraviť HTML kód a ku každému uzlu pridať prvok, ktorý bude predstavovať jednoduché kontextové menu. Toto menu musí byť zároveň jednoducho editovateľné, to znamená, že jeho obsah musí byť definovaný tiež v tomto dokumente.
- JavaScriptový kód - Bolo potrebné pridať kód, ktorý po kliknutí na tlačidlo zobrazí toto menu a pri opätovnom kliknutí naňho toto menu skryje. Taktiež musia byť pokryté situácie, kedy je menu zobrazené a používateľ klikne na iné tlačidlo pri inej položke prípadne klikne niekde inam v rámci dokumentu.
- CSS kód - Bolo potrebné menu naštýlovať.

Riešenie

Samotné riešenie pozostáva s trochu spomínaných častí. V HTML sa nachádza menu, ktoré obsahuje zvolené položky. Toto menu je pri načítaní dokumentu neviditeľné. O to sa stará CSS kód, ktorý definuje, že prvok nie je zobrazený. Po kliknutí na ikonku pre menu sa zobrazí príslušné menu pre danú položku v strome. Menu je zobrazené hneď vedľa ikonky a takéto zobrazenie je zabezpečené nastavením absolútnej pozície v rámci dokumentu. Samotná pozícia je vyrátaná práve na základe pozície tlačidla (obr. č. 18).

```
menu.css({ 'top': $(this).offset().top, 'left': $(this).offset().left +
$(this).outerWidth() })
```

Obr. č. 18 Výpočet a úprava pozície

Testovanie

Testovanie prebiehalo overovaním očakávaného správania a overovaním všetkých situácii popísaných v časti "Návrh". Testovanie prebehlo úspešne.

6.3 KONTROLA PRÍSTUPU K DÁTAM

Používateľ požaduje zabezpečenie prístupu k dátam tak, aby mal každý používateľ právo vidieť dáta len z projektu, ku ktorému je priradený.

Analýza

Ak nekontrolujeme prístup k dátam na stránke, môže nastať situácia, že používateľ zmení údaje v odkaze (v linku) a zobrazia sa mu dáta, ku ktorým možno nemá prístup. V našom prípade môže

používateľ prezerať obsah ľubovoľného projektu, ak zmení *Id* projektu. Z tohto dôvodu musíme kontrolovať tento prístup voči právam používateľa.

Tento problém sa dá riešiť rôznymi spôsobmi, zvažovali sme nasledujúce možnosti:

A) Vytvorenie metódy na správu prístupu, ktorá sa volá na začiatku controllera, ktorý by teoreticky mohol zobraziť používateľovi dáta ku ktorým nemá prístup.
   Nevýhody:
   - neprehľadnosť kódu
   - duplicita kódu

B) Vytvorenie atribútov. Táto metóda je efektívna prehľadná. Použitie je jednoduché – nad definíciu controllera sa napíše atribút do hranatých zátvoriek („*[]*“). Atribúty sú jazykové konštrukcie, ktoré môžu doplniť elementy programového kódu o špecifické doplňujúce informácie.

Na prístup k súborom sme museli zistiť *id* projektu podľa *id* súboru. *AST-RCS* neposkytovalo takúto možnosť, preto sme museli požiadať *GRATEX* o implementovanie tejto funkcionality.

Riešenie

Pre tento problém sme zvolili riešenie pomocou atribútov. V súčasnom stave sme potrebovali vyriešiť kontrolu prístupu k projektu a k súboru.

Kontrola prístupu k projektu:

Vytvorili sme atribúty `[AccessDeniedProject]` a `[AccessDeniedChangeset]`. Tieto atribúty sa píšu pred definíciu controllera, ktorý obsahuje parameter *projectId* a pre súbor *changesetId*.

```
[AccessDeniedProject]
public ActionResult Index(int projectId = 0, string projectName = "")
{
```

Obr. č. 19 Použitie atribútu *AccessDeniedProject*

```
[AccessDeniedChangeset]
public ActionResult TreeViewChanges(int type = 0, int? changesetId = null)
{
```

Obr. č. 20 Použitie atribútu *AccessDeniedChangeset*

Atribút *AccessDeniedProject* nájde podľa parametra v linku *projectId* a overí či aktuálne prihlásený používateľ má prístup k tomuto projektu. Ak používateľ má prístup, controller pokračuje vo vykonávaní. Ak nemá, zobrazí sa chybová stránka o zamietnutí prístupu.

```
public class AccessDeniedProjectAttribute : AuthorizeAttribute
{
  public override void OnAuthorization(AuthorizationContext filterContext)
  {
    if (filterContext.HttpContext.Request.Params["projectId"] != null &&
        !CrAuthorize.CanViewProject(UserMethods.GetUserIdFromName(
          filterContext.HttpContext.Request.Params["AUTH_USER"]),
          int.Parse(filterContext.HttpContext.Request.Params["projectId"])))
          {
                    filterContext.Result = new ViewResult() { ViewName = "AccessError" };
          }
  }
}
```

Pre atribút *AccessDeniedChangeset* je prístup analogický ako pri *AccessDeniedProject* s rozdielom, že sa pozerá na parameter *changesetId*.

6.4 FILTRÁCIA HISTÓRIE ODOVZDANÍ

V projekte je implementovaná funkcionalita, ktorá nám zobrazí v grafe odovzdania projektu. Bola vytvorená v druhom šprinte s názvom Korienok, pod príbehom s názvom Zobrazenie grafu odovzdaní projektu. Našou úlohou je upraviť túto funkcionalitu tak, aby nám poskytla iba uzly a graf changesetov, kde sa vyskytuje konkrétna nami zvolená entita.

Analýza

V prvom rade sme analyzovali naprogramované riešenie príbehu Zobrazenie grafu odovzdaní projektu. To poskytuje výpis všetkých changesetov projektu v podobe grafu. Niektoré uzly v grafe však potrebujeme vynechať. Tu využijeme funkcionalitu načítania histórie entity, ktorá už bola vypracovaná v príbehoch Načítanie histórie entity zdrojového kódu z AST-RCS a Načítanie histórie súborovej entity z AST-RCS. Pomocou nich získame changesety, ktoré sa týkajú histórie určitej entity, čiže uzly, ktoré máme ponechať zo všetkých uzlov projektov. Našou úlohou je teda odstrániť nepotrebné uzly a správne pospájať tie, ktoré nám ostanú, čiže vytvoriť filter.

Návrh

Správnym postupom je využitie už naprogramovaných metód pre získanie potrebných changesetov, čiže uzlov grafu. Z naprogramovaných metód tak dostaneme 2 potrebné listy a tými je list changesetov projektu a changesetov entity.

V ďalšom kroku potrebujeme aplikovať filter. Tie changesety, ktoré sa nenachádzajú v liste changesetov entity môžeme vymazať. Musíme však dávať pozor na to, aby sme správne opravili a vyplnili predchodcov changesetov. Tu nastáva viacero prípadov, ktoré bude potrebné ošetriť.

Po správnom vymazaní changesetov dostaneme iba tie, ktoré potrebujeme vypísať. Pred zobrazením však ešte musíme opraviť zoznamy children v jednotlivých changesetoch. Po ich úprave vytvoríme model, ktorý následne dokáže spracovať JavaScript pre zobrazenie grafu. Výsledný model pošleme do View, ktoré zobrazuje graf odovzdaní projektov.

Riešenie

V prvom rade musíme oznámiť, že riešenie implementované v tomto šprinte nie je konečné a bude potrebné na ňom ďalej zapracovať. Pri práci riešiteľ neodhadol správne svoj časový plán a stihol vytvoriť iba základné riešenie.

Aktuálny výsledok poskytuje úpravu takého typu projektu, ktorý nie je rozvetvený. Taktiež zatiaľ spracujeme len históriu changesetov pre súborovú entitu. Metóda je dočasne vytvorená z časti iných metód, hlavne kvôli jej vývojovému charakteru. Momentálne sme sa však zamerali na reprezentovanie základnej funkcionality tohto príbehu s tým, že celkovú funkcionalitu čo najskôr dopracujeme počas ďalšieho šprintu.

Metóda pre zobrazenie histórie súborovej entity v grafe (`FileEntityBranchGraph`), ktorá je aktuálnym výsledkom, sa nachádza v súbore *BranchGraphController.cs*. Po jej dokončení a otestovaní z nej vezmeme a použijeme iba určitú časť, ktorá poslúži ako filter.

Pri vývoji metódy sme použili v prvom kroku funkciu pre získanie changesetov (*LoadEntities.LoadChangesetsToTree*). Druhým krokom bolo získanie changesetov pre nami zadanú

súborovú entitu s VersionID zadaným v parametri. Počas tohto načítania sme vytvorili list, ktorý obsahuje ID changesetov ktoré nemáme vymazať.

Po načítani oboch listov changesetov sme prechádzali changestmi projektu a zisťovali, či je jednotlivý changeset určený na vymazanie alebo ho máme ponechať. Ak sme ho mali ponechať, pokračovali sme ďalej. Ak je určený na vymazanie, hľadali sme iné changesety, ktorým je predkom. V nich sme potom upravili ID changesetu predka tak, aby sme changeset určený na mazanie vynechali. Zaznamenali sme si, ktorý changeset treba vymazať. Po prejdení všetkých changesetov a úprave predkov sme vymazali nepotrebné changesety. Ostali iba tie, ktoré prešli naším filtrom, pričom im boli upravený predkovia. Predtým, ako sme tento zoznam zobrazili v grafe sme ešte upravili zoznamy detí v jednotlivých changesetoch. Je to potrebné pre JavaScript a korektné zobrazenie v grafe. Správne vyplnený *modelBranchGraph* následne posielame do už vytvoreného View pre históriu projektu a zobrazíme prefiltrovaný graf.

Testovanie

Testovanie zatiaľ prebiehalo iba vizuálnou kontrolou. Zistili sme však, že nefunguje odkaz na porovnanie posledných verzií changesetov, čo je zrejme spôsobené úpravou predkov, ktorú sme vykonali. Taktiež sme si všimli, že text changesetov sa prelína. Tento problém je už zaznamenaný v product backlogu a bude sa riešiť.

6.5 Stránkovanie histórie odovzdaní

Kvôli možnosti ľubovoľnej veľkosti grafu je možné, že canvas dosiahne maximálnu veľkosť. V tomto prípade používateľ neuvidí kompletný graf. Preto je nutné nájsť nový spôsob zobrazenia dát alebo stránkovania, ktorý zabráni dosiahnutiu maximálnej veľkosti canvasu.

Analýza

Vytvorenie stránkovania je založené na statickej veľkosti elementu canvas a zobrazovania len rozsah viditeľný používateľom.

Ovládanie stránkovania je možné vytvorením nového „scroll baru“, ktorý bude ovládať vykresľovanie v statickom canvase. Avšak je tiež možné využiť existujúci „scroll bar“, ktorý v súčasnej verzií ovláda skrolovanie elementu „div“, v ktorom sú vnorené elementy canvas a element, ktorý obsahuje popisi k jednotlivým bodom grafu.

Návrh

Menej zložitou metódou je zanechať „scroll bar“, ktorý bude ovládať iba časť s popismi. Samotný canvas bude mať fixnú pozíciu, avšak jeho obsah sa na základe posunu „scroll baru“ bude meniť. To bude vytvárať ilúziu skrolovania, v skutočnosti sa však bude iba meniť obsah canvasu. Pri takomto prístupe je možné zobraziť graf s ľubovoľnou veľkosťou.

Zobrazovanie iba viditeľnej časti údajov je možne dosiahnuť metódou, ktorá pri načítaní stránky vypočíta všetky pozície bodov v grafe. Pri skrolovaní sa zistí poloha a podľa nej je možné vybrať ten správny obsah.

Riešenie

Úprava oproti predošlej verzií spočíva v úprave výpočtovej funkcií, ktorá predpočíta obsah a do štruktúry uloží koordináty každého bodu.

Pri každom pohybe „scroll baru“ za zavolá funkcia:

```
$("#BranchGraphContainer").scroll(function () {
```

```
        ctx.clearRect(0, 0, 400, 600);
        drawnode($("#BranchGraphContainer").scrollTop());
    });
```

Táto funkcia vymaže predošlí zobrazený obsah a pomocou funkcie *drawnode(x)* sa vykreslí práve obsah od koordinátu x, ktorý reprezentuje polohu „scroll baru“.

Testovanie

Testovanie je prevedené rovnako ako v prechádzajúcej verzií príbehu: ZOBRAZENIE GRAFU ODOVZDANÍ PROJEKTU.

6.6 PRIRADENIE PROJEKTOV K POUŽÍVATEĽOVI

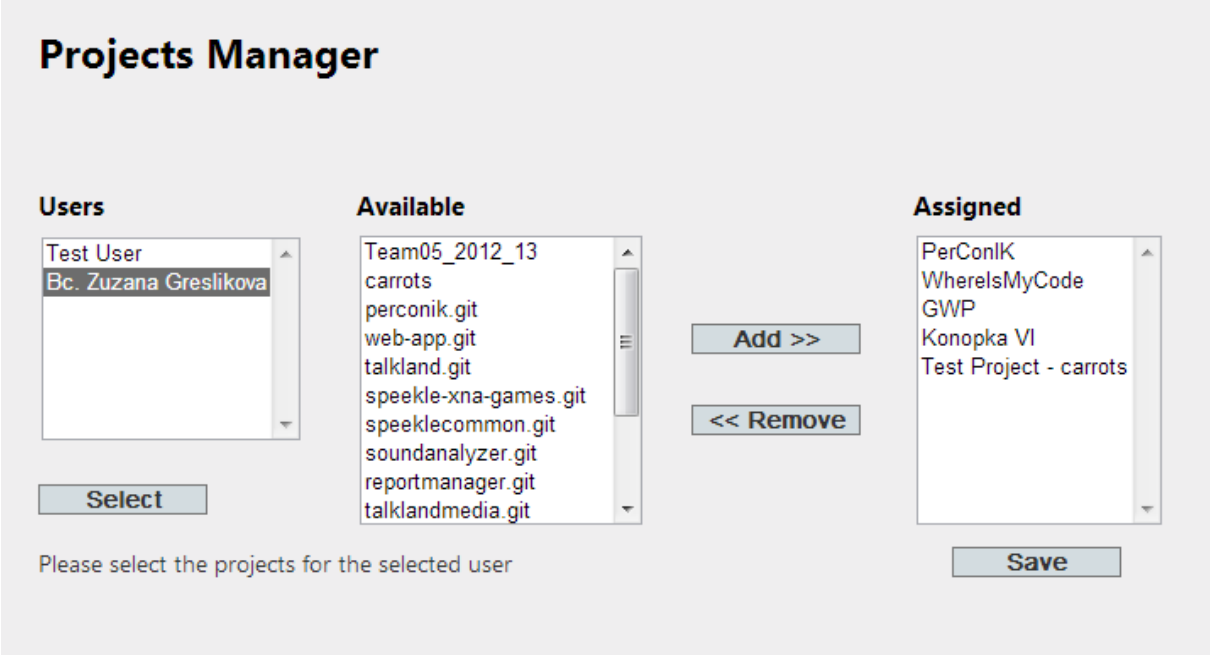
Tento príbeh nadväzuje na príbeh z prvého šprintu 4.4. Niekedy je nevyhnutné pridať jednému používateľovi väčšie množstvo projektov. Je dôležité, aby rozhranie bolo čo najjednoduchšie a aby administrátor nemusel každý projekt priradiť používateľovi individuálne, a aby bolo možné zvoliť skupinu projektov, ktoré budú používateľovi priradené.

Návrh

Je potrebné vytvoriť rozhranie ktoré umožní multiselekciu projektov.

Riešenie

Vytvorí sa model, do ktorého sa vloží zoznam všetkých používateľov. Ako prvý sa zobrazí prvý používateľ zo zoznamu, a preto model obsahuje ďalej list projektov priradených tomuto používateľovi a taktiež list projektov, ktoré mu nie sú priradené. Model sa zobrazí ako je možné vidieť na obrázku č. 21, pričom „Users“ predstavuje zoznam používateľov (so zvoleným prvým používateľom). „Available“ predstavuje projekty, ku ktorým daný používateľ nemá prístup a „Assigned“ predstavuje projekty daného používateľa.



Obr. č. 21 Rozhranie na pridávanie projektov používateľovi

Pre zvolenie jedného s používateľov musí administrátor svoju voľbu označiť a potvrdiť „Select“. Po zvolení používateľa sa zobrazí list projektov (priradených aj nepriradených) tomuto používateľovi. Pri získavaní projektov zvoleného používateľa sa používa metóda

```
public SwapperProjectModel getAllProjects(int userId)
        {
            var serializer = new JavaScriptSerializer();
            SwapperProjectModel model = new SwapperProjectModel();

            List<Project> userProjects =
DB_Methods.ProjectMethods.GetUserProjects(userId);
            List<Project> allProjects = DB_Methods.ProjectMethods.GetAllProjects();
            List<Project> noUserProjects = allProjects.Except(userProjects, new
ProjectComparer()).ToList<Project>();

            model.AssignedList = new List<SelectListItem>();
            foreach (var p in userProjects)
            {
                model.AssignedList.Add(new SelectListItem() { Text = p.Name, Selected
= true, Value = p.ProjectId.ToString() });
            }

            model.AvailableList = new List<SelectListItem>();
            foreach (var p in noUserProjects)
            {
                model.AvailableList.Add(new SelectListItem() { Text = p.Name, Selected
= true, Value = p.ProjectId.ToString() });
            }

            model.currentAvailableList = serializer.Serialize(model.AvailableList);
            model.currentAssignedList = serializer.Serialize(model.AssignedList);

            return model;
        }
```

Vstupom tejto metódy je id používateľa, ktorého projekty chceme získať. Výstupom je objekt, ktorý obsahuje informácie potrebné pre zobrazenie informácií administrátorovi.

Po zvolení „Add“ sa označený projekt z časti „Available“ priradí do listu priradených projektov a zároveň sa odstráni z listu nepriradených projektov. Po zvolení „Remove“ sa označený projekt v časti „Assigned“ zmaže z listu priradených projektov a priradí sa do listu nepriradených projektov.

Po zvolení „Save“ sú do databázy vkladané objekty z listu priradených projektov a naopak odstraňované projekty z listu nepriradených projektov, a to v metóde

```
public static void AddRemoveProjectsToUser(int u, List<int> addProjects, List<int>
removeProjects)
        {
            using (var dc = new CRDataContext())
            {
                foreach (var p in addProjects)
                {
                    if ((dc.UserProjects.Where(x => ((x.ProjectId.Equals(p)) &&
(x.UserId.Equals(u)))).Count()) == 0)
                    {
                        var project = new UserProject() { ProjectId = p, UserId = u };
                        dc.UserProjects.InsertOnSubmit(project);
                    }
                }
                foreach (var p in removeProjects)
                {
                    if ((dc.UserProjects.Where(x => ((x.ProjectId.Equals(p)) &&
(x.UserId.Equals(u)))).Count()) > 0)
                    {
                        var project = (from up in dc.UserProjects where up.ProjectId
== p && up.UserId == u select up).First();
                        dc.UserProjects.DeleteOnSubmit(project);
                    }
                }
                dc.SubmitChanges();
                return;
            }
        }
```

Testovanie

Výsledky boli testované manuálne porovnávaním s údajmi v databáze.

6.7 PRIRADENIE POUŽÍVATEĽOV K PROJEKTU

Tento príbeh nadväzuje na príbeh z prvého šprintu 4.4. Je dôležité, aby rozhranie bolo čo najjednoduchšie, a aby administrátor nemusel každého používateľa priradiť ku projektu individuálne. Je potrebné, aby bolo možné zvoliť skupinu používateľov, ktorí budú k projektu priradený.

Návrh

Je potrebné vytvoriť rozhranie, ktoré umožní multiselekciu používateľov.

Riešenie

Pri riešení bol zvolený rovnaký postup ako pri priradení projektov používateľovi (časť 6.6), s tým rozdielom, že projekty boli nahradené používateľmi a naopak.

Testovanie

Výsledky boli testované manuálne porovnávaním s údajmi v databáze.

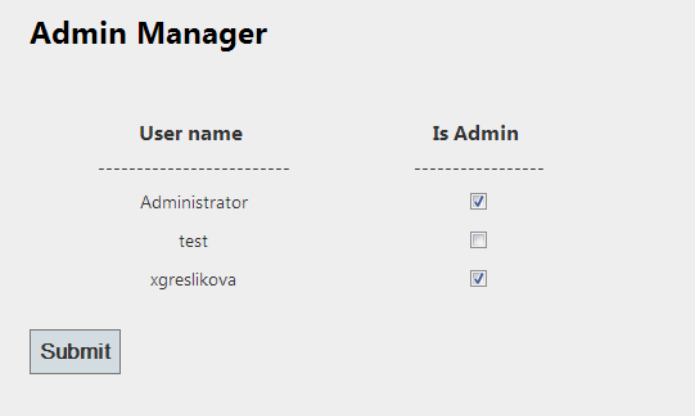
6.8 URČENIE ADMINISTRÁTOROV SYSTÉMU

Je nevyhnutné aby mal systém viac ako jedného administrátora. Tiež je potrebné aby časom bolo možné pridať nového administrátora systému no tiež odstrániť administrátorov, ktorí budú nečinní. Z tohto dôvodu je potrebné aby administrátori mali právo pridávať aj odstraňovať administátorov systému.

Analýza
Každý s používateľov má v systéme svoju rolu. Používateľ môže mať viacero rolí. Na pridávanie administrátorov má právo iba administrátor. Údaje o rolách jednotlivých používateľoch získame z databázy.

Návrh
V prvom kroku je potrebné získať z databázy všetkých používateľov a ich roly. Následne sa zistí, ktorí používatelia sú už administrátori systému. Tieto údaje sa zobrazia v prehľadnej tabuľke, kde bude prihlásenému administrátorovi umožnené zmeniť tieto údaje.

Riešenie
Na načítanie údajov o používateľoch a ich rolách v systéme bola použitá metóda `UserMethods.GetUsersAndRoles()`. Na základe týchto údajov bol vytvorený model pre každého používateľa , ktorý obsahoval základne údaje o používateľovi (Id a meno používateľa) a údaj či daný používateľ je, alebo nie je administrátorom systému. Tieto údaje boli zobrazené v tabuľke, ktorá obsahuje meno používateľa a checkbox. Ak checkbox je označený, znamená to, že daný používateľ je administrátorom systému. Náhľad sa nachádza na obrázku č. 22.



Obr. č. 22 Pridávanie administrátorov

6.9 DEFINOVANIE AKTÍVNYCH ODKAZOV V KONTEXTOVOM MENU

Úlohou bola pridávanie aktívnych odkazov do kontextové menu, ktorou sa mali zabezpečiť správne prepojenia medzi funkcionalitami systému.

Návrh

Aktívne odkazy pre entity:

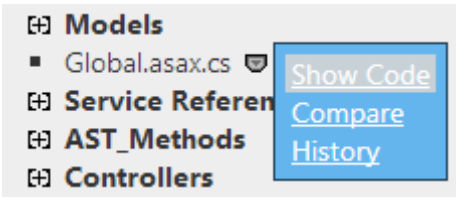
- zobrazenie kódu
- zobrazenie histórie
- porovnanie s predchádzajúcou verziou

Aktívne odkazy pre odovzdania:

- Porovnanie s prechádzajúcou verziou
- Zobrazenie detailov o odovzdaní

Riešenie

Do kontextové menu pri entitách boli pridané aktívne odkazy, ktoré boli navrhnuté. Aktívne odkazy boli pridané do View kontroléra *FileViewer*. Na obrázku č. 21 je znázornená menu s pridanými odkazmi.



Obr. č. 23 Kontextové menu pre entity s odkazmi

Aktívne odkazy pri entitách

- *Show Code* – po kliknutí sa zavolá príslušná metóda *FileRequest()* pre zobrazenie zdrojového kódu entity vedľa stromovej štruktúry projektu.
- *Compare* – zavolá sa metóda *FilesCompared()* kontroléra *EntityVersionController*, ktorá porovnáva zdrojový kód s predchádzajúcou verziou.
- *History* – zavolá sa metóda *EntityVersionController.Index()* pre zobrazenie histórie verzií danej entity

Pre zabezpečenie správnej funkcionality boli zmenené metódy kontroléra *EntityVersionController.*

Aktívne odkazy pri odovzdaniach

Pri zobrazeniach odovzdania existovali aktívne odkazy pre definované akcie. Tieto odkazy neboli v celku zmenené. Opravený bol iba nesprávne volanie zobrazenia zdrojového kódu v stromovej štruktúre zvýraznenými zmenami.

6.10 Úplné zobrazenie číslovania riadkov

Je potrebné upraviť číslovanie riadkov, ktoré je pri zobrazovaní obsahu súboru vždy po piatich.

Analýza
Problém treba riešiť drobnou úpravou kódu a zmenením triedy v HTML dokumente, ktorá je využívaná skriptom zabezpečujúcim formátovanie.

Návrh
Najvhodnejším riešením je úprava triedy "linenums".

Riešenie
Trieda "linenums:5" bola upravená na "linenums:1". Táto drobná úprava zabezpečí, že číslo riadku bude uvedené na každom riadku a nie na každom piatom riadku.

Testovanie
Testovanie prebehlo jednoduchým overením výpisu obsahu súborov. Overenie riešenia bolo úspešné.

6.11 Vizualizácia rozbalenia uzla stromu

Cieľom tohto príbehu je zabezpečiť jednoduchý vizualizačný prvok v rámci zobrazovaného stromu. Ak sa dá uzol stromu rozbaliť, pri jeho názve bude zobrazený príslušný piktogram a to isté platí aj ak sa uzol rozbaliť nedá.

Analýza
Je potrebné opäť zvážiť, akým spôsobom je možné docieliť interakciu v okne prehliadača pri ktorej sa priamo mení obsah zobrazovaného dokumentu. Najrozumnejšie riešenie spočíva v použití JavaScriptu, ktorý po kliknutí na uzol stromu zabezpečí zobrazenie vizuálneho príznaku pri uzle stromu. Keďže sme v minulom riešení použili knižnicu jQuery, aj v tomto prípade ju použijeme. Treba aj zvážiť, akým spôsobom docielime zobrazenie samotného piktogramu.

Návrh
Návrh riešenia spočíva v zachytávaní kliknutia v JavaScriptovom kóde. Následne zistíme, či bol uzol rozbalený alebo nie. Na základe tohto údaju priradíme príslušný piktogram dynamickou manipuláciou CSS dokumentu priamo v prehliadači.

Riešenie
Riešenie spočíva vo využití možnosti štýlovania neusporiadaného zoznamu. Základný zoznam využíva prednastavený piktogram plnej guličky pre jednotlivé položky zoznamu. Tento piktogram sa dá úplne odstrániť prípadne nahradiť obrázkom. Samotné riešenie teda spočíva v tom, že po kliknutí zmeníme štýl odrážky v zozname. To docielime definovaním si dvoch tried v našom CSS dokumente (obr. č. 22).

```
.plus
{
    list-style-image: url("../Images/li-plus.png");
}

.minus
{
    list-style-image: url("../Images/li-minus.png");
}
```

Obr. č. 24 Triedy v CSS dokumente

Tieto dve triedy definujú dva rôzne piktogramy. Následne už len stačí po kliknutí tieto dve triedy striedať (obr. č. 23).

```
$(".toggle").click(function () {
      $(this).toggleClass("plus");
      $(this).toggleClass("minus");
      $(this).next().slideToggle();
});
```

Obr. č. 25 Striedanie CSS tried

Testovanie

Testovanie bolo jednoduché. Stačilo overiť, či uzavretý uzol má vždy jeden typ piktogramu a otvorený uzol má vždy druhý typ piktogramu. Toto testovanie sa priamo spájalo s testovaním rozbaľovania stromu. Testovanie bolo úspešné.

6.12 PRÁZDNE RIADKY NA ZAČIATKU A KONCI KÓDU

Úlohou bola odstránenie nadbytočných znakov nového riadku, ktoré boli na začiatku a na konci zobrazených kódov.

Riešenie:

Tento problém vznikol kvôli bielych znakov medzi značkami *<pre></pre>* a reprezentáciou zdrojového kódu nejakej entity. Problém bol odstránení tým, že sa zmazali všetky biele znaky medzi značkami a spomenutou reprezentáciou.

7 CELKOVÝ POHĽAD NA PROJEKT

Po troch šprintoch môžeme napísať priebežný pohľad na aktuálny stav projektu. Počas posledných týždňov pracovali všetci členovia tímu postupne na projekte a všetkých jeho častiach. Celková práca sa dá rozdeliť do dvoch základných kategórii.

7.1 TVORBA SOFTVÉROVÉHO PROJEKTU

Počas prvých troch šprintov bola vytvorená základná kostra projektu. Táto kostra je postavená na architektonickom vzore MVC a logickú štruktúru projektu rozdeľuje na 3 základné časti: Model, View a Controller. Boli zabezpečené základné funkcionality v projekte. Sú nimi registrácia a prihlásenie používateľa, vytvorenie administrátora, ktorý môže prideľovať projekty, zobrazovanie jednotlivých vlastností projektov. Boli taktiež zabezpečené funkcionality, ktoré nie sú priamo pre bežného používateľa viditeľné, ako napr. získavanie informácii o projektoch z verziovacieho systému AST-RCS, vytvorenie potrebných dátových štruktúr na reprezentáciu údajov alebo zabezpečenie zobrazovania informácii iba pre používateľa, ktorý má na to oprávnenie.

Okrem toho boli zabezpečené všetky ďalšie súčasti, ktoré boli nevyhnutné pre samotnú prácu na projekte. Bol nakonfigurovaný server, databázový systém ako aj systém na verziovanie.

7.2 TVORBA DOKUMENTÁCIE

Nevyhnutnou súčasťou našej práce bola aj priebežná tvorba dokumentácie, ktorá opisovala našu prácu na projekte v jednotlivých šprintoch. Okrem toho bol vytvorený ešte druhý dokument, v ktorom je rozoberané riadenie v rámci projektu. Tento projekt obsahuje zoznam členov tímu, opis ich manažérskych rolí, vypracované metodiky a ako prílohu obsahuje zápisnice z jednotlivých týždenných stretnutí. Spomínané metodiky boli vypracované do detailov a obsahujú presné opisy postupov pri rôznych činnostiach akými sú napr. tvorba dokumentácie alebo písanie zdrojového kódu.