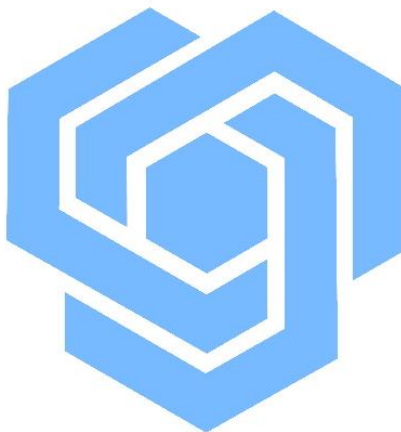


Технически Университет София

Факултет по Компютърни Системи и Управление



Дипломна Работа

На тема: „Проектиране и реализиране на приложение за изпълнение на тестове на отдалечени машини”

**Дипломант: Антон Станиславов Ангелов,
ф.н.:121208260**

Научен ръководител: доц. д-р Румен Трифонов

Съдържание

| | |
|--|----|
| 1. Увод..... | 5 |
| 2. Обзор на съществуващи подобни приложения..... | 6 |
| The Gallio Automation Platform (http://gallio.org/)..... | 6 |
| NUnit (http://nunit.org) | 7 |
| P NUnit Parallel NUnit (http://www.plasticscm.com/infocenter/technical-articles/pnunit.aspx) | 8 |
| Visual Studio & Team Foundation Server Build | 9 |
| Bromine 3 RC 1 (http://sourceforge.net/projects/bromine/) | 10 |
| Изводи от обзора на подобни приложения | 11 |
| 3. Описание на използваните технологии | 12 |
| MS SQL SERVER | 12 |
| ADO.NET Entity Framework | 12 |
| Windows Presentation Foundation (WPF) | 13 |
| MSBuild | 14 |
| 4. Функционално описание | 15 |
| Многослойна архитектура..... | 15 |
| View Model Viewmodel (VMVM) архитектура..... | 16 |
| Deployment диаграма | 18 |
| Use Case диаграма | 18 |
| Представяне на по-важните класове и техните по-важни методи | 20 |
| ServerAgent..... | 20 |
| TcpClientWrapper | 21 |
| CommandLineExecutor | 22 |
| MemberManager | 23 |
| Member | 24 |
| RegistryManager | 24 |
| ProjectSettingsLoadingView..... | 25 |
| 5. Проектиране на базата данни | 26 |
| Таблица Member..... | 28 |
| Таблица MemberRole | 29 |
| Таблица Status..... | 29 |
| Таблица Team | 30 |
| Таблица Project | 30 |

| | |
|---|----|
| Таблица AdditionalPath | 31 |
| Таблица AgentMachine..... | 31 |
| Таблица Test..... | 31 |
| Таблица ExecutionResultRun | 32 |
| Таблица ExecutionStatus..... | 33 |
| Таблица TestResultRun..... | 33 |
| Таблица Suite | 34 |
| Таблица SuiteResultRun | 35 |
| 6. Ръководство на потребителя..... | 35 |
| Клиентско приложение..... | 35 |
| Вход в системата..... | 35 |
| Валидации във формата за вход | 36 |
| Регистрационна форма | 37 |
| Форма за промяна на външния вид | 38 |
| Форма за активиране на потребители | 38 |
| Форма за активиране на акаунт | 39 |
| Форма за менажиране на настройките..... | 39 |
| Форма за добавяне на нов екип | 40 |
| Форма за добавяне на нов допълнителен път | 41 |
| Форма за добавяне на нов проект..... | 41 |
| Форма за добавяне на нова агентска машина | 41 |
| Форма за избиране на Mode..... | 42 |
| Форма за избиране на проекти | 42 |
| Форма за показване на текущия статус на зареждане на тестовете..... | 43 |
| Форма за показване на текущия статус на зареждане на командите на агентска машина | 44 |
| Форма показваща резултатите от изпълнените тестове..... | 46 |
| Форма показваща всички пускания на тестове за определени екипи..... | 46 |
| 7. Приложения..... | 47 |
| Приложение 2- клас Tcp Client Wrapper | 52 |
| Приложение 3- клас CommandLineExecutor | 54 |
| Приложение 4- клас MemberManager..... | 56 |
| Приложение 5- клас RegistryManager..... | 59 |
| Приложение 6- клас ProkectSettingsLoadingView | 62 |

| | |
|--|----|
| 8. Заключение | 67 |
| 9. Използвани източници | 68 |
| 1. http://gallio.org/Docs.aspx | 68 |
| 2. http://www.nunit.org/index.php?p=documentation | 68 |
| 3. http://www.plasticscm.com/infocenter/technical-articles/pnunit.aspx | 68 |
| 4. http://msdn.microsoft.com/en-us/library/ms181710(v=vs.90).aspx | 68 |
| 5. http://www.methodsandtools.com/tools/tools.php?bromine | 68 |
| 6. http://www.pssuk.com/Articles/SQLServer.htm | 68 |
| 7. http://blogs.msdn.com/b/dsimmons/archive/2008/05/17/why-use-the-entity-framework.aspx | 68 |
| 8. http://en.wikipedia.org/wiki/Windows_Presentation_Foundation | 68 |
| 9. http://social.msdn.microsoft.com/Forums/en-US/wpf/thread/42636e55-a1e0-4b29-bbd1-cd8073585584 | 68 |
| 10. http://en.wikipedia.org/wiki/MSBuild | 68 |
| 11. http://dotnetslackers.com/articles/net/IntroductionTo3TierArchitecture.aspx | 68 |
| 12. http://msdn.microsoft.com/en-us/magazine/dd419663.aspx | 68 |

1. Увод

Все повече и повече в днешни дни софтуерните продукти стават все по-важна част от нашия живот. Възможностите на предлаганите продукти се увеличават с всеки изминал ден, пропорционално нарастват и очакванията на потребителите. Когато някой отвори онлайн, десктоп или мобилно приложение иска то да е бързо, да не блокира и да успее да си свърши нужната работа както трябва.

Софтуерната индустрия е един от най-печелившите сектори в момента и един от най-бързо развиващите. Но за да бъде една компания успешна, трябва много бързо да реализира своята идея, за да може да спечели пазарна ниша. Но обикновено бързата реализация води до занижено качество, ако не се избере правилният подход за разработка.

В последните 30 години процесите използвани при изграждането на софтуерни системи за масова употреба са се променили корено. Един от наложилите се начини е така нареченият “Waterfall” модел, при който първо се пише целият код и едва след това се тества. Ако се окаже, че има проблеми в програмата, се налага преработка, която може да забави пускането на продукта. В текущата икономическа обстановка всяко забавяне или критичен дефект в разработваната система би коствала много на компанията. Затова много успешни компании като Google, Facebook, Microsoft избират така наречените “Agile” методологии за разработка, където продуктът се строи бързо, но тестването се извършва паралелно с всяка фаза на разработка. По този начин критичните проблеми се намират и оправят навреме.

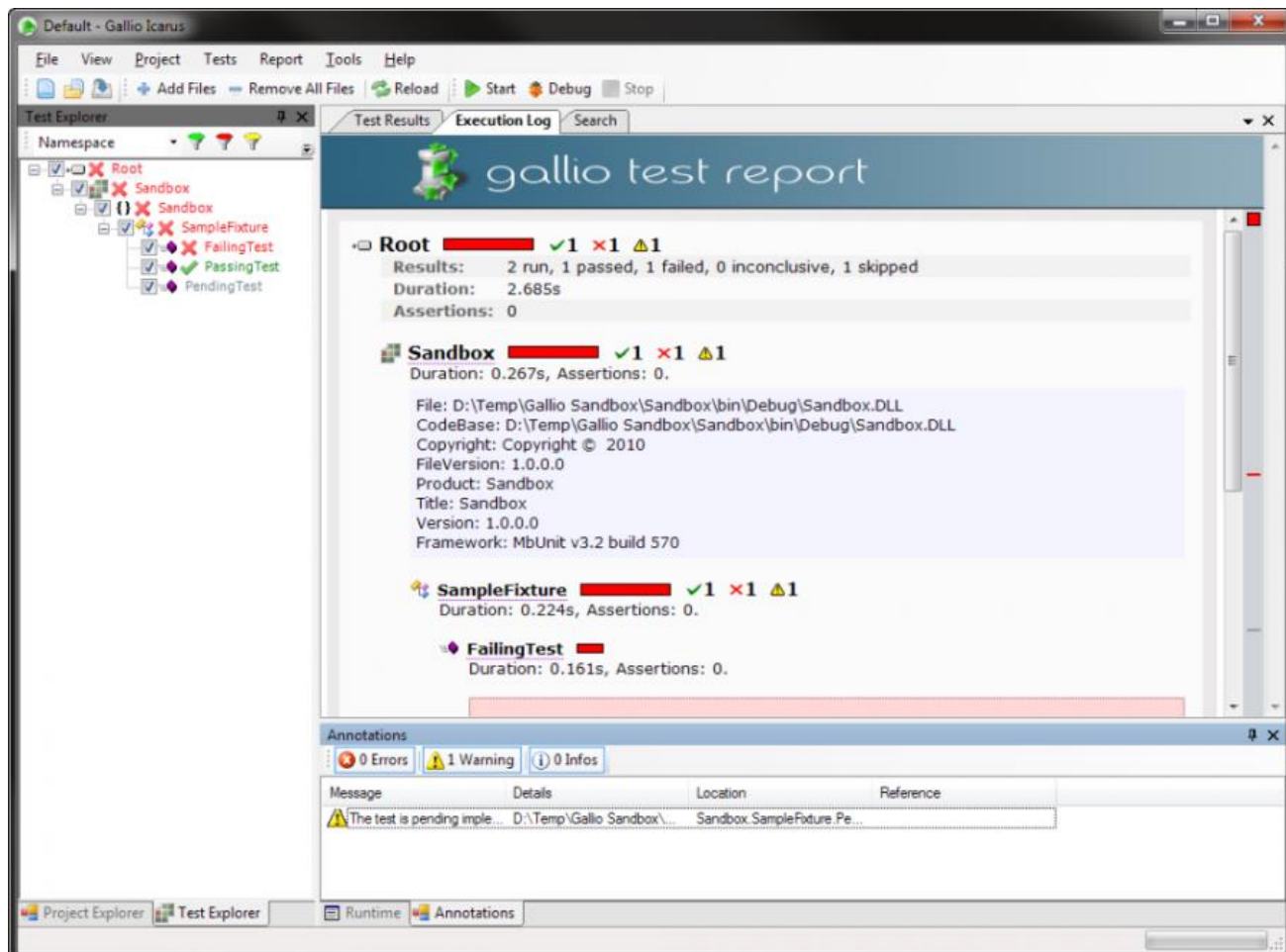
Тъй като софтуерните системи стават все по-големи и се разширяват с всеки изминал месец, за да се гарантира качеството на старата функционалност- “Agile”, методологиите разчитат много на различни видове автоматичното тестване - unit, integration, system.

Настоящата дипломна работа разглежда проблема за проектиране и реализация на десктоп приложение, чрез което могат да се изпълняват тестове на отдалечени машини, а след това резултатите да се изобразяват и да се пазят централизирано. Поддържат се различни потребителски роли. Само определени потребители ще може да настройват системата, други да изпълняват тестовете или да преглеждат резултатите от тях.

Приложението би било от голяма полза за компания, която иска бързо да предложи нови версии на продукта си. Чрез поддържането на голям набор от тестове ще може да си гарантира стабилността му и липсата на критични дефекти. Хора без специални технически познания ще могат да използват системата за изпълнение на тестове, за да следят качеството на нейното разработване. Възможността различен набор от тестове да се изпълнява на различни машини ще допринесе за по-бързото откриване на критични проблеми.

2. Обзор на съществуващи подобни приложения

The Gallio Automation Platform (<http://gallio.org/>)



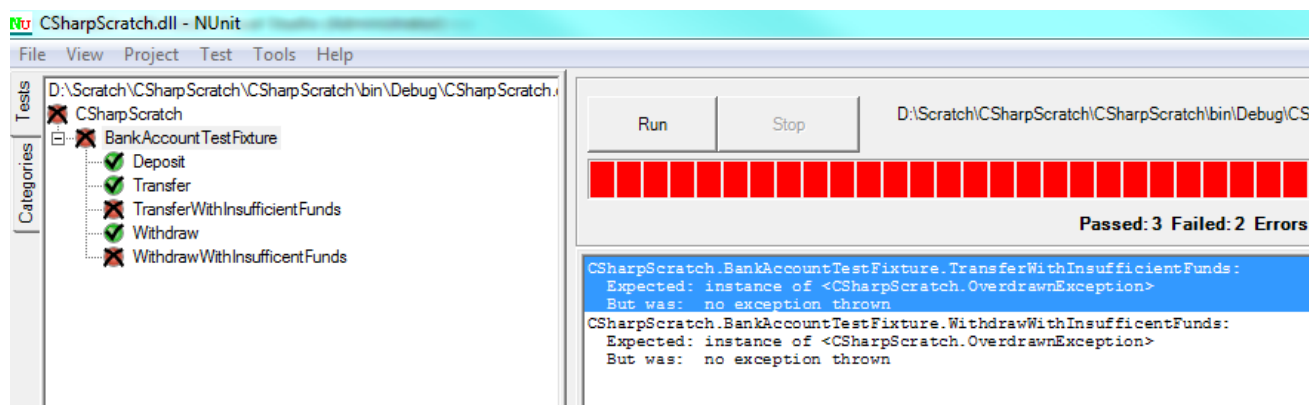
Фиг. 2.1: Потребителски интерфейс на Icarus UI runner част от Gallio Automation Platform

Gallio е много добра платформа за разработка на unit тестове. Icarus UI runner позволява на потребителя да добави DLL на компилирания си проект. След това програмата показва във формата на дърво всички класове и тестовите в тях. Маркираните от тях се изпълняват на клиентската машина и може веднага да се разбере резултата им. Ако намерим грешка, може да пуснем програмата в debug mode. Icarus има интеграция с Visual Studio 2010 и изпълнението ще спре, ако в кода са поставени break points. В резултатите има и пълен stack trace от изпълнението. Когато е указан ред в кода, той е под формата на линк и при натискането му се отваря файлът, където се намира проблемът. Gallio Automation Platform притежава и console runner. Чрез определени команди може да се настрои да изпълни тестовите ни и да генерира доклад с резултатите. ^[1]

За да може да се използва програмата, трябва предварително да се генерира DLL на тестовия проект. Тестовите могат да се изпълнят само на клиентската машина, която може да

се окаже слаба за определен вид(performance, load). Не се поддържа интеграция с Source Control системи. Генерираните резултати не се пазят централизирано и не може да се правят сравнения между различните изпълнения.^[1]

NUnit (<http://nunit.org>)



Фиг. 2.2: Потребителски интерфейс на Visual NUnit runner

NUnit е широко използван test framework за писане на unit тестове. Също така е open source. Тестовите написани на него могат да бъдат изпълнени по няколко начина- Visual NUnit, NUnit Console Runner, VS Visual NUnit(Extension). За да може да се видят във Visual NUnit трябва предварително да разполагаме с билднат DLL от проект. Както Gallio, тестовите се изпълняват на клиентската машина и резултатите от тях се виждат веднага. Налична е интеграция с Visual Studio. При натискане на линк към код, той се отваря в VS Editor. За да има генериран доклад с резултати, тестовите трябва да се пуснат през NUnit Console Runner. Те не се пазят на централно място и не може да се прави сравнение с предходни пускания. За момента показаните NUnit програми поддържат само тестове писани на NUnit Framework. Няма възможност за отдалечено изпълнение. Ограничена и остаряла документация.^[2]

PNUnit Parallel NUnit (<http://www.plasticscm.com/infocenter/technical-articles/pnunit.aspx>)

```
C:\pnunit>launcher test.conf
Test 1 of 1
INFO - Starting Testing test Testing on localhost:8080
INFO - Result for TestGroup Testing, Test Testing: PASS
===== Tests Results for Parallel TestGroup Testing =====
<1>Name: TestLibraries.Testing.EqualTo19
Result: SUCCESS      Assert Count: 0   Time:      0

Summary:
Total: 1
Executed: 1
Failed: 0
Success: 1
% Success: 100
Bigger Execution Time: 0 s
Launcher execution time: 1.1875 seconds
C:\pnunit>
```

Фиг. 2.3:Стартиране на PNUnit

Parallel NUnit е платформа, която се явява разширение на NUnit, чрез която може да се изпълняват тестовите паралелно на различни машини. Това би помогнало по-бързо да се изпълнят всички налични и да се разбере дали промените върху разработвания софтуер са създали нови критични дефекти. Също се предоставя възможност за изпълнение на по-мощни и изискващи много ресурси тестове. [3]

За да може да се работи с PNUnit, трябва да се добавят неговите DLLs към проекта и да се използват определени атрибути и конфигурации, за да може да се настрои. Няма визуален runner, тестовите могат да се стартират единствено през конзолата. Отново както при NUnit може да се изпълнява единствено код, използващ NUnit Framework. Библиотеката е Open Source и не се знае кога биха били добавени нови неща към нея. Документацията ѝ е оскъдна. Изпълнението на отдалечени машини изисква инсталирането на агентско приложение. Липсва интеграция с Source Control системи. Не се поддържа централизирано пазене на резултатите. Няма интеграция с NUnit приложенията за визуално представяне на тестовите. Няма интеграция с Visual Studio. [3]

Visual Studio & Team Foundation Server Build

Team Foundation Build uses a build process template defined by a Windows Workflow (XAML) file. The behavior of this template can be customized by setting the build process parameters provided by the selected template.

Build process template: **Busv4_BuildProcessTemplate.xaml** [Show details](#)

Build process parameters:

| | |
|-------------------------------------|---|
| 1. Required | |
| Items to Build | Build 4 project(s) for 1 platform(s) and configuration(s) |
| 2. Basic | |
| Automated Tests | Run tests in assemblies matching ***test*.dll |
| Build Number Format | \$(BuildDefinitionName)_\$(Date:yyyyMMdd)\$(Rev:.r) |
| Clean Workspace | All |
| Logging Verbosity | Normal |
| Perform Code Analysis | AsConfigured |
| Source And Symbol Server Settings | Index Sources |
| 3. Advanced | |
| Agent Settings | Use agent where Name=* and Tags is empty; Max Wait Time: 04:00:00 |
| Analyze Test Impact | True |
| Associate Changesets and Work Items | False |
| Copy Outputs to Drop Folder | True |
| Create Work Item on Failure | False |
| Disable Tests | False |
| Get Version | |
| Label Sources | True |
| MSBuild Arguments | |
| MSBuild Platform | X86 |
| Private Drop Location | Auto |

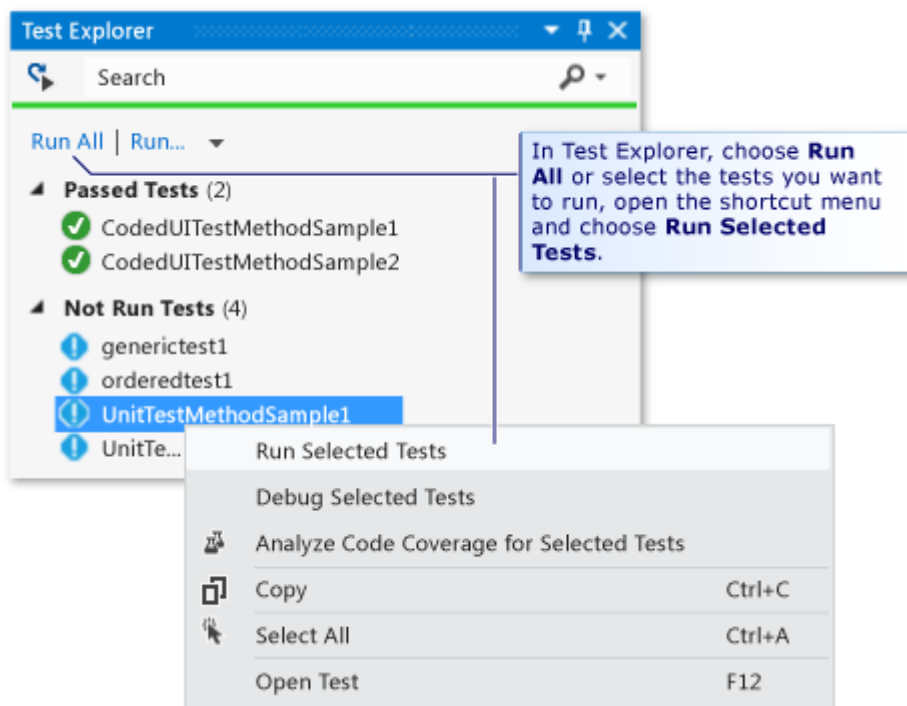
MSBuild Platform
Specify the platform of MSBuild.exe to use in the build process. Use Auto to detect the platform based on the current operating system.

Фиг. 2.4: Team Foundation Build Definition

Team Foundation Build предоставя възможност за създаване на последователност от действия, изпълнявани се след това на машини, на които има инсталирани TFS агенти. Visual Studio има подразбиращ се шаблон (default template), който включва интеграция с Source Control модула на Team Foundation Server. Кодът се тегли и билдва автоматично на зададената машина и след това се изпълняват всички тестове от тест листата или от всички проекти, намерени в сорс кода. Шаблонните (Template) файловете се дефинират на езиците - Windows Workflow Foundation (WWF) и .NET Visual Basic. Използването им помага да се разширят готовите шаблони (templates). Също може да се добавят проекти ,разработвани на MS Build. Visual Studio IDE има много добър визуален редактор за конфигурация на WWF и MS Build шаблони. Резултатите от изпълнените тестове могат да бъдат публикувани на сървър. Поддържа се асоцииране на тестовите към тестови сценарии, писани на тест мениджмънт системата, разработена от Microsoft, Test Manager. ^[4]

Тестовите, които може да се изпълняват , трябва да са разработвани на MS Test framework. Visual Studio IDE притежава UI Test Runner, чрез който може да се изпълняват тестове локално. Предоставя и възможност за дебъгване. Резултатите от локалните пускания се пазят в инсталирания на дадената машина SQL Server. При желание те могат да бъдат публикувани на даден сървър. ^[4]

Инсталацията и конфигурацията на всички приложения ,участващи в системата ,може да бъде доста сложна и да изисква специални познания. Разширяването на тези модули не може да бъде направено от нетехнически лица или такива, които не познават езиците ,използвани за конфигуриране. По- ниските версии на Visual Studio IDE нямат модул за изпълнение на тестове или връзка с Team Foundation Server. Настройването на различни роли и права за пускане на тестовите е твърде сложно. [4]



Фиг. 2.5: Visual Studio 2012 Test Explorer

За да работи цялата система, трябва да се инсталират твърде много продукти, чиято инсталация може да бъде твърде сложна и продължителна. Нетехнически лица трудно биха се научили как да използват продукта. [4]

Bromine 3 RC 1 (<http://sourceforge.net/projects/bromine/>)

Bromine 3 RC 1 е Open Source уеб приложение, което дава възможност за изпълнение на тестове, писани на Selenium Framework, на различни машини. Вътрешно Bromine 3 RC 1 използва Selenium Server (Selenium RC). Има приятен и лесен за употреба интерфейс. При изпълнение на тестовите се показва хронологията от изпълнените команди в реално време. Резултатите се пазят в локалната за сървър MySQL база. Има ограничени възможности за анализ на предходни пускания. Могат да се изпълняват единствено тестове писани на Selenium. Няма възможност за разширяне процеса на изпълнение. Няма интеграция със сорс контрол система. Не предоставя модул за определяне на ролите за различните потребители. Много ограничена и оскъдна документация. Замразено развитие. [5]



Фиг. 2.5: Bromine потребителски интерфейс

Изводи от обзора на подобни приложения

При изграждането на приложение за изпълнение на тестове едно от най-важните неща е дизайнът му да е лесен и разбираем за работа. Това се постига с опростен и добре изглеждащ интерфейс. Лесен достъп до всички важни функции. Друго нещо, което трябва да се вземе в предвид е дали приложението може да се инсталира и конфигурира лесно, колко достъпно е за нетехнически лица като мениджъри и тестери да го използват. Нужни ли са познания на даден език за програмиране? Важна част е интеграцията със сорс контрол системи, за да може винаги да се изпълнява последната версия на тестовете, а не само локалната. Повечето от разгледаните приложения поддържат само тестове писани на специфичен framework. За да се използва широко програмата, трябва да се поддържат най-различни библиотеки за разработка. При обзора се изясни, че изпълнението на различни машини би допринесло за качеството на разработвания от нас софтуер. Резултатите се виждат по-бързо и така може да се вземат по-бързо мерки, ако се открият критични дефекти. Би било много удобно за потребителя, ако може да вижда хронологията от различните машини в реално време. Друг важен аспект на отдалеченото изпълнение е, че тестовете могат да се изпълняват винаги на чиста и контролирана среда. Сорс кодът да се сваля и билдва всеки път. Така ще е сигурно, че винаги се тества последната версия на разработваната програмата.

Една от най-важните части е как се показват резултатите на потребителя. За да бъде максимално полезна системата, те трябва да се пазят централизирано, независимо на коя машина се пускат или изпълняват тестовете. По този начин хората управляващи екипа, който разработва софтуера, ще може да следят и анализират качеството във времето.

Не на последно място добро решение е, ако има админски панел, където да може да се променят настройките на приложението или да се одобряват ново-регистриралите се потребители.

3. Описание на използваните технологии

MS SQL SERVER

MS SQL SERVER е технология за проектиране и работа с релационни бази от данни. В много случаи предлага по-висока производителност от Access. Предлага възможност за поддръжка на много по-големи като обем бази от данни, с много повече информация. Много е ефективна в комбинация със сървърните операционни системи на Microsoft, тъй като изпълнява заявки паралелно, използвайки множество нишки в един процес. Така се намаляват изискванията за памет при наличие на множество потребители. ^[6]

При SQL SERVER може да се прави резервно копие на базата, частично или пълно, докато тя е в употреба. Това означава, че потребителите не трябва да прекратяват работата си с нея, за да може да се направи нейно резервно копие и така тя е достъпна непрекъснато. ^[6]

SQL SERVER може да се интегрира със системата за сигурност на Windows Server и така да се осигури достъп само на оторизирани потребители до базата. Така се улеснява прилагането на сложни схеми за сигурност. ^[6]

В случай на изключителни ситуации като проблеми с операционната система или внезапно спиране на захранването SQL SERVER, може да възстанови базата до последното и стабилно състояние за няколко минути. Така приложения от голяма важност могат да стартират отново възможно най-бързо. ^[6]

Обработката на транзакции е съществено изискване за система, проектирана да поддържа критични приложения като тези от финансовия сектор. Например, SQL SERVER поддържа атомични транзакции, които гарантират, че или всички операции в рамките на транзакцията се изпълняват успешно, или всички се отказват поради възникнала грешка. Консистентността и възстановяемостта на транзакцията са гарантирани дори при грешка по време на сложни операции от повече от един потребители. Всички промени в рамките на транзакция се считат като една задача. По дефиниция или цялата минава успешно и промените се потвърждават, или тя се отказва и направените до момента от нея промени се премахват. Поддържат се и синхронизирани транзакции, които могат да се изпълняват на повече от един сървър. Като консистентност се осигурява на всички сървъри. SQL SERVER намалява мрежовия трафик като обработва заявките на сървъра преди да изпрати резултата към клиента. Така обработката при клиент/сървър приложенията се извършва там, където може да се извърши най-добре, а именно на сървъра. ^[6]

ADO.NET Entity Framework

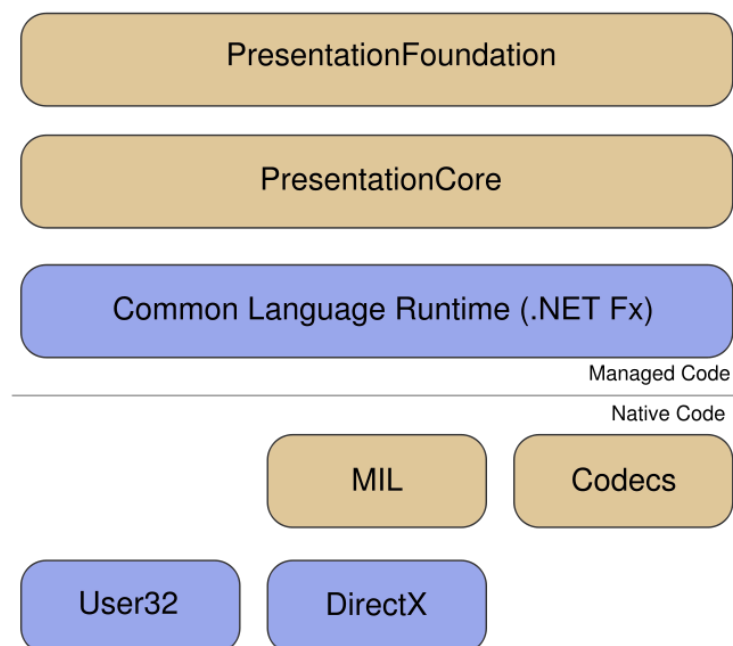
Entity Framework е съвременна технология за достъп до бази данни през приложения, работещи на платформата .NET на Microsoft. Тя предлага обектно ориентиран подход за работа. Таблиците в базата са представени като обекти, които могат да се манипулират програмно. Системата автоматично следи промените по обектите и ги отразява в базата, което улеснява операциите по актуализиране на информацията. Спестява се голямо количество код, което би трябвало да се напише и да се поддържа след това. Тъй като съответствието между

обектите и базата е описано декларативно, могат да се правят промени по схемата на базата, без това да наложи съществени промени по кода, който работи с нея. Entity Framework внася ниво на абстракция, което спомага за изолирането на базата данни от приложението. Заявките, които се пишат са на език, универсален за всички сървъри на бази данни. Това се постига тъй като заявките първо се отправят към системата на Entity Framework, която разбира този универсален синтаксис (LINQ или Entity SQL) и го превежда към синтаксиса характерен за съответния сървър (Oracle, SQL).^[7]

С Entity Framework се повишава и сигурността на приложението. Не се налага да се обработват (parse) заявки написани в текстов вид, което намалява възможността за SQL Injection атаки.^[7]

Windows Presentation Foundation (WPF)

Windows Presentation Foundation е основната технология за разработка на десктоп приложения за операционната система Windows и голяма част от Windows 7 е базирана на WPF. Всяко приложение писано на WPF е разделено на поне два слоя – Презентационен слой (потребителски интерфейс на приложението) и code-behind слой (съдържа самата програмна логика и функционалност на приложението съдържащ C# или Visual Basic .NET код). Използва XML базиран език- XAML за дефиниране на UI елементите на приложението. Предоставя много мощна структура за дефиниране на стилове и скинове. Много удобна система за data binding, която може да се използва за изграждане на сложни форми, използващи шаблони (templates).^[8]



Фиг. 3.1: Структура на Windows Presentation Foundation

Графичното изчертаване, което предоставя WPF, се базира на Direct X. За разлика от Windows Forms, WPF използва графичната карта за чертаене на потребителския интерфейс и по- този начин работи доста по-бързо. Освен това в зависимост от резолюцията, приложението се показва по различен начин, увеличава се автоматично и не стои на пиксели. Интеграцията с Direct X добавя и нови възможности за лесно използване на мултимедия, анимации и графики. Голяма част от кода на контролите писани на WP може да се преизползва. В интернет може да се намери много добра документация и много примери. Има много Open Source и безплатни теми, които могат да бъдат използвани за изграждане на външният вид на приложението. Поддържа Windows Forms, част от контролите могат да се преизползват. ^[9]

MSBuild

MSBuild е главната платформа, която използва Visual Studio IDE, за да изпълнява билд дефиниции чрез Team Foundation Server, както и за да билдва вътрешно проектите си. MSBuild компилира сорс кодът до MSIL(Microsoft Intermediate Language), който след това може да бъде изпълнен от CLR (Common Language Runtime). Използва се XML базиран синтаксис за дефиниране на действия. Има много големи възможности за логване на хронологията от изпълнени команди. Предоставя библиотеки за разширяване на процеса на логване, както и дефиниране на собствени команди. За да работят файловете, писани на MSBuild, нямат нужда от компилация. ^[10]

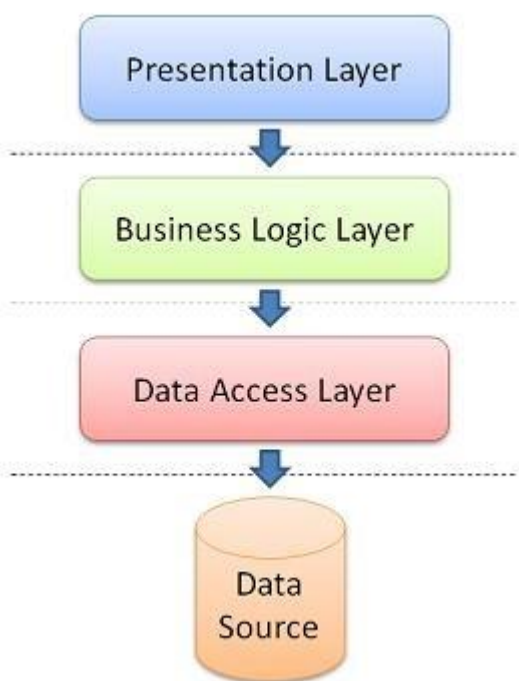
```
<Target Name="CreateWorkspace">
  <Exec Command="$(Tf) workspace /new /noprompt
/server:&quot;$(TfsServerUrl)&quot; &quot;$(WorkspaceName)&quot;" />
  <Exec Command="$(Tf) workfold /unmap
/workspace:&quot;$(WorkspaceName)&quot; $" />
  <Exec Command="$(Tf) workfold /map &quot;$(TfsPath)&quot;
&quot;$(LocalPath)&quot; /workspace:&quot;$(WorkspaceName)&quot;" />
  <Message Text="Workspace '$(WorkspaceName)' created sucessfully"/>
  <Exec Command="$(Tf) workfold
/workspace:&quot;$(WorkspaceName)&quot;" />
</Target>
```

Чрез лесния си синтаксис предоставя удобен начин за деклариране на конзолни команди, на които могат да се подават параметри и да се изпълняват една след друга. Освен това чрез добрата система за логване, може да се следи техният изход или да се филтрира по желание. ^[10]

4. Функционално описание

Многослойна архитектура

Приложението е изградено с многослойна архитектура(Фиг. 4.1). Схематично представена тя изглежда по следния начин:



Фиг. 4.1 N-Tier Architecture – схема

В случая думата „слой“(layer) се използва в значение на преизползваема част от кода, която изпълнява определена функция. В .NET слойът обикновено се реализира като отделен проект. Той има за задача да взаимодейства с останалите слоеве за изпълнение на определена цел. В едно приложение, в което презентационният слой трябва да извлече информация от база данни ,той би извикал друг слой, който да вземе информацията и да му я предаде, вместо директно да взаимодейства с базата.^[11]

Ключов компонент в повечето приложения са данните. Тези данни трябва по някакъв начин да се предадат до презентационния слой, за да бъдат показани на потребителя. Слойът за данни (Data Access Layer) е отделен компонент, чиято цел е да вземе исканата информация от базата данни и да я предаде на викащия модул. Използвайки този подход, данните могат да бъдат логически преизползвани. Това означава, че две или повече части от приложението, които трябва да изпълнят една и съща заявка към базата, могат да го направят като просто извикат даден метод от слоя за данни. Така не се налага всеки път тази заявка да се пише на наново, което улеснява поддръжката.^[11]

Въпреки, че презентационният слой може директно да комуникира със слоя за данни, той обикновено минава през още един слой наречен Business Rules или Business Logic Layer. Този слой е от съществено значение, тъй като той валидира входните данни преди да извика метод от слоя за данни. Така се осигурява коректността на входните параметри, а с това и на изходните в голяма част от случаите. В този слой обаче не се извършва само валидация на данните. Всъщност е добре в него да се сложи възможно по-голяма част от логиката на самото приложение, което дава възможност тя да се използва и в други проекти. Една от съществените причини, които налагат логиката да може да се преизползва е фактът, че приложения, които в началото са били малки, често с времето се разрастват значително като функционалност. ^[11]

Презентационният слой (Presentation Layer) е този, който потребителите виждат и с който взаимодействат директно. Най-добре е в този слой да има колкото е възможно по-малко бизнес логика.

Най-голямото предимство на разделянето на приложението на слоеве е повишената възможност за преизползване на кода. Това е от голямо значение, тъй като с времето приложението се разраства като функционалност и може да претърпи сериозни промени. Започнало като десктоп ,след време част от функционалността може да се премине в мобилно приложение. Част от функционалността може да се раздели например между web site и web или windows service, който се изпълнява на сървър. ^[11]

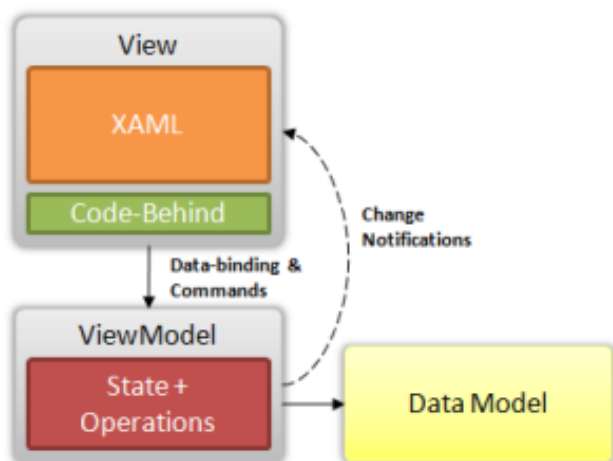
Разделянето на приложението на слоеве си има и своите недостатъци. Те са главно свързани с повечето време, което отнема да се напише приложение, чиято основна част от логиката е в business rules слой. Удълженото време идва главно от там, че се налага създаването на няколко множества обекти – data layer обекти и business layer обекти, вместо кодът директно да се вложи в презентационната част. ^[11]

Въпреки това разделянето на приложението на слоеве е по-добрият подход, тъй като в противен случай с нарастването на написания код се случва следното: кодът се копира често на различни места или се преизползва под формата на класове, които лесно биха могли да бъдат обособени в бизнес слой; код който е сходен ,често се копира с леки промени като става трудно проследяването на евентуални повторения; приложението става трудно за поддръжка и автоматично тестване. ^[11]

View Model Viewmodel (VMVM) архитектура

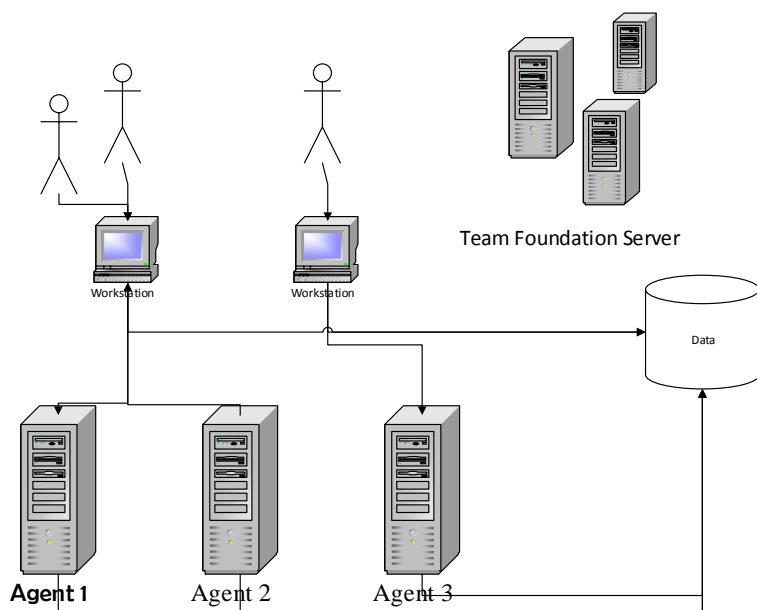
Презентационният слой от своя страна също може да бъде разделен логически на няколко части, използвайки View Model Viewmodel (VMVM) архитектура (Фиг. 4.2.).

Всяка страница от приложението е разделена на три части- View, Viewmodel, Data Model. View е XAML страницата и нейният code behind файл, където са дефинирани контроли на страницата (UI елементите).



Фиг. 4.2. View Model Viewmodel (VMVM) архитектура – схема

Използвайки мощните възможности на WPF за Data Binding и използване на шаблони за построяване на презентацията на дадена страница, може много лесно да се създадат сложни контроли. Може да се свърже шаблона на дадена контрола към нейният Viewmodel, откъдето ще си вземе данните. Viewmodel се явява специален обект за дадена логика, където за него са добавени всички нужни обекти и са премахнати останалите. Вътрешно може да се извикват методи от Data Access Layer, за да се инициализира даденият модел. WPF също използва VMVM архитектура, което прави работата още по- лесна. Има готови класове, които връзват свойствата(properties) и обектите от Viewmodel с View и при промяна те автоматично се обновяват в презентационния слой. Изолирането на данните от презентацията, би позволило да се прехвърли логиката на приложението по- лесно, ако се наложи. Например, ако е имало десктоп приложение и трябва да се превърне в мобилно или уеб. ^[12]



Фиг. 4.3. Deployment диаграма

Deployment диаграма

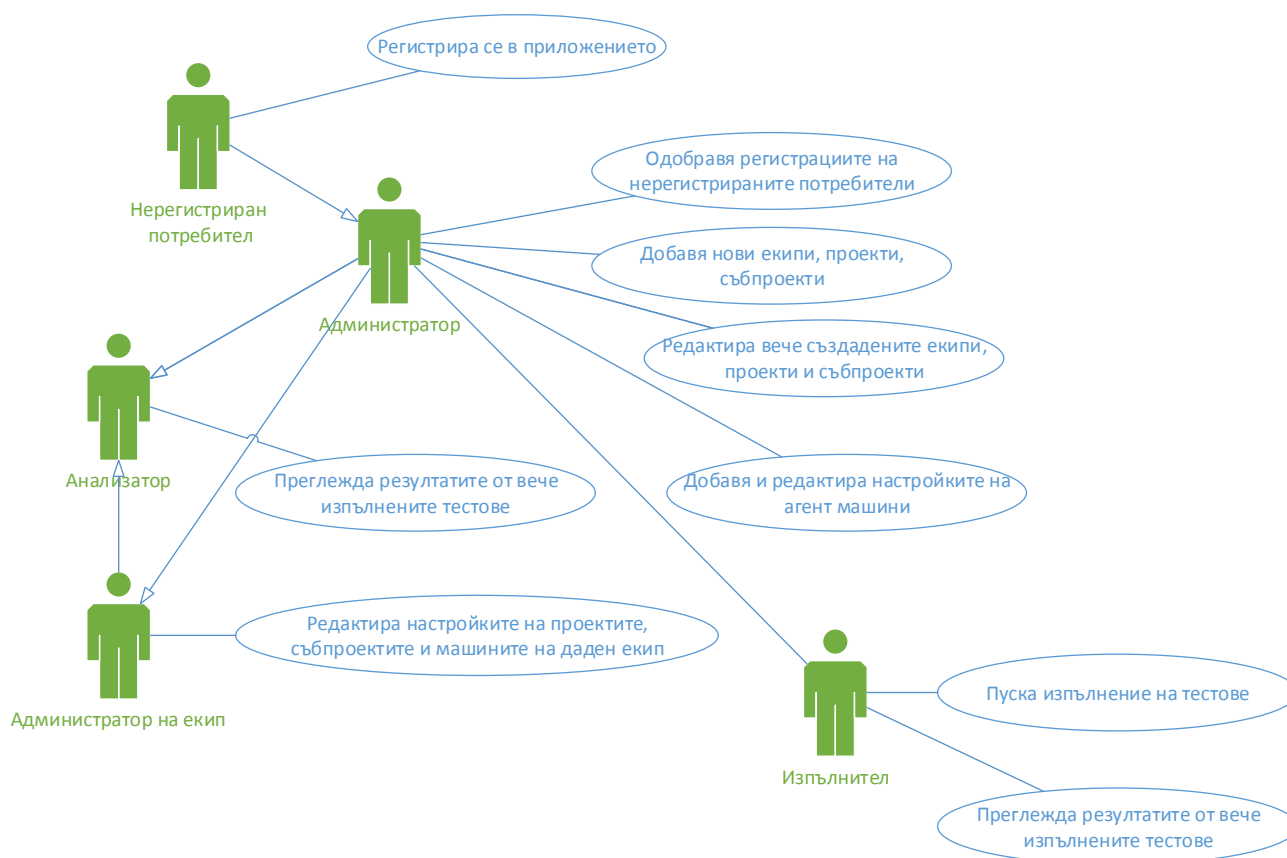
По-горе е показана диаграма как различните потребители на системата си комуникират с агентските приложения и базата.

На всяка клиентска машина може да се пазят настройките на повече от един потребител. Те включват цвят и тема на приложението, пътища до проектите или до специфичен код. В зависимост от потребителското име, програмата ще прочете тези настройки и ще покаже специфичните данни.

Множество хора от даден екип може да използват програмата едновременно. Всеки екип разполага с X на брой агентски машини, на които могат да се изпълняват тестове. Всеки потребител може да се свърже с повече от една машина едновременно и да наблюдава в реално време действията извършвани на нея. Една машина може да бъде използвана само от един човек през даден период от време. Ако някой друг от екипа иска да пусне своите тестове, ще трябва да намери свободна машина.

Клиентските приложения и агентите си споделят една централизирана база, която се намира на отделна машина. Всяка машина и агент се свързват към общия Team Foundation Server, за да изтеглят последната версия на сорс кода.

Use Case диаграма



Фиг. 4.4. Use Case диаграма

По-горе е представена диаграма ,изобразяваща различните потребители, които системата разпознава и какво всеки от тях може да прави в нея.

Нерегистрираният потребител може да се регистрира като попълни съответната форма. Трябва да отбележи в кои екипи иска да участва и каква роля ще заема. Задължително е да запише своят имейл, където след това ще получи активационен код. Не може да използва системата докато акаунта му не се одобри от администратор.

Администраторът от своя страна може да преглежда постъпилите регистрации и да редактира настройките им. Има достъп до модулите за добавяне на екипи, проекти, подпроекти и агентски машини. Може да променя вече съществуващи данни. Няма пряк достъп до секциите за изпълнение на тестове и свързаните с тях резултати.

Анализаторът има право да вижда резултатите от вече изпълнените тестове на екипите, в които е член. Няма право да пуска нови серии или да променя настройките на проектите и машините.

Администраторът на екип има право да добавя нови проекти и подпроекти към текущите. Може да променя и добавя нови агентски машини към вече съществуващите. Може да изпълнява тестове и да вижда свързаните с тях резултати.

Потребителите с роля – „изпълнител“ нямат право да променят настройките на своите екипи. Могат да изпълняват тестове и да следят резултатите свързани с тях.

Представяне на по-важните класове и техните по-важни методи

ServerAgent

```
public class ServerAgent : BaseLogger
{
    ...
    public static void Main(string[] args)
    {
        while (true)
        {
            try
            {
                InitializeSettings();
                clientProcessMessageThreadWorker = Task.Factory.StartNew(() =>
                ProcessClientMessage(agentListener), token, TaskCreationOptions.LongRunning,
                TaskScheduler.Current);
                ...
                messageLoggerThreadWorker = Task.Factory.StartNew(() =>
                SendLogMessages(agentListener, messagesToBeSend), token, TaskCreationOptions.LongRunning,
                TaskScheduler.Current);
                ...
                executeCommandThreadWorker = Task.Factory.StartNew(() =>
                ExecuteCommands(), token, TaskCreationOptions.LongRunning, TaskScheduler.Current);
                BaseLogger.Log.Info(ExecutionTaskStartedMsg);
                msBuildLogListnerThreadWorker = Task.Factory.StartNew(() =>
                ATACore.TcpWrapperProcessor.TcpMsBuildLoggerProcessor.ProcessMsBuildLoggerMessage(messages
                ToBeSend, msBuildLogSettings), TaskCreationOptions.LongRunning);
                Task.WaitAll(new Task[] { clientProcessMessageThreadWorker,
                executeCommandThreadWorker, messageLoggerThreadWorker, msBuildLogListnerThreadWorker });
                ...
            }
        }
    }
    ...
}
```

По-подробен листинг на кода на класа - [Приложение 1](#)

Server Agent е конзолно приложение, което се стартира на агентските машини, за да обработва съобщенията от свързаните с него клиентски приложения. При стартиране се пускат четири паралелни нишки. Първата, наречена `clientProcessMessageThreadWorker`, слуша на порт 8888 за съобщения, пристигащи от другите машини. След това ги обработва, сортира и в зависимост от вида им ги добавя в специалната паралелна опашка, наречена `commandsToBeExecuted`. Тя е от тип `ConcurrentQueue<string>`, който позволява множество нишки да добавя и четат едновременно от нея, без да се получават dead locks. Втората нишка, която се пуска на от агентското приложение, е `executeCommandThreadWorker`, която взима текущата команда от опашката и стартира нужните процеси. Командите, които се получават от клиентското приложение, са в XML формат. След това чрез помощни методи се конвертират до съответните класове, за да може да се използват по- лесно. (Фиг. 4.5.)

```

XmlSerializer deserializer = deserializer = new
XmlSerializer(typeof(MessageArgsParseResult));
    StringReader textReader = new StringReader(currentCommandXml);
    MessageArgsParseResult msgArgParseResult =
(MessageArgsParseResult)deserializer.Deserialize(textReader);

```

Фиг. 4.5. Конвертиране на XML команда до спомагателен клас

Третата нишка от приложението чете от втората опашка, наречена `messagesToBeSend`. Създава обект от тип съобщение, конвертира го до XML и го изпраща до клиентското приложение, за да бъде показано в хронологията от изпълнени команди.

Последната паралелна нишка стартира TCP IP listener към IP на текущата машина на порт 8889. При стартиране на процес от MS Build винаги се прикрепя специален TCP IP logger, който се свързва с тази нишка и изпраща информация за текущият процес. Тези съобщения се обработват и се слагат в опашката `messagesToBeSend`.

При прекратяване на връзката между агента и клиентското приложение се извиква методът `CancelCurrentlyExecutingProcess`. Вътрешно той спира текущо изпълняващия се процес и извиква `CancellationToken` на всички нишки, за да ги спре. След като приключат тази серия от действия, агентът е готов да бъде използван от друго клиентско приложение.

Класът наследява `BaseLogger`, който е помощен модул за записване на важни съобщения към общият лог на приложението.

TcpClientWrapper

```

public class TcpClientWrapper : BaseLogger
{
    public void SendMessageToClient(TcpClient tcpClient, string messageToSend)
    {
        NetworkStream ns = tcpClient.GetStream();
        Byte[] bytesToSend = Encoding.ASCII.GetBytes(messageToSend);
        int totalBytes = bytesToSend.Length;
        int startIndex = 0;
        byte[] intBytes = BitConverter.GetBytes(totalBytes);
        if (BitConverter.IsLittleEndian)
            Array.Reverse(intBytes);
        byte[] result = intBytes;
        ns.Write(result, 0, result.Length);
        do
        {
            int endIndex = (totalBytes - startIndex) > 1000 ? 1000 : totalBytes -
startIndex;
            ns.Write(bytesToSend, startIndex, endIndex);
            ns.Flush();
            startIndex += endIndex;
        }
        while (startIndex != totalBytes);
        Log.InfoFormat("Sending >> ", messageToSend);
    }
}

```

...

По-подробен листинг на кода на класа - [Приложение 2](#)

Този клас е от Бизнес слоя на приложението. Притежава методи за изпращане и четене на съобщения чрез TCP IP.

Методът `SendMessageToClient` приема TCP Client, който вече е свързан с определен listener и съобщението, което трябва да бъде доставено. Тъй като определени съобщения могат да бъдат по- големи от 8к байта, методът преобразува съобщението до масив от байтове. В първите четири байта на съобщението слага общата дължина на изпращаните байтове. След това започва да ги изпраща на парчета по 1000.

`ReadLargeClientMessage` първо прочита големината на съобщението, след това започва да чете по 1000 байта от мрежовият поток. Методът `NullRemover` се използва да премахва нулевите терминиращи стрингове от полученият текст. Процесът продължава докато се срещне специалната поредица от символи "\$\$", която значи край на съобщението.

CommandLineExecutor

```
public class CommandLineExecutor
{
    public const string MSBUILD_PATH =
@"C:\Windows\Microsoft.NET\Framework\v4.0.30319\MSBuild.exe";

    public Process ExecuteMsbuildProject(string msbuildProjPath, IPAddressSettings
ipAddressSettings, string additionalArgs = "")
    {
        string currentAssemblyLocation =
System.Reflection.Assembly.GetExecutingAssembly().Location;
        string currentAssemblyFullpath = String.Format(currentAssemblyLocation,
@"\\AutomationTestAssistantCore.dll");
        string additionalArguments = BuildMsBuildAdditionalArguments(msbuildProjPath,
ipAddressSettings, additionalArgs, currentAssemblyFullpath);
        ProcessStartInfo procStartInfo = new ProcessStartInfo(MSBUILD_PATH,
additionalArguments);
        procStartInfo.RedirectStandardOutput = false;
        procStartInfo.UseShellExecute = true;
        procStartInfo.CreateNoWindow = false;
        //procStartInfo.UseShellExecute = false;
        //procStartInfo.CreateNoWindow = true;
        Process proc = new Process();
        proc.StartInfo = procStartInfo;
        proc.Start();
        return proc;
    }
}
```

...

По-подробен листинг на кода на класа – [Приложение 3](#)

Класът `CommandLineExecutor` е част от бизнес слоя на приложението. Главно се използва от агентската програма за изпълнение на серия от команди. Съдържа серия методи, които генерират конзолни команди. `ExecuteMsbuildProject` стартира MS Build с определени параметри. Към процеса се прикрепя специалният `TcpIpLogger`, който изпраща подробности

за текущо изпълнявания се процес. Процесът се стартира на заден план и не се показва конзолен прозорец. За да се зададе правилният път до [TcpIpLogger](#), се използват класове от System.Reflection.

MemberManager

```
public class MemberManager
{
    public const string Salt = "5B-4B-05-AD-A5-1D-3E-C5-E5-62-8D-32-74-B4-8A-0A-49-74-31-25-E4-8C-77-34-21-66-FE-64-E8-14-8C-74";
    ...

    public Member CreateUser(ATAEntities context, string userName, string password, string email, string tfsUserName, string role, List<string> teams, string comment)
    {
        string hashedPassword = CreatePasswordHash(password);
        MemberRole currentMemberRole =
        ATACore.Managers.MemberRoleManager.GetMemberRoleByRoleName(context, role);
        Member newMember = new Member()
        {
            UserName = userName,
            Password = hashedPassword,
            Email = email,
            Comment = comment,
            TfsUserName = tfsUserName,
            MemberRoleId = currentMemberRole.MemberRoleId
        };
        newMember.StatusId = (int)Statuses.ToBeApproved;
        AddTeamsToNewMember(context, teams, newMember);
        context.Members.Add(newMember);
        context.SaveChanges();

        return newMember;
    }
    ...
}
```

По-подробен листинг на кода на класа – [Приложение 4](#)

Класът MemberManager е част от бизнес слоя на приложението, съдържа множество методи за работа с потребителски обекти. Методът CreateUser създава нов потребител в статус ToBeApproved. Използва се Entity Framework, за да се запише обекта в базата данни. За да се защити паролата на потребителя, не се записва в чист вид. Използва се метода CreatePasswordHash, за да се генерира специална hash стойност. Използва се класа PasswordDeriveBytes от библиотеката System.Security.Cryptography, за да се криптира hash стойността.

Когато администратор иска да одобри дадена регистрация, се извиква методът ApproveUser, който сменя статуса на потребителя и му изпраща имейл с активационен код, който може да се използва, за да се активира акаунта.

Member

```
public partial class Member
{
    public Member()
    {
        this.SuiteResultRuns = new HashSet<SuiteResultRun>();
        this.ActivationCodes = new HashSet<ActivationCode>();
        this.Teams = new HashSet<Team>();
        this.ExecutionResultRuns = new HashSet<ExecutionResultRun>();
    }

    public int MemberId { get; set; }
    public string UserName { get; set; }
    public string Password { get; set; }
    public string Email { get; set; }
    public int MemberRoleId { get; set; }
    public int StatusId { get; set; }
    public string Comment { get; set; }
    public string TfsUserName { get; set; }
}
```

Това е класът който се използва в слоя за данни за капсулиране на информацията за един потребител.

RegistryManager

```
public class RegistryManager
{
    ...

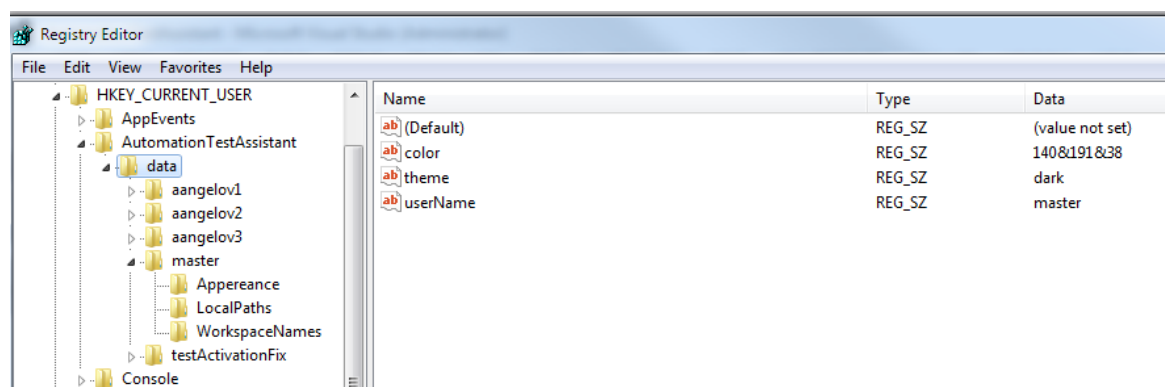
    public void WriterLocalPathToRegistry(string currentUserName, string tfsPath, string
localPath)
    {
        RegistryKey ata = Registry.CurrentUser.CreateSubKey(MainRegistrySubKeyName);
        RegistryKey dataR = ata.CreateSubKey(DataRegistrySubKeyName);
        RegistryKey currentUserR = dataR.CreateSubKey(currentUserName);
        RegistryKey localPathsR =
currentUserR.CreateSubKey(LocalPathsRegistrySubKeyName);
        localPathsR.SetValue(tfsPath, localPath);
        localPathsR.Close();
        currentUserR.Close();
        dataR.Close();
        ata.Close();
    }
    ...
}
```

По-подробен листинг на кода на класа – [Приложение 5](#)

RegistryManger е клас от бизнес слоя на приложението. Той се използва главно от клиентското приложение за записване и четене на важни данни в регистрите на Windows. За да може приложението да се използва от повече от един потребител на машина, записва персоналната информация в регистрите. За да може да се видят тестовите от проектите на даден екип, трябва да се определят къде на хард диска ще се сваля последната версия на всеки един от тях. Прави се mapping между всеки Team Foundation Server и локален път. Под

главната папка на приложението имаме папката data, където се пазят три главни регистри. Кой е текущият потребител, текущата тема и цвят. На поднива се намира информация за всеки потребител, използвал клиентското приложение на текущата машина. В папката Appearance е информацията за темата и цвета, които последно е използвал дадения потребител. Когато се логне в програмата, се извиква методът GetTheme, който отваря последователно регистрите – AutomationTestAssistant, data, “име на потребителя”, Appearance и чете данните за цвят и тема.

При смяна на външния вид на приложението, се извиква метода WriterCurrentThemeToRegistry за да се запишат новите стойности.



Фиг. 4.5. Папки на регистрите, използвани от клиентското приложение

Класът съдържа подобни методи за работа с локални пътища и workspaces.

ProjectSettingsLoadingView

```
public partial class ProjectSettingsLoadingView : UserControl, IContent
{
    ...

    private void On_Loaded(object sender, RoutedEventArgs e)
    {
        if (!IsLoaded)
        {
            MainWindow.MsBuildLogListnerThreadWorker =
                Task.Factory.StartNew(() =>
                ATACore.TcpWrapperProcessor.TcpMsBuildLoggerProcessor.ProcessMsBuildLoggerMessage(MainWindow.messagesToBeSend,
                MainWindow.LocalMsBuildLogIpSettings),
                TaskCreationOptions.LongRunning);

            MainWindow.MessageLoggerThreadWorker = Task.Factory.StartNew(() =>
            DisplayMsBuildLog());
            MainWindow.GetBuildThreadWorker = Task.Factory.StartNew(() =>
            ProcessSelectedProjectInfos());
            IsLoaded = true;
        }
    }
}
```

По-подробен листинг на кода на класа – [Приложение 6](#)

Класът `ProjectSettingsLoadingView` е част от презентационния слой на приложението. Когато потребителят избере кои проекти иска да зареди, се стартира текущата контрола, която показва хронология на зареждащите операции. При зареждане на контролата се извиква методът `On_Loaded`. Паралелна работа започват три нишки. Едната зарежда `MS Build Logger`, който слуша изпълняващите се команди и ги зарежда в паралелната опашка – `messagesToBeSend`. Втората нишка има за задача да чете от тази структура от данни и да показва съобщенията на потребителя. Използваме делегата `UpdateProgress`, за да показваме съобщенията в текстовите контроли, тъй като само главната UI нишка има право.

Третата нишка `GetBuildThreadWorker` върши основната работа. Обхожда обектите от `AdminProjectSettingsViewModel`, за да извлече нужната информация. Вътре се изглежда масив от нишки, които се изпълняват последователно на края на изпълнението на метода. Те извършват различни команди за изтеглянето и билдването на избраните от потребителя проекти. Първо се изтриват асоциираните към проекта `workspaces`. Изтриват се старите файлове. След това се създава нов `workspace`. Взема се последна версия на файловете в оказаната от потребителя локална папка. След това се повтарят същите команди за всички допълнителни подпроекти. Билдва се главният проект и се генерират неговите DLLs. От тях се извличат всички методи, които могат да бъдат изпълнени на агентските машини и се показват на потребителя в следващата страница.

Когато всички второстепенни нишки се изпълнят, се използва делегата `NavigateToNextPage`, за да се прехвърлим на следващата страница. Отново се използва тази техника, защото навигацията трябва да се извършва от главната UI нишка

5. Проектиране на базата данни

За проектиране на базата данни в настоящата дипломна работа се използва `SQL Server` и графичната среда за работа с него – `Management Studio`. Тя позволява бързо и лесно изпълнение на операциите по създаване на базата, създаване на таблици и техните полета, както и създаване на връзките между таблиците. Следва схема на базата данни и описание на таблиците и техните полета.

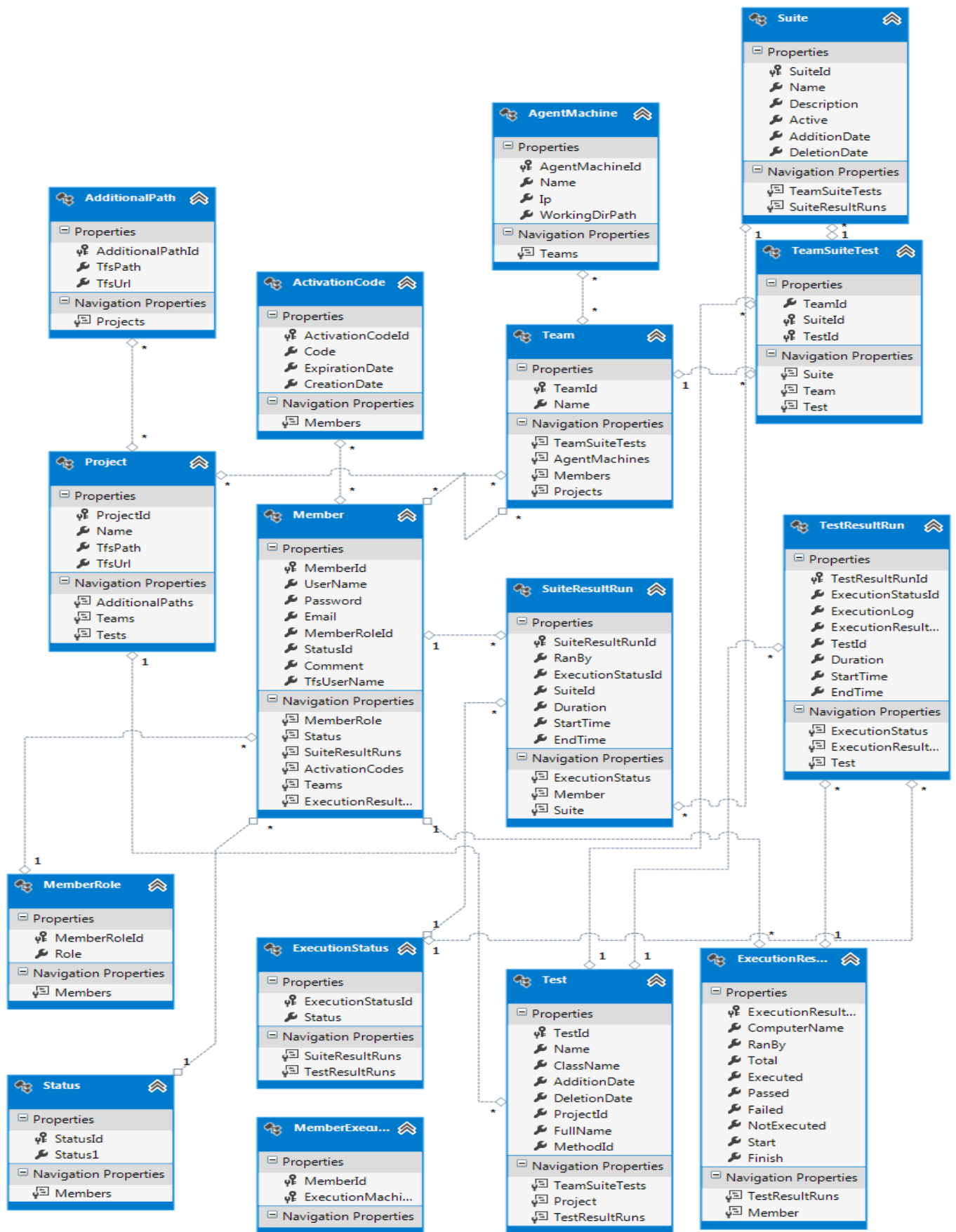


Таблица ActivationCode

Таблицата изглежда по следния начин:

| Име на колона | Тип на данните | Позволяване на null стойности |
|------------------|----------------|-------------------------------|
| ActivationCodeId | Int | Не |
| Code | nvarchar(50) | Не |
| ExpirationDate | date | Не |
| CreationDate | date | Не |

В базата се пази информация за кодовете за активация. Кога са създадени, до кога са валидни. Потребителите могат да използват кода, за да активират акаунта си.

Полетата в таблицата са:

- ActivationCodeId – първичният ключ на таблицата представлява номера на записа в нея.
- Code – текстова репрезентация на кода за активация.
- ExpirationDate – дата до кога е валиден кодът.
- CreationDate – датата кога е създаден кодът.

Таблицата има връзка много с таблицата Member.

Таблица Member

Таблицата изглежда по следния начин:

| Име на колона | Тип на данните | Позволяване на null стойности |
|---------------|----------------|-------------------------------|
| MemberId | Int | Не |
| UserName | nvarchar(50) | Не |
| Password | nvarchar(MAX) | Не |
| Email | nvarchar(50) | Не |
| MemberRoleId | int | Не |
| StatusId | int | Не |
| Comment | nvarchar(MAX) | Да |
| TfsUserName | nvarchar(50) | Не |

Тази таблица се използва за съхранение на специфични данни за всеки потребител. Запис в нея се прави при подаване на заявка за регистриране в клиентското приложение, ако вече няма данни за съответния потребител.

- MemberId - първичният ключ на таблицата представлява номера на записа в нея.
- UserName – името, което ще използва потребителя, за да влезе в системата

- Password – криптирания hash на паролата
- Email –имейлът, на който ще бъде изпратен кодът за активация
- MemberRoleId – Foreign key към таблицата MemberRole, определящ ролята на текущия потребител
- StatusId - Foreign key към таблицата Status, определящ статуса на текущия потребителя
- Comment – коментар към администратора, който ще одобрява регистрацията
- TfsUserName – потребителското име, което ще бъде използвано за вход към TFS

Таблицата има връзка много с Activation Code. Един потребител може да има повече от един активационен код, тъй като кодът може да изтече. Има връзка много към едно с таблиците Status и MemberRole. Различните потребители могат да имат различни статуси и роли. Също така един екип може да има много членове, затова имаме таблицата TeamMember, която обединява двете. За да може да се прави по-добър анализ на данните от изпълнението на тестовете и листите, за всяко пускане се пази информация кой потребител го е стартирал. Затова имаме връзка едно към много в таблиците SuiteResultRun и ExecutionResultRun.

Таблица MemberRole

Таблицата изглежда по следния начин:

| Име на колона | Тип на данните | Позволяване на null стойности |
|---------------|----------------|-------------------------------|
| MemberRoleId | Int | Не |
| Role | nvarchar(50) | Не |

Текущата таблица пази различните потребителски роли. Те се използват в клиентското приложение, за да се определя достъпът на текущия потребител до различните модули.

Има връзка едно към много с таблицата Member.

- MemberRoleId - първичният ключ на таблицата представлява номера на записа в нея.
- Role – името на ролята на потребителя

Таблица Status

Таблицата изглежда по следния начин:

| Име на колона | Тип на данните | Позволяване на null стойности |
|---------------|----------------|-------------------------------|
| StatusId | Int | Не |
| Status | nvarchar(50) | Не |

Текущата таблица пази различните потребителски статуси. Те се използват в клиентското приложение, за да се определя дали потребителят има право да влиза в системата.

Има връзка едно към много с таблицата Member.

- StatusId - първичният ключ на таблицата представлява номера на запис в нея.
- Status – името на статуса на потребителя

Таблица Team

Таблицата изглежда по следния начин:

| Име на колона | Тип на данните | Позволяване на null стойности |
|---------------|----------------|-------------------------------|
| TeamId | Int | Не |
| Name | nvarchar(50) | Не |

Тук се пази имената на различните екипи, които ще бъдат показвани в клиентското приложение. Един екип може да има множество членове, затова има връзка много към много с таблицата Member. Освен това всеки екип разполага с множество агентски машини, като те могат да се споделят между повече от един екип, затова е добавена съединяващата таблица TeamAgentMachine. Има връзка много към много с таблицата Project, чрез TeamProject. TeamSuiteTest свързва определени тестове с тяхната листа и пази техният екип.

- TeamId - първичният ключ на таблицата представлява номера на запис в нея.
- Name – името на екипа

Таблица Project

Таблицата изглежда по следния начин

| Име на колона | Тип на данните | Позволяване на null стойности |
|---------------|----------------|-------------------------------|
| ProjectId | Int | Не |
| Name | nvarchar(50) | Не |
| TfsPath | nvarchar(MAX) | Не |
| TfsUrl | nvarchar(50) | Не |

Таблицата пази данни за проектите на даден екип, които ще бъдат използвани, за да изтегли последна версия на техния код. Всеки проект може да има повече от един екип, затова се използва обединяващата таблица TeamProject. Също така всички проекти могат да имат множество допълнителни подпроекти, които да трябва да се изтеглят преди да могат да бъдат билднати, затова имаме таблица много към много ProjectAdditionalPath.

- ProjectId - първичният ключ на таблицата представлява номера на запис в нея

- Name – името на проекта
- TfsPath – пътя на проекта в Team Foundation Server
- TfsUrl – URL към текущия TFS

Таблица AdditionalPath

| Име на колона | Тип на данните | Позволяване на null стойности |
|------------------|----------------|-------------------------------|
| AdditionalPathId | Int | Не |
| TfsPath | nvarchar(MAX) | Не |
| TfsUrl | nvarchar(50) | Не |

- AdditionalPathId - първичният ключ на таблицата, представлява номера на записа в нея
- TfsPath – пътя на проекта в Team Foundation Server
- TfsUrl – URL към текущия TFS

Таблица AgentMachine

| Име на колона | Тип на данните | Позволяване на null стойности |
|----------------|----------------|-------------------------------|
| AgentMachineId | Int | Не |
| Name | nvarchar(50) | Не |
| Ip | nvarchar(50) | Не |
| WorkingDirPath | nvarchar(MAX) | |

Тук се пазят данни за агент машините. Клиентското приложение използва данните от тази таблица, за да се свързва. Също така текущата работна папка спомага за генерирането на съобщенията изпращани към агентската програма.

- AgentMachineId - първичният ключ на таблицата, представлява номера на записа в нея
- Name – име на машината
- Ip – IP адрес
- WorkingDirPath – текущата работна директория на агентското приложение, която се задава по време на инсталация

Има връзка много към много с таблицата Team чрез TeamAgentMachine.

Таблица Test

Таблицата изглежда по следния начин

| Име на колона | Тип на данните | Позволяване на null стойности |
|---------------|----------------|-------------------------------|
| TestId | Int | Не |
| Name | nvarchar(MAX) | Не |

| | | |
|--------------|---------------|----|
| ClassName | nvarchar(MAX) | Да |
| AdditionDate | date | Не |
| DeletionDate | date | Не |
| ProjectId | int | Не |
| FullName | nvarchar(MAX) | Не |
| MethodId | nvarchar(50) | Не |

Това е една от най-важните таблици, в нея се пазят данните за всички налични по проекти тестове. Всеки тест има връзка с проекта, от който идва чрез ProjectId. Всеки TestResultRun има Foreign Key към разглежданата таблица- TestId. Използва се, за да може да се покаже допълнителна информация за теста, когато се показват готовите резултати. Също така има връзка между Suite, Test и Team в таблицата TeamSuiteTest, която вече беше разгледана. На база на датите за добавяне и изтриване, главната програма определя дали да дава право за избиране на съответния и тест и дали да го показва изобщо, тъй като той вече може да е премахнат от проекта.

- TestId - първичният ключ на таблицата, представлява номера на записа в нея
- Name – име на теста
- AdditionDate – дата на добавяне
- DeletionDate – дата на изтриване
- ProjectId – Foreign Key към таблицата Project
- FullName – пълно име, което включва namespace и име на класа
- MethodId – е специален guid, генериран от пълното име на теста

Таблица ExecutionResultRun

| Име на колона | Тип на данните | Позволяване на null стойности |
|----------------------|----------------|-------------------------------|
| ExecutionResultRunId | Int | Не |
| ComputerName | nvarchar(MAX) | Не |
| RanBy | Int | Не |
| Total | Int | Не |
| Executed | Int | Да |
| Passed | int | Да |
| Failed | Int | Да |
| NotExecuted | Int | Да |
| Start | datetime | Не |
| Finish | datetime | Не |

Агентското приложение прави записи в тази таблица, когато обработи файла с резултатите от тестовете. Първичният ключ се използва, за да се асоциира всеки изпълнен тест и неговият резултат към общото пускане. Когато агентът завърши парсването, последното съобщение, което изпраща до клиентското приложение, е ExecutionResultRunId, чрез което ResultDisplay User контролата може да покаже резултатите за тестовете асоциирани към този guid.

- ExecutionResultRunId - първичният ключ на таблицата представлява номера на записа в нея
- ComputerName – името на машината, на която са били изпълнени тестовете
- RanBy – ForeignKey към таблицата Member
- Total – общ брой на всички изпълнени тестове
- Executed – брой на изпълнените
- Passed – брой на преминалите успешно
- Failed – брой на преминалите неуспешно
- NotExecuted – брой на неизпълнените
- Start – дата и час на стартиране
- Finish – дата и час на приключване

Таблица ExecutionStatus

| Име на колона | Тип на данните | Позволяване на null стойности |
|-------------------|----------------|-------------------------------|
| ExecutionStatusId | Int | Не |
| Status | nvarchar(50) | Не |

- ExecutionStatusId - първичният ключ на таблицата представлява номера на записа в нея
- Status – името на статуса на теста

Таблица TestResultRun

| Име на колона | Тип на данните | Позволяване на null стойности |
|----------------------|----------------|-------------------------------|
| TestResultRunId | Int | Не |
| ExecutionStatusId | Int | Не |
| ExecutionLog | nvarchar(MAX) | Да |
| ExecutionResultRunId | nvarchar(50) | Не |
| TestId | Int | Не |
| Duration | Time(7) | Не |
| StartTime | datetime | Не |
| EndTime | datetime | Не |

- TestResultRunId - първичният ключ на таблицата представлява номера на записа в нея
- ExecutionStatusId – Foreign key към таблицата Execution status, определящ статуса на текущия тест

- ExecutionLog- записват се възникналите грешки при изпълнение
- ExecutionResultRunId - Foreign key към таблицата ExecutionResultRun, чрез който теста се асоциира към определено пускане
- TestId – id на изпълнения тест
- Duration – продължителност на изпълнение
- StartTime – час и дата на започване на изпълнението
- EndTime – час и дата на приключване на изпълнението

Текущата таблица пази данни за изпълнението на даден тест на определена дата.

Асоцииране се към общо пускане чрез ExecutionStatusId. Има връзка и към таблицата Test, за да се знае кой е изпълняваният тест, както и към ExecutionStatus, за да се определи статусът на теста.

Агентското приложение попълва данни в таблицата при обработване на резултатите.

Докато при показването им на потребителя, клиентското приложение чете от нея като филтрира тестовете по ExecutionStatusId. По този начин изкарва тестовете, асоциирани към специфично пускане.

Таблица Suite

Таблицата изглежда по следния начин

| Име на колона | Тип на данните | Позволяване на null стойности |
|---------------|----------------|-------------------------------|
| SuiteId | Int | Не |
| Name | nvarchar (50) | Не |
| Description | nvarchar(MAX) | Да |
| Active | bit | Не |
| AdditionDate | date | Не |
| DeletionDate | date | Не |

Използваме таблицата, за да пазим данни за създадените от даден потребител тестови групи. Те са видими за всички членове на даден екип. Приложението подсигурява, че няма да има групи с едни и същи тестове или имена в даден екип. Всяка листа може да има множество тестове ,затова използваме свързващата таблица TeamTestSuite. Всеки ред от SuiteResultRun, който пази информация за изпълнението на съответна група, има връзка с текущата таблица чрез колоната SuiteId.

- SuiteId - първичният ключ на таблицата представлява номера на записа в нея
- Name – име на листата от тестове
- Description – пояснителна бележка за останалите членове от екипа
- Active – ключ, който определя дали листата е активна
- AdditionDate – дата на добавяне
- DeletionDate – дата на изтриване

Таблица SuiteResultRun

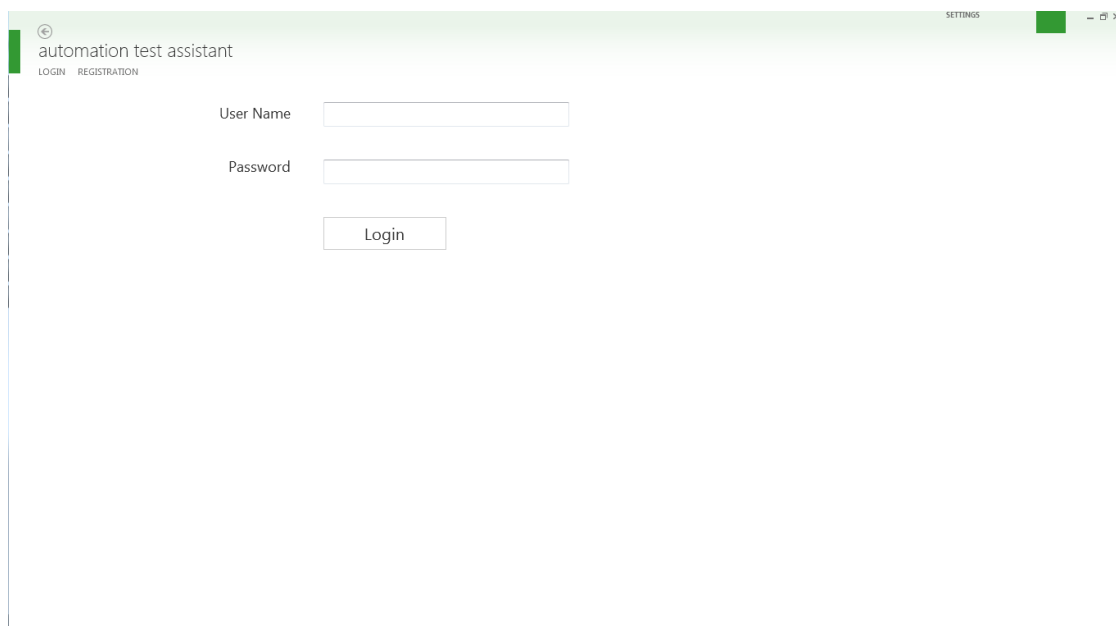
| Име на колона | Тип на данните | Позволяване на null стойности |
|-------------------|----------------|-------------------------------|
| SuiteResultRunId | Int | He |
| RanBy | int | He |
| Duration | time(7) | He |
| ExecutionStatusId | int | He |
| StartTime | datetime | He |
| EndTime | datetime | He |
| SuiteId | int | He |

- SuiteResultRunId - първичният ключ на таблицата представлява номера на записа в нея
- RanBy – Foreign key към таблицата Member, оказващ кой потребител е стартирал листата
- Duration – продължителност на изпълнение
- ExecutionStatusId - Foreign key към таблицата ExecutionStatusId, определящ статуса на групата след изпълнение
- StartTime – дата на стартиране
- EndTime – дата на приключване
- SuiteId – прави връзка с таблицата Suite

6. Ръководство на потребителя

Клиентско приложение

Вход в системата



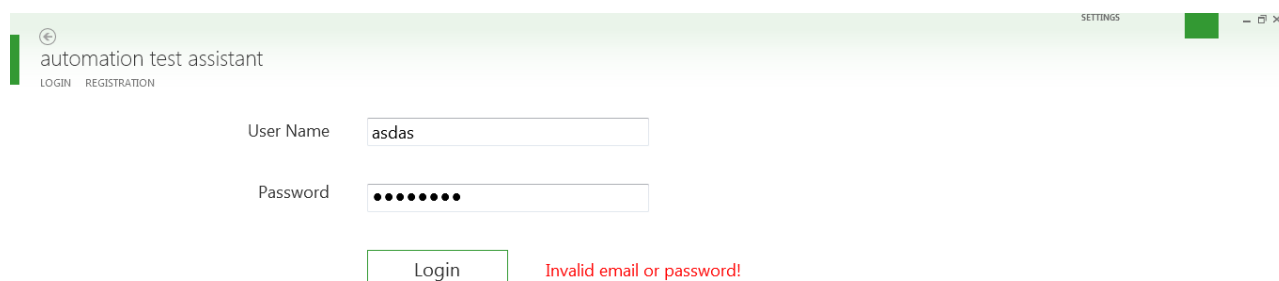
Фиг. 6.1 Форма за вход в клиентското приложение

Валидации във формата за вход

При стартиране на клиентското приложение, първата форма, която се зарежда е формата за вход. Ако потребителят има вече активиран акаунт, може да използва своето потребителско име и парола, за да влезе в системата. Ако обърка някои от данните, се изписва предупреждаващо съобщение (Фиг. 6.2.).



Фиг. 6.2. Required Validation на входящите данни

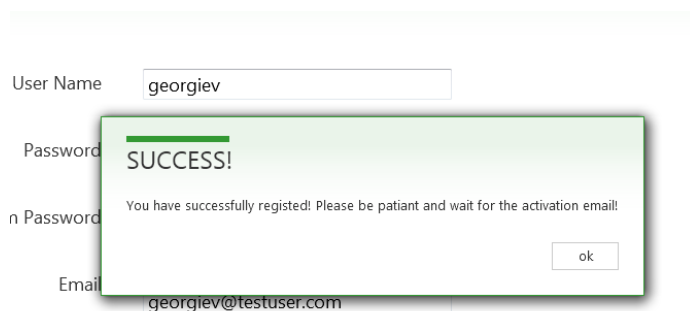


Фиг. 6.3. Validation за невалидна парола или потребителско име на входящите данни

Ако потребителят за първи път стартира програмата, може да натисне линка за регистрация и да си направи нов акаунт. Ако не се въведат задължителните полета, се изписва съобщение за валидация. Тъй като има твърде много информация, която трябва да се покаже на регистрационната форма, зареждат се всички екипи и роли от базата, се стартира нова нишка. В нея се зарежда нужната информация. През това време се показва progress bar, за да се уведоми потребителя, че нещо се случва на заден план.

Когато всички данни са валидни и се натисне бутона Register, се създава нов акаунт и се изписва съобщение на клиента, че процеса е преминал успешно.

Регистрационна форма



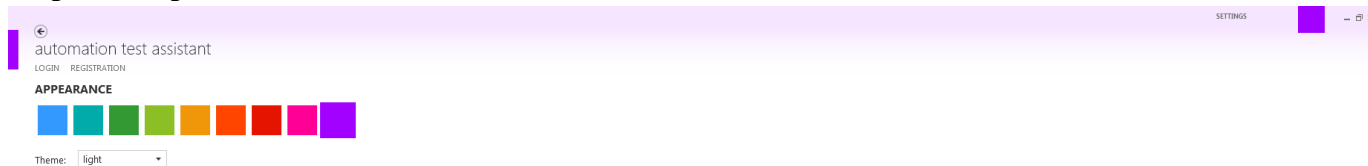
Фиг. 6.4. Съобщение за успешна регистрация

A screenshot of the 'automation test assistant' application window. The title bar shows 'automation test assistant' and 'SETTINGS'. Below the title bar, there are tabs for 'LOGIN' and 'REGISTRATION'. The 'REGISTRATION' tab is active. The form contains the following fields: 'User Name' (with placeholder 'newUserName'), 'Password' (masked with dots), 'Confirm Password' (masked with dots), 'Email' (with placeholder 'newUserName@yahoo.com'), 'Tfs User Name' (with placeholder 'newUserNameTfsUser'), 'Role' (a dropdown menu with 'Admin' selected), 'Team' (a group of checkboxes: 'SIT' is checked, 'CRM', 'WCAT', and 'PLATFORM' are unchecked), and 'Comment' (with placeholder text 'Please approve my request!'). At the bottom of the form is a 'Register' button.

Фиг. 6.5. Форма за регистрация

Всеки потребител има възможност да променя външния вид на приложението, когато зареди Settings формата, където може да се сменя цвета и темата. При смяна текущите настройки се запазват под папката с името на текущия потребител в регистрите. По този начин следващия път, когато влезе в системата, ще му бъде зареден същия външен вид. На един и същи компютър приложението може да се използва от различни хора и за всеки един от тях ще се зарежда индивидуална тема и цвят.

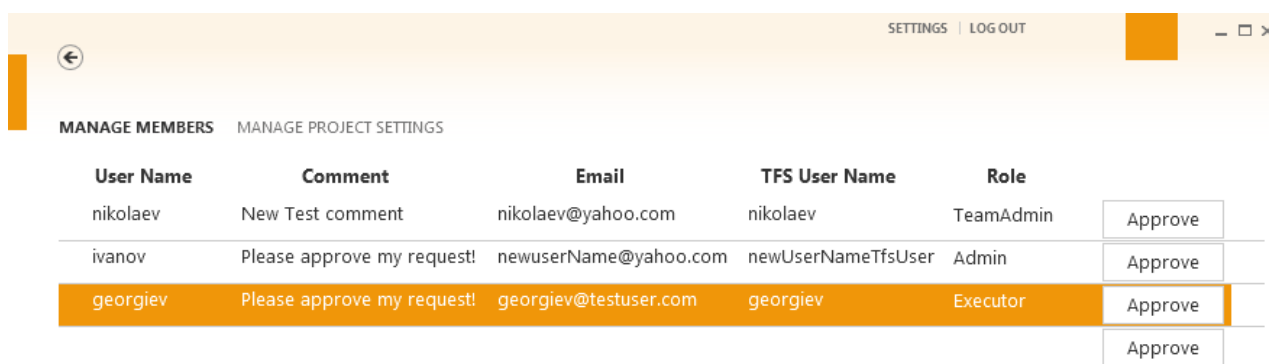
Форма за промяна на външния вид



Фиг. 6.6. Форма за променяне на външния вид

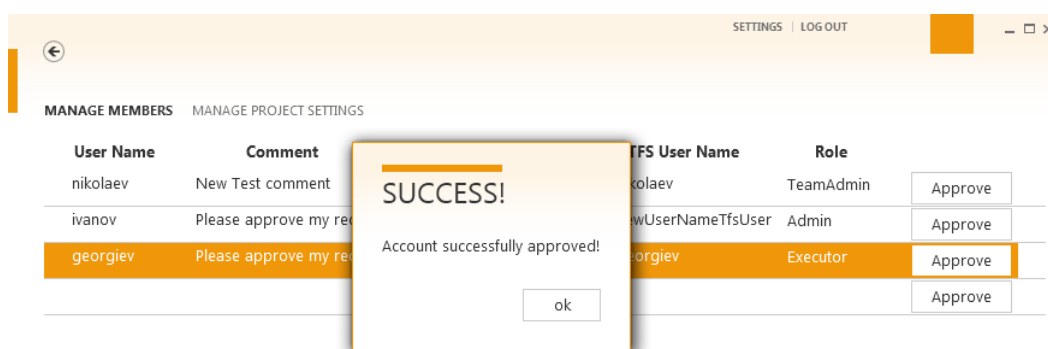
След като даден потребител влезе в програмата, в зависимост от ролята му се показват различни форми. Ако е админ, ще може да одобрява ново-регистриралите се потребители (Фиг. 6.7.) и да добавя нови екипи, проекти и агентски машини.

Форма за активиране на потребители



Фиг. 6.7. Форма за одобряване на ново-регистрирали се потребители

Админът има право да променя настройките на новите потребители, например да сменя ролята им, ако прецени, че тя е некоректна. При натискане на бутона Approve клиентът е одобрен и му се изпраща имейл с активационен код. Тогава записът се премахва от таблицата и остават само неодобрените.



Фиг. 6.8. Съобщение за успешно одобряване на потребител

Форма за активиране на акаунт

automation test assistant
LOGIN REGISTRATION

Activation Code

Activate

Фиг. 6.9. Форма за активация на акаунт

При опит за вход на потребител в статус PendingActivation, ще му се покаже горната форма, където ако въведе кода, който му е изпратен на имейла, акаунта му ще бъде активиран.

Форма за менажиране на настройките

Когато админът натисне линка „Manage Project Settings“, му се отваря нова форма, където се показват всички екипи, техните проекти и машини. Има възможност да се добавят **НОВИ**.

MANAGE MEMBERS MANAGE PROJECT SETTINGS

New Team New Project New Additional Path New Machine

SIT

TimeTest

ID: 1

Name: TimeTest

TFS Path: ~/CorporateSites/Main

TFS URL: sampleTfsUrl

AdditionalPaths

| ID | TFS Path | TFS Url |
|----|--------------|--|
| 1 | ~/FirstAddP1 | http://aangelov-pc:8080/tfs/DefaultCollection/ |
| 2 | ~/ScondAddP2 | http://aangelov-pc:8080/tfs/DefaultCollection/ |

TestAutomationAssistant

ID: 2

Name: TestAutomationAssistant

TFS Path: ~/CoporateSites/Test

TFS URL: sampleUrl2

AdditionalPaths

| ID | TFS Path | TFS Url |
|----|--------------|--|
| 1 | ~/FirstAddP1 | http://aangelov-pc:8080/tfs/DefaultCollection/ |
| 2 | ~/ScondAddP2 | http://aangelov-pc:8080/tfs/DefaultCollection/ |

CRM

| Id | Name | Ip |
|----|-------------|---------------|
| 4 | GEORGIEV-PC | 192.168.1.125 |
| 8 | AANGELOV-PC | 192.168.1.321 |

WCAT

PLATFORM

Admin

WPF_Team1

ASP.NET2

ASP.NET1

AANGELOV

| Id | Name | Ip |
|----|--------------------|---------------|
| 19 | Windows7FirstAgent | 192.168.1.120 |

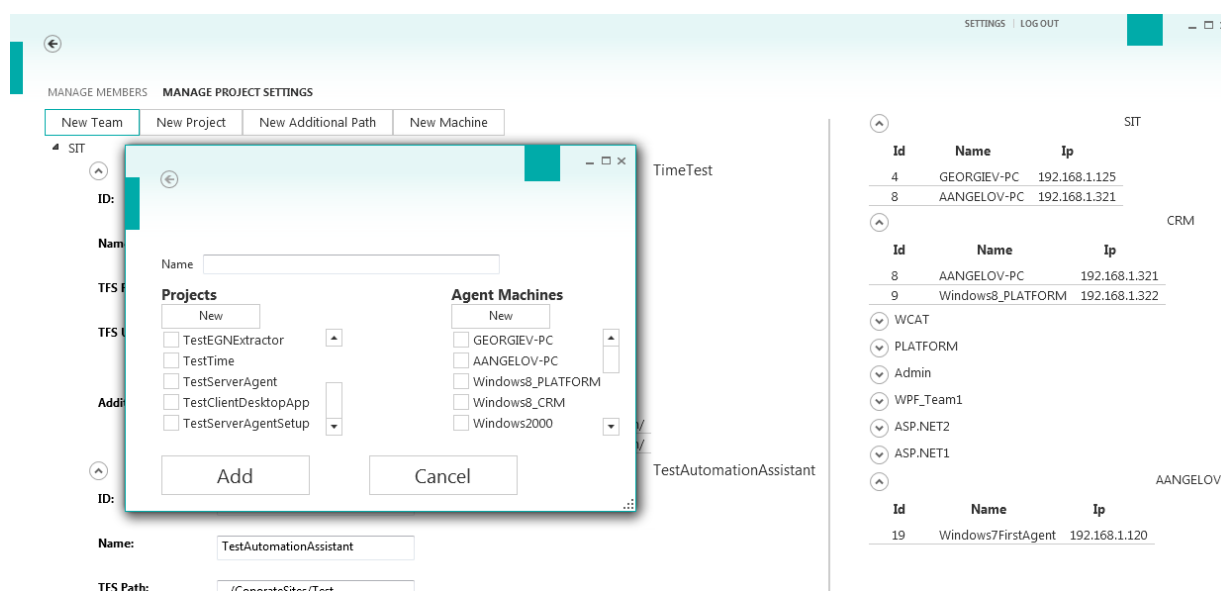
Фиг. 6.10. Форма за показване на екипи и техните проекти, машини

В дървовидна структура се виждат всички екипи. Потребителят може да затвори тези, които не го интересуват. Също така, тъй като всеки проект има твърде много данни, те са разделени на две части - главна информация и допълнителни пътища, показвайки се в

отделна вградена таблица, която може отново да се скрива. Всички полета могат да се редактират. От дясната страна са показани всички екипи и техните агентски машини, отново подредени в дървовидна структура, за да може да се скриват ненужните редове.

При натискане на бутона NewTeam се отваря нов прозорец, в който се зарежда формата за добавяне на екипи.

Форма за добавяне на нов екип

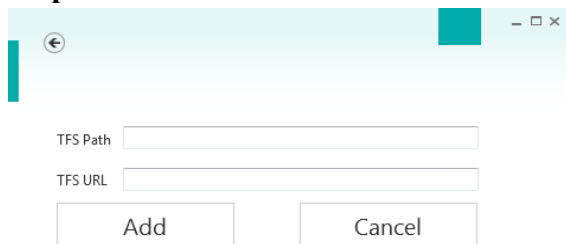


Фиг. 6.11. Форма за добавяне на екипи

От тази форма може да се добавят нови проекти и агентски машини. При натискане на бутона New, новата логика ще се зареди в същия прозорец. Може да се върнем или чрез бутона стрелка назад или ако натиснем Cancel. Когато се зареди формата за добавяне на проекти или агентски машини, в зависимост дали сме я заредили от главната или от екипната страница, при натискане на бутона Cancel ще бъдем върнати на формата източник.

За да се въведе нов екип, трябва да се добави име и да се асоциират поне един проект и агентска машина към него, за да може да се използва.

Форма за добавяне на нов допълнителен път



TFS Path

TFS URL

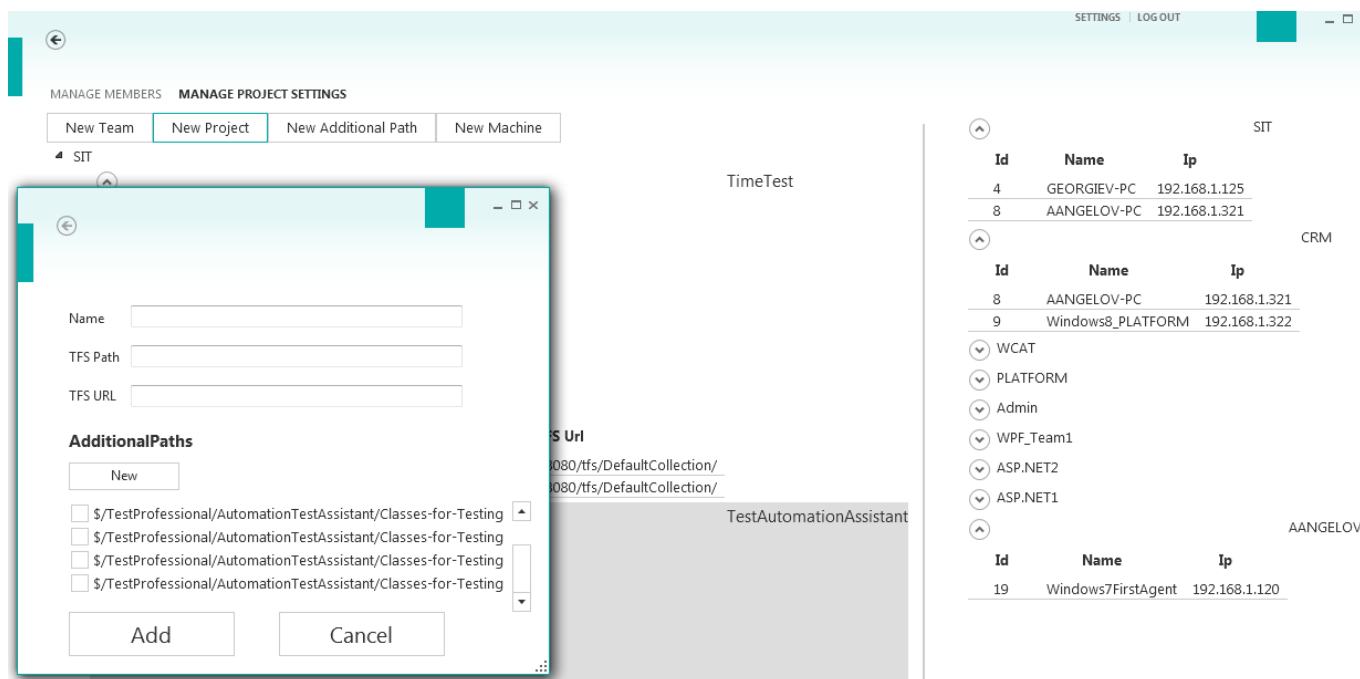
Add Cancel

Фиг. 6.12. Форма за добавяне на допълнителни пътища

Новодобавените проекти и машини ще се покажат, когато при връщане на тази страница.

Форма за добавяне на нов проект

За да се добави нов проект, трябва да се натисне бутона New Project на страницата Manage Project Settings или аналогичния бутон от формата за нов екип. Нужно е да въведен име, път в Team Foundation Server и TFS URL - това са задължителните части. По избор може да се асоциират допълнителни пътища, които да се теглят при изпълнение на проекта. Ако желания път не се съдържа в базата, може да се добави с натискане на бутона New.



MANAGE MEMBERS MANAGE PROJECT SETTINGS

New Team New Project New Additional Path New Machine

▲ SIT

TimeTest

TestAutomationAssistant

S Url

080/tfs/DefaultCollection/

080/tfs/DefaultCollection/

Id Name Ip

| | | |
|---|-------------|---------------|
| 4 | GEORGIEV-PC | 192.168.1.125 |
| 8 | AANGELOV-PC | 192.168.1.321 |

CRM

Id Name Ip

| | | |
|---|-------------------|---------------|
| 8 | AANGELOV-PC | 192.168.1.321 |
| 9 | Windows8_PLATFORM | 192.168.1.322 |

WCAT

PLATFORM

Admin

WPF_Team1

ASP.NET2

ASP.NET1

AANGELOV

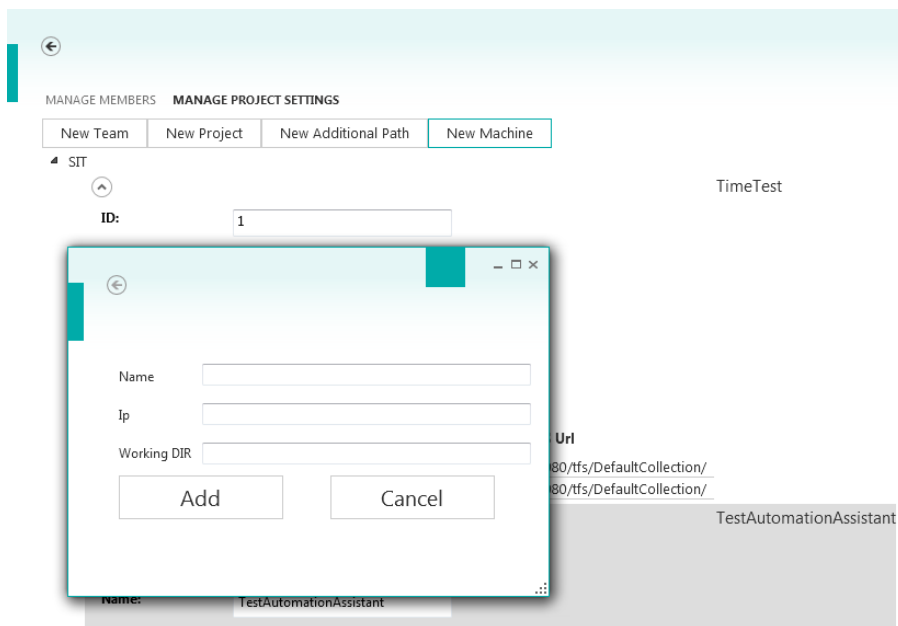
Id Name Ip

| | | |
|----|--------------------|---------------|
| 19 | Windows7FirstAgent | 192.168.1.120 |
|----|--------------------|---------------|

Фиг. 6.13. Форма за добавяне на проекти

Форма за добавяне на нова агентска машина

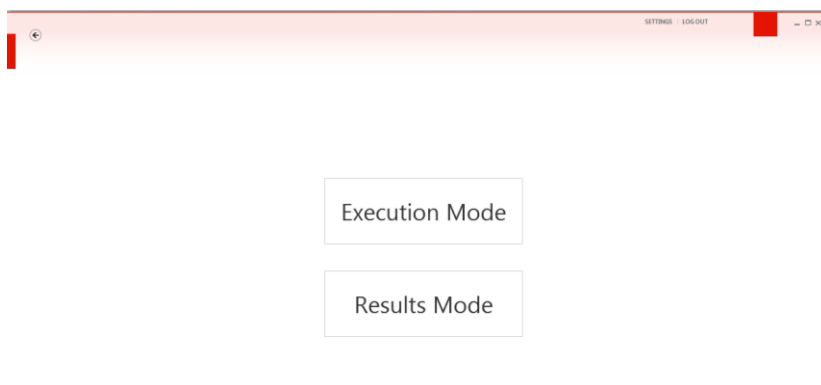
Може да се добави машина по аналогичен начин като се натисне бутона New Machine.



Фиг. 6.14. Форма за добавяне на агентски машини

Когато влезият потребител не е админ, а неговата роля е *Executor*, има право да изпълнява тестове и да вижда резултатите от предходни пускания. Поради тази причина веднага след формата за вход, се зареждат два бутона, които определят в кой от двата *mode* иска текущият клиент.

Форма за избиране на Mode



Фиг. 6.15. Форма за избиране на mode

Форма за избиране на проекти

Ако потребителят избере първия бутон, ще му се зареди нова страница, където ще види всички проекти, асоциирани към екипите, на които е член.

SETTINGS LOG OUT

SIT

AANGLOV

TestEncryption

TestEGNExtractor

ID: 6

Name: TestEGNExtractor

TFS Path: \$/TestProfessional/AutomationTestAssistant/TestEGNExtractor

TFS URL: http://aangelov-pc8080/tfs/DefaultCollection/

Local Path: E:\TestProfessional\AutomationTestAssistant\TestEGNExtractor

AdditionalPaths

TestTime

ID: 7

Name: TestTime

TFS Path: \$/TestProfessional/AutomationTestAssistant/TestTime

TFS URL: http://aangelov-pc8080/tfs/DefaultCollection/

Local Path: D:\TestProfessional\AutomationTestAssistant\TestTime

| ID | TFS Path | TFS Uri | Local Path |
|----|---|---|---|
| 4 | \$/TestProfessional/AutomationTestAssistant/Classes-for-Testing | http://aangelov-pc8080/tfs/DefaultCollection/ | D:\TestProfessional\AutomationTestAssistant\Classes-for-Testing |

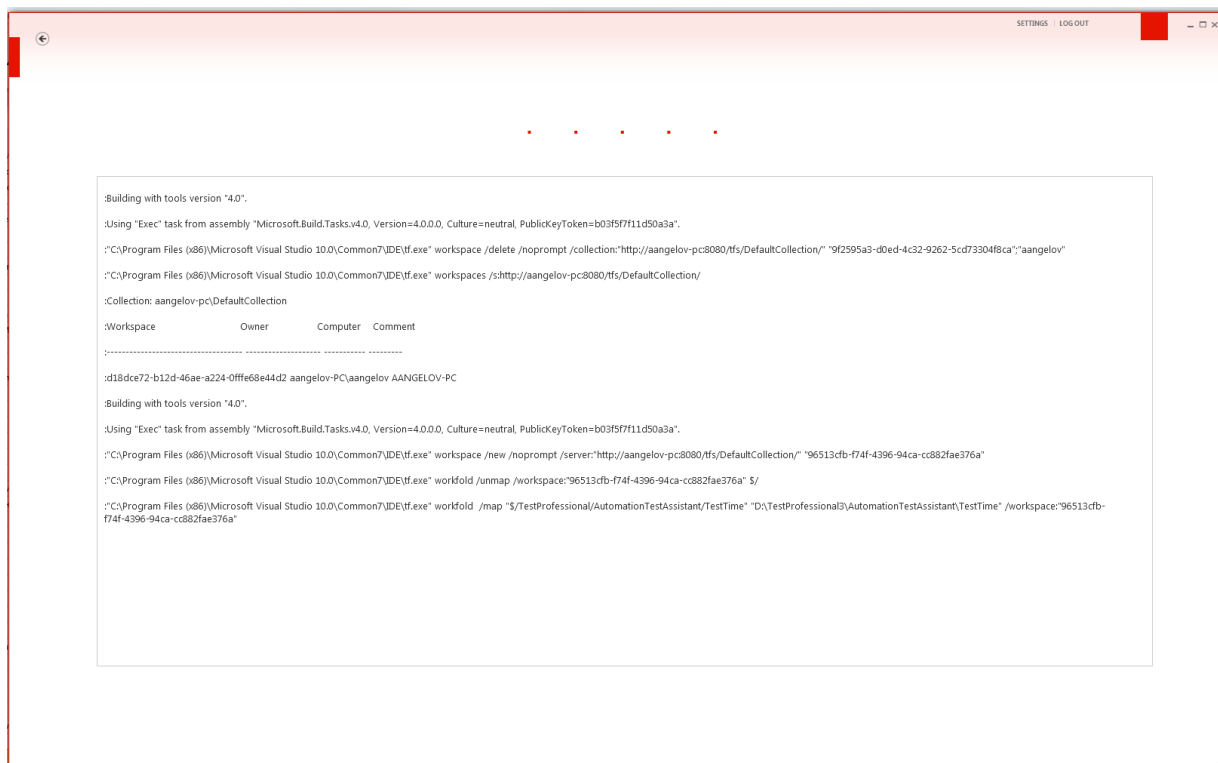
Load

Фиг. 6.16. Форма за избиране на проекти

Тук клиентът трябва да избере проектите, чиито тестове иска да зареди. За целта трябва да въведе локални пътища, където ще свали последната версия на проектите. Тези пътища се записват в регистрите на Windows под папката на съответния потребител. По този начин, ако друг човек влезе през същата машина, той ще може да настрои различни пътища без да пречи на другите потребители. Когато потребител едно влезе отново в системата, ще му бъдат заредени пътищата, които е сложил при предишното зареждане на програмата.

Форма за показване на текущия статус на зареждане на тестовите

Отново таблиците се показват в дървовиден вид с възможност за скриване на информацията, която представлява интерес. При натискане на бутона Load, се появява нова контрола, която показва прогреса на зареждане на тестовите и текущо изпълняваните команди като: създаване на нов TFS Workspace, Delete TFS Workspace, вземане на последна версия на проекта, билдване.

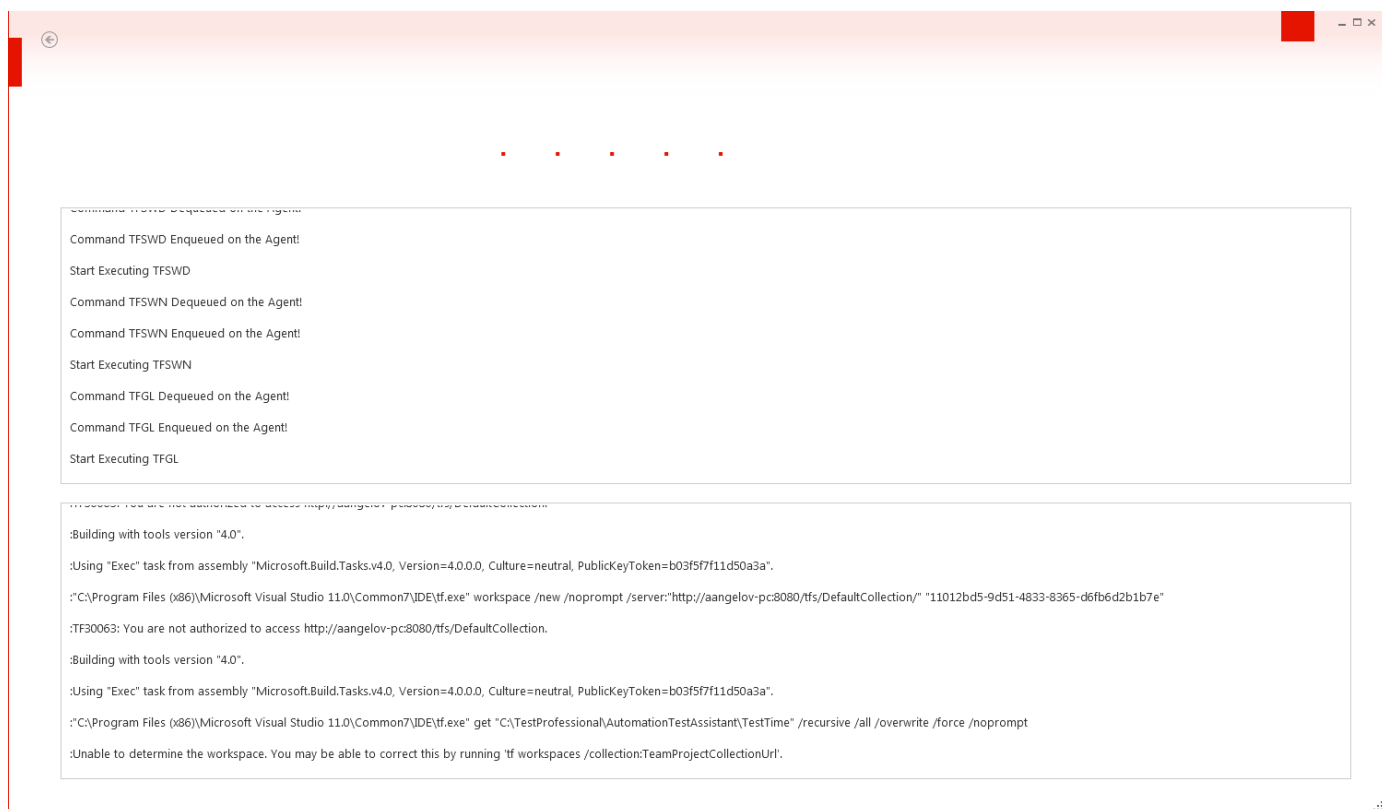


Фиг. 6.17. Форма за показване на текущия статус на зареждане на тестовите

След като премине процесът по зареждане, се вижда нова форма, където се подреждат тестовите по екипи и проекти. От лявата страна може да се изберат тестовите, които ни интересуват. По подразбиране всички тестове са избрани. Преди да ги изпълним, трябва да се избере агентска машина. Може да се избере само една.

Форма за показване на текущия статус на зареждане на командите на агентска машина

При натискане на бутона Execute, ще се отвори нов прозорец за зареждане, където ще се вижда прогреса на изпълняваните команди на другата машина в реално време. При избиране на друга машина, ще се отвори втори такъв прозорец. Може едновременно да се следи прогреса на множество агенти.



Фиг. 6.18. Форма за показване на текущия статус на зареждане на командите на агентска машина

След като тестовите бъдат изпълнени на агента, той връща съобщение с номера на текущото пускане. Чрез него се зарежда нова форма с текущите резултати.

| TEST RESULTS | | | | | |
|---|---------------|---------------|---|-----------------|--|
| Total: 32 Executed: 32 Passed: 32 NotExecuted: 0 Failed: 0 | | | | | |
| Start: 06/02/13 14:49:17 Finish: 06/02/13 14:49:17 RanBy: master Computer Name: TEST-PC | | | | | |
| StartTime | EndTime | Duration | TestName | ExecutionStatus | |
| 02:49:17.1530 | 02:49:17.4330 | 00:00:00.0223 | TestIncrementNormalValuesSufidIncrement | Passed | |
| 02:49:17.4370 | 02:49:17.4400 | 00:00:00.0004 | TestSubtractionDecreaseHours | Passed | |
| 02:49:17.4430 | 02:49:17.4470 | 00:00:00.0003 | TestAdditionNormalValues | Passed | |
| 02:49:17.4530 | 02:49:17.4570 | 00:00:00.0002 | TestSubtractionWithMoreThan60Minutes | Passed | |
| 02:49:17.4600 | 02:49:17.4600 | 00:00:00.0002 | TestAdditionWithNegativeAmountOfMinutes | Passed | |
| 02:49:17.4670 | 02:49:17.4700 | 00:00:00.0002 | TestSubtractionNormalValues | Passed | |
| 02:49:17.4730 | 02:49:17.4770 | 00:00:00.0010 | TestEqualsObjectEqualTimes | Passed | |
| 02:49:17.4800 | 02:49:17.4830 | 00:00:00.0004 | TestAdditionIncreaseHour | Passed | |
| 02:49:17.4830 | 02:49:17.4900 | 00:00:00.0003 | TestEqualsObjectNotEqualTimes | Passed | |
| 02:49:17.4900 | 02:49:17.4970 | 00:00:00.0034 | TestTimeConstructorWithTooBigHour | Passed | |
| 02:49:17.5000 | 02:49:17.5030 | 00:00:00.0006 | TestIncrementWith59Minutes | Passed | |
| 02:49:17.5030 | 02:49:17.5070 | 00:00:00.0002 | TestEqualsNotEqualTimes | Passed | |
| 02:49:17.5100 | 02:49:17.5130 | 00:00:00.0002 | TestSubtractionWith0Mintes | Passed | |
| 02:49:17.5170 | 02:49:17.5200 | 00:00:00.0003 | TestSubtractionWhen0Hour10MinutesMinus20Minutes | Passed | |
| 02:49:17.5200 | 02:49:17.5270 | 00:00:00.0014 | TestTimeConstructorWithNegativeAmountMinutes | Passed | |
| 02:49:17.5270 | 02:49:17.5330 | 00:00:00.0003 | TestSubtractionWithNegativeAmountOfMinutes | Passed | |
| 02:49:17.5330 | 02:49:17.5400 | 00:00:00.0006 | TestGetHashCodeDifferentObjects | Passed | |
| 02:49:17.5400 | 02:49:17.5430 | 00:00:00.0002 | TestEqualNull | Passed | |
| 02:49:17.5470 | 02:49:17.5500 | 00:00:00.0002 | TestIncrementNormalValuesPrefixIncrement | Passed | |
| 02:49:17.5500 | 02:49:17.5570 | 00:00:00.0012 | TestTimeConstructorWithTooBigAmountMinutes | Passed | |
| 02:49:17.5600 | 02:49:17.5630 | 00:00:00.0012 | TestTimeConstructorAreMinutesCorrect | Passed | |
| 02:49:17.5670 | 02:49:17.5670 | 00:00:00.0003 | TestDecrementNormalValuesPrefixIncrement | Passed | |
| 02:49:17.5700 | 02:49:17.5730 | 00:00:00.0003 | TestEqualsEqualTimes | Passed | |
| 02:49:17.5770 | 02:49:17.5800 | 00:00:00.0004 | TestDecrementNormalValuesSufidIncrement | Passed | |
| 02:49:17.5800 | 02:49:17.5870 | 00:00:00.0004 | TestAdditionWithMoreThan60Minutes | Passed | |
| 02:49:17.5900 | 02:49:17.5970 | 00:00:00.0015 | TestToString | Passed | |
| 02:49:17.5970 | 02:49:17.6000 | 00:00:00.0003 | TestEqualObjectNull | Passed | |
| 02:49:17.6030 | 02:49:17.6070 | 00:00:00.0003 | TestDecrementWith59Minutes | Passed | |
| 02:49:17.6100 | 02:49:17.6130 | 00:00:00.0002 | TestAdditionWith0Mintes | Passed | |
| 02:49:17.6130 | 02:49:17.6170 | 00:00:00.0002 | TestGetHashCodeEqualObjects | Passed | |
| 02:49:17.6200 | 02:49:17.6200 | 00:00:00.0002 | TestTimeConstructorAreHoursCorrect | Passed | |
| 02:49:17.6230 | 02:49:17.6270 | 00:00:00.0002 | TestAdditionWhen23Hour59MinutesPlus10Minutes | Passed | |

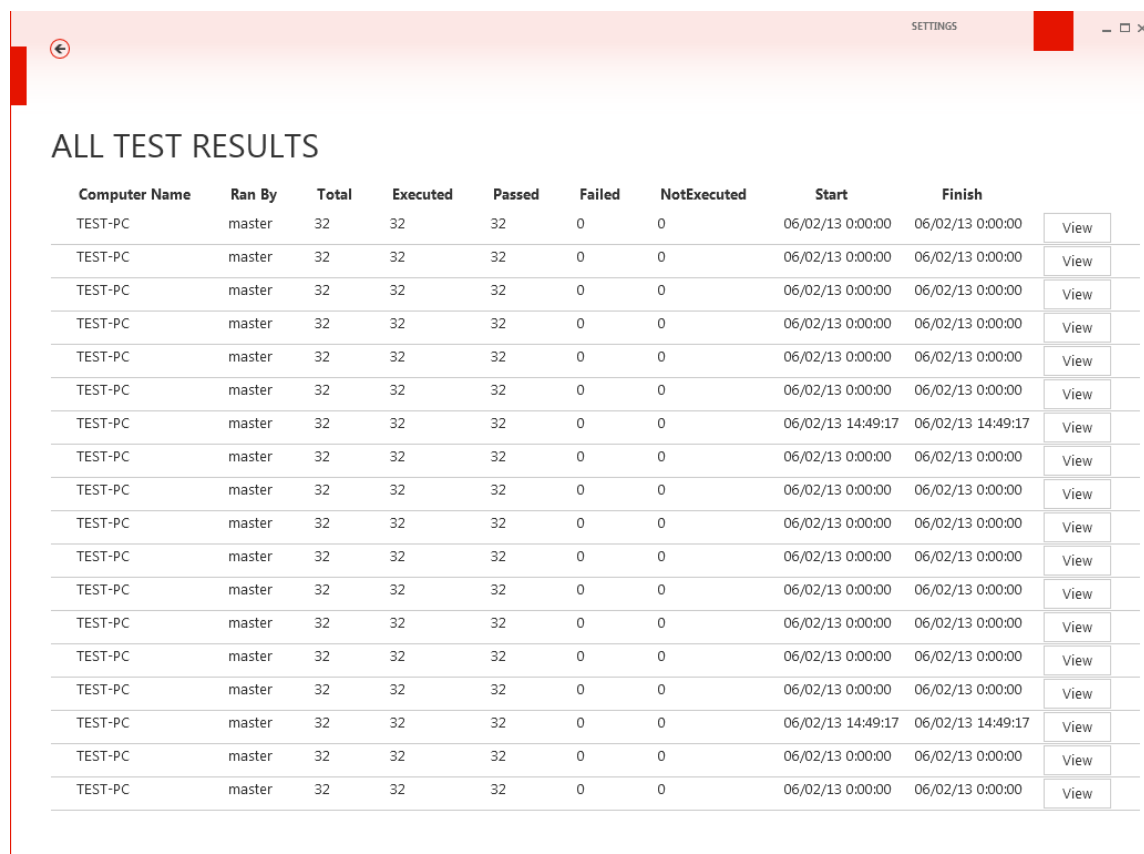
Фиг. 6.19. Форма показваща резултатите от изпълнените тестове

Форма показваща резултатите от изпълнените тестове

След като изпълнението приключи, може отново да използва агентската машина, за да се изпълнят други тестове на нея. При стартиране старите проекти ще бъдат изтрети и ще се изтеглят наново от Team Foundation Server, за да е сигурно, че ще са последна версия.

Форма показваща всички пускания на тестове за определени екипи

Ако след влизане в системата се избере Result Mode, ще се покаже форма с всички пускания на тестове от членове на екипите, в които участваме.



| Computer Name | Ran By | Total | Executed | Passed | Failed | NotExecuted | Start | Finish | |
|---------------|--------|-------|----------|--------|--------|-------------|-------------------|-------------------|------|
| TEST-PC | master | 32 | 32 | 32 | 0 | 0 | 06/02/13 0:00:00 | 06/02/13 0:00:00 | View |
| TEST-PC | master | 32 | 32 | 32 | 0 | 0 | 06/02/13 0:00:00 | 06/02/13 0:00:00 | View |
| TEST-PC | master | 32 | 32 | 32 | 0 | 0 | 06/02/13 0:00:00 | 06/02/13 0:00:00 | View |
| TEST-PC | master | 32 | 32 | 32 | 0 | 0 | 06/02/13 0:00:00 | 06/02/13 0:00:00 | View |
| TEST-PC | master | 32 | 32 | 32 | 0 | 0 | 06/02/13 0:00:00 | 06/02/13 0:00:00 | View |
| TEST-PC | master | 32 | 32 | 32 | 0 | 0 | 06/02/13 0:00:00 | 06/02/13 0:00:00 | View |
| TEST-PC | master | 32 | 32 | 32 | 0 | 0 | 06/02/13 14:49:17 | 06/02/13 14:49:17 | View |
| TEST-PC | master | 32 | 32 | 32 | 0 | 0 | 06/02/13 0:00:00 | 06/02/13 0:00:00 | View |
| TEST-PC | master | 32 | 32 | 32 | 0 | 0 | 06/02/13 0:00:00 | 06/02/13 0:00:00 | View |
| TEST-PC | master | 32 | 32 | 32 | 0 | 0 | 06/02/13 0:00:00 | 06/02/13 0:00:00 | View |
| TEST-PC | master | 32 | 32 | 32 | 0 | 0 | 06/02/13 0:00:00 | 06/02/13 0:00:00 | View |
| TEST-PC | master | 32 | 32 | 32 | 0 | 0 | 06/02/13 0:00:00 | 06/02/13 0:00:00 | View |
| TEST-PC | master | 32 | 32 | 32 | 0 | 0 | 06/02/13 0:00:00 | 06/02/13 0:00:00 | View |
| TEST-PC | master | 32 | 32 | 32 | 0 | 0 | 06/02/13 0:00:00 | 06/02/13 0:00:00 | View |
| TEST-PC | master | 32 | 32 | 32 | 0 | 0 | 06/02/13 14:49:17 | 06/02/13 14:49:17 | View |
| TEST-PC | master | 32 | 32 | 32 | 0 | 0 | 06/02/13 0:00:00 | 06/02/13 0:00:00 | View |
| TEST-PC | master | 32 | 32 | 32 | 0 | 0 | 06/02/13 0:00:00 | 06/02/13 0:00:00 | View |

Фиг. 6.20. Форма показваща всички пускания на тестове за определени екипи

Може да види цялата информация за дадено пускане като: кой е стартирал процеса, на коя машина, общ брой на минали успешно и неизпълнени се тестове, кога са били изпълнени и т.н. При натискане на бутона View ще се отвори подробна информация тест по тест за съответния ред.

7. Приложения

Приложение 1- клас ServerAgent

```
public class ServerAgent : BaseLogger
{
    private const string AgentSettingsInitializedMsg = "Agent Settings Initialized";
    private const string ExecutionTaskStartedMsg = "Execution Task Successfully
started";
    private const string MessageLoggerTaskStartedMsg = "Message Logger Task
Successfully started";
    private const string MessageLoggerTcpClientConnectedMsg = "Message Logger Task Tcp
Client Started";
    private const string ClientMessageProcessorTcpClientConnectedMsg = "Client Message
Processor Tcp Client Started";
    private const string ClientMessageProcessorTaskStartedMsg = "Client Message
Processor Task Successfully started";
    private const string ClientMessageMsBuildListnerTaskStartedMsg = "MsBuild Listner
task Successfully started";
    private static Process currentlyExecutedProc;
    private static Task executeCommandThreadWorker;
    private static Task msBuildLogListnerThreadWorker;
    private static Task messageLoggerThreadWorker;
    private static Task clientProcessMessageThreadWorker;
    private static Object lockObject;
    private static IPAddressSettings clientSettings;
    private static IPAddressSettings agentSettings;
    private static IPAddressSettings msBuildLogSettings;
    private static TcpListener agentListener;
    private static ConcurrentQueue<MessageArgsLogger> messagesToBeSend;
    private static ConcurrentQueue<string> commandsToBeExecuted;
    private static CancellationToken token;
    private static CancellationTokenSource cts;

    static void Main(string[] args)
    {
        while (true)
        {
            try
            {
                InitializeSettings();
                clientProcessMessageThreadWorker = Task.Factory.StartNew(() =>
ProcessClientMessage(agentListener), token, TaskCreationOptions.LongRunning,
TaskScheduler.Current);
                Console.WriteLine(ClientMessageProcessorTaskStartedMsg);
                BaseLogger.Log.Info(ClientMessageProcessorTaskStartedMsg);
                messageLoggerThreadWorker = Task.Factory.StartNew(() =>
SendLogMessages(agentListener, messagesToBeSend), token, TaskCreationOptions.LongRunning,
TaskScheduler.Current);
                Console.WriteLine(MessageLoggerTaskStartedMsg);
                BaseLogger.Log.Info(MessageLoggerTaskStartedMsg);
                executeCommandThreadWorker = Task.Factory.StartNew(() =>
ExecuteCommands(), token, TaskCreationOptions.LongRunning, TaskScheduler.Current);
                Console.WriteLine(ExecutionTaskStartedMsg);
                BaseLogger.Log.Info(ExecutionTaskStartedMsg);
                msBuildLogListnerThreadWorker = Task.Factory.StartNew(() =>
ATACore.TcpWrapperProcessor.TcpMsBuildLoggerProcessor.ProcessMsBuildLoggerMessage(messages
ToBeSend, msBuildLogSettings), TaskCreationOptions.LongRunning);
                Console.WriteLine(ClientMessageMsBuildListnerTaskStartedMsg);
                BaseLogger.Log.Info(ClientMessageMsBuildListnerTaskStartedMsg);
                Task.WaitAll(new Task[] { clientProcessMessageThreadWorker,
executeCommandThreadWorker, messageLoggerThreadWorker, msBuildLogListnerThreadWorker });
            }
            catch { }
        }
    }
}
```

```

    }

    catch (AggregateException ae)
    {
        ae = ae.Flatten();
        ae.InnerExceptions.ToList().ForEach(ex =>
BaseLogger.Log.Error(ex.Message));
    }
    catch (TaskCanceledException ce)
    {
        BaseLogger.Log.InfoFormat("Agent main/sub task(s) canceled: {0}",
ce.InnerException);
    }
    finally
    {
        agentListener.Stop();
        CancelCurrentlyExecutingProcess();
    }
}

private static void InitializeSettings()
{
    clientSettings = new
IpAddressSettings(ConfigurationManager.AppSettings["clientIp"],
ConfigurationManager.AppSettings["clientPort"]);
    agentSettings = new
IpAddressSettings(ConfigurationManager.AppSettings["agentIp"],
ConfigurationManager.AppSettings["agentPort"]);
    msBuildLogSettings = new
IpAddressSettings(ConfigurationManager.AppSettings["agentIp"],
ConfigurationManager.AppSettings["msBuildAgentPort"]);
    Console.BackgroundColor = ConsoleColor.DarkGreen;
    commandsToBeExecuted = new ConcurrentQueue<string>();
    messagesToBeSend = new ConcurrentQueue<MessageArgsLogger>();
    cts = new CancellationTokenSource();
    token = cts.Token;
    currentlyExecutedProc = null;
    lockObject = new Object();
    agentListener = new TcpListener(agentSettings.GetIPAddress(),
clientSettings.Port);
    agentListener.Start();
    BaseLogger.Log.Info(AgentSettingsInitializedMsg);
    Console.WriteLine(AgentSettingsInitializedMsg);
}

private static void SendLogMessages(TcpListener agentListener,
ConcurrentQueue<MessageArgsLogger> messagesToBeSend)
{
    TcpClient agentTcpWriter = default(TcpClient);
    lock (lockObject)
        agentTcpWriter = agentListener.AcceptTcpClient();
    BaseLogger.Log.Info(MessageLoggerTcpClientConnectedMsg);
    Console.WriteLine(MessageLoggerTcpClientConnectedMsg);
    SendLogMessagesInternal(messagesToBeSend, agentTcpWriter);
}

```



```

private static void SendLogMessagesInternal(ConcurrentQueue<MessageArgsLogger>
messagesToBeSend, TcpClient agentTcpWriter)
{
    while (true)
    {
        SendLogMessagesProcessCancellation(agentTcpWriter);
        if (messagesToBeSend.Count > 0)
        {
            MessageArgsLogger msgArgs;
            bool isDequeued = messagesToBeSend.TryDequeue(out msgArgs);
            if (isDequeued)
            {
                string messageToBeSend =
ATACore.CommandExecutor.GenerateCurrentCommandParametersXml(msgArgs);
                BaseLogger.Log.InfoFormat("\n\nMessage To BE Send:\n{0}\n\n",
messageToBeSend);

ATACore.TcpWrapperProcessor.TcpClientWrapper.SendMessageToClient(agentTcpWriter,
messageToBeSend);
            }
            Thread.Sleep(50);
        }
    }

private static void SendLogMessagesProcessCancellation(TcpClient agentTcpWriter)
{
    if (token.IsCancellationRequested)
    {
        agentTcpWriter.Close();
        token.ThrowIfCancellationRequested();
    }
}

private static void ExecuteCommands()
{
    while (true)
    {
        if (commandsToBeExecuted.Count > 0 && (currentlyExecutedProc == null ||
currentlyExecutedProc.HasExited))
        {
            string currentCommandXml = String.Empty;
            bool dequeueSuccessfull = commandsToBeExecuted.TryDequeue(out
currentCommandXml);
            if (!dequeueSuccessfull)
                continue;
            Command currentAgentCommand =
ATACore.CommandExecutor.GetCommandToBeExecutedFromMessage(currentCommandXml);
            string dequeuedMsg = String.Format("Command {0} Dequeued on the
Agent!", currentAgentCommand);
            ATACore.CommandExecutor.EnqueueNewMessage(dequeuedMsg,
MessageSource.DenqueueLogger, messagesToBeSend);
            // Waits until the both threads are synchronized. The backgroudworker2
should be initialized again
            //InitializeMsBuildLogger();
            if (currentAgentCommand.Equals(Command.PARSE))
            {

```

```

XmlSerializer deserializer = deserializer = new
XmlSerializer(typeof(MessageArgsParseResult));
    StreamReader textReader = new StreamReader(currentCommandXml);
    MessageArgsParseResult msgArgParseResult =
(MessageArgsParseResult)deserializer.Deserialize(textReader);
    Member currentMember =
ATACore.Managers.MemberManager.GetMemberByUserName(ATACore.Managers.ContextManager.Context
, msgArgParseResult.UserName);
    ATACore.Managers.ContextManager.Dispose();
    string executionResultRunId =
ATACore.TestExecution.TestListParser.GetExecutionResultRunFromResultFile(msgArgParseResult
.ResultsFilePath, currentMember.MemberId);

ATACore.TestExecution.TestListParser.GetResultRunsFromResultFile(msgArgParseResult.Results
FilePath, executionResultRunId);
    ATACore.CommandExecutor.EnqueueNewMessage(executionResultRunId,
MessageSource.ResultsParser, messagesToBeSend);
    }
    else
    {
        currentlyExecutedProc =
ATACore.CommandExecutor.QueueCommandToExecute(currentCommandXml);
    }

    string agentResponseMessage = String.Concat("Start Executing ",
ATACore.CommandExecutor.GetCommandToBeExecutedFromMessage(currentCommandXml));
    ATACore.CommandExecutor.EnqueueNewMessage(agentResponseMessage,
MessageSource.ExecutionLogger, messagesToBeSend);
    }
}

private static void InitializeMsBuildLogger()
{
    // Locks the current thread until the agentMsBuildListener is initialized
    msBuildLogListenerThreadWorker = Task.Factory.StartNew(() =>
ATACore.TcpWrapperProcessor.TcpMsBuildLoggerProcessor.ProcessMsBuildLoggerMessage(messages
ToBeSend, msBuildLogSettings), TaskCreationOptions.LongRunning);
    Thread.Sleep(1000);
}

private static void ProcessClientMessage(TcpListener agentListener)
{
    bool connected = true;
    TcpClient agentTcpListener = default(TcpClient);
    lock (lockObject)
        agentTcpListener = agentListener.AcceptTcpClient();
    BaseLogger.Log.Info(ClientMessageProcessorTcpClientConnectedMsg);
    Console.WriteLine(ClientMessageProcessorTcpClientConnectedMsg);
    ProcessClientMessageInternal(agentListener, connected, agentTcpListener);
}

private static void ProcessClientMessageInternal(TcpListener agentListener, bool
connected, TcpClient agentTcpListener)
{

```

```

while (connected)
{
    ClientMessageProcessCancellation(agentTcpListener);
    // Only one thread a time can use the Tcp objects and the listener because
of that we use mutex to protect the shared resource. Only one thread a time can own the
mutex.
    // we use the WaitOne to signal other threads that the resource is
currently used by this thread. After work we release the mutex.
    string dataFromClient =
ATACore.TcpWrapperProcessor.TcpClientWrapper.ReadLargeClientMessage(agentTcpListener);
    if (!String.IsNullOrEmpty(dataFromClient))
        connected = ProcessCurrentAgentCommand(agentTcpListener,
agentListener, dataFromClient);
}

private static void ClientMessageProcessCancellation(TcpClient agentTcpListener)
{
    if (token.IsCancellationRequested)
    {
        agentTcpListener.Close();
        token.ThrowIfCancellationRequested();
    }
}

private static bool ProcessCurrentAgentCommand(TcpClient agentTcpReader,
TcpListener agentListener, string dataFromClient)
{
    Command currentAgentCommand =
ATACore.CommandExecutor.GetCommandToBeExecutedFromMessage(dataFromClient);
    bool connected = true;
    switch (currentAgentCommand)
    {
        case Command.DISCONNECT:
            cts.Cancel();
            break;
        case Command.KILL:
            CancelCurrentlyExecutingProcess();
            break;
        default:
            Console.WriteLine(currentAgentCommand.ToString());
            ProcessDefaultAgentCommand(dataFromClient);
            break;
    }

    return connected;
}

private static void ProcessDefaultAgentCommand(string dataFromClient)
{
    commandsToBeExecuted.Enqueue(dataFromClient);
    string queuedMsg =
ATACore.CommandExecutor.GetCommandToBeExecutedFromMessage(dataFromClient).ToString();
    string agentResponseMessage = String.Format("Command {0} Enqueued on the
Agent!", queuedMsg);
    BaseLogger.Log.Info(queuedMsg);
    ATACore.CommandExecutor.EnqueueNewMessage(agentResponseMessage,
MessageSource.DenqueueLogger, messagesToBeSend);
}

```

```

    }

    private static void CancelCurrentlyExecutingProcess()
    {
        if (!currentlyExecutedProc.HasExited && currentlyExecutedProc != null)
        {
            string killMessage = String.Format("Process {0} with ID:{1} was canceled successfully on Machine: {2}", currentlyExecutedProc.ProcessName, currentlyExecutedProc.Id, Environment.MachineName);
            ATACore.CommandExecutor.EnqueueNewMessage(killMessage, MessageSource.ExecutionLogger, messagesToBeSend);
            BaseLogger.Log.Info(killMessage);
            currentlyExecutedProc.Kill();
        }
    }
}

```

Приложение 2- клас Tcp Client Wrapper

```

public class TcpClientWrapper : BaseLogger
{
    public void SendMessageToClient(TcpClient tcpClient, string messageToSend)
    {
        NetworkStream ns = tcpClient.GetStream();
        Byte[] bytesToSend = Encoding.ASCII.GetBytes(messageToSend);
        int totalBytes = bytesToSend.Length;
        int startIndex = 0;
        byte[] intBytes = BitConverter.GetBytes(totalBytes);
        if (BitConverter.IsLittleEndian)
            Array.Reverse(intBytes);
        byte[] result = intBytes;
        ns.Write(result, 0, result.Length);
        do
        {
            int endIndex = (totalBytes - startIndex) > 1000 ? 1000 : totalBytes -
startIndex;
            ns.Write(bytesToSend, startIndex, endIndex);
            ns.Flush();
            startIndex += endIndex;
        }
        while (startIndex != totalBytes);
        Log.InfoFormat("Sending >> ", messageToSend);
    }

    public string ReadLargeClientMessage(TcpClient tcpClient)
    {
        string dataFromClient = String.Empty;
        NetworkStream networkStream = tcpClient.GetStream();
        string messageChunk = String.Empty;
        int endIndexOfMsg = -1;
        byte[] bytesTotalBytes = new byte[4];
        networkStream.Read(bytesTotalBytes, 0, bytesTotalBytes.Length);
        if (BitConverter.IsLittleEndian)
            Array.Reverse(bytesTotalBytes);
    }
}

```

```

int totalBytes = BitConverter.ToInt32(bytesTotalBytes, 0);
do
{
    byte[] bytesFrom = new byte[1000];
    networkStream.Read(bytesFrom, 0, 1000);
    bytesFrom = NullRemover(bytesFrom, 1000);
    messageChunk = System.Text.Encoding.ASCII.GetString(bytesFrom);
    endIndexOfMsg = messageChunk.IndexOf("$$");
    if (endIndexOfMsg == -1)
    {
        dataFromClient += messageChunk;
    }
    else
    {
        string lastPart = messageChunk.Substring(0, endIndexOfMsg);
        dataFromClient += lastPart;
    }
}
while (endIndexOfMsg == -1);

return dataFromClient;
}

public string ReadClientMessage(TcpClient tcpClient)
{
    string dataFromClient = String.Empty;
    NetworkStream networkStream = tcpClient.GetStream();
    string messageChunk = String.Empty;
    int endIndexOfMsg = -1;
    byte[] bytesTotalBytes = new byte[4];
    networkStream.Read(bytesTotalBytes, 0, bytesTotalBytes.Length);
    if (BitConverter.IsLittleEndian)
        Array.Reverse(bytesTotalBytes);
    int totalBytes = BitConverter.ToInt32(bytesTotalBytes, 0);
    do
    {
        byte[] bytesFrom = new byte[totalBytes];
        networkStream.Read(bytesFrom, 0, totalBytes);
        dataFromClient = System.Text.Encoding.ASCII.GetString(bytesFrom);
        int endIndexOf = dataFromClient.IndexOf("$$");
        if (endIndexOf != -1)
            dataFromClient = dataFromClient.Substring(0,
dataFromClient.IndexOf("$$"));
        else
            return null;
    }
    while (endIndexOfMsg != -1);

    return dataFromClient;
}

public string ReadSimpleClientMessage(TcpClient tcpClient)
{
    string dataFromClient = String.Empty;
    NetworkStream networkStream = tcpClient.GetStream();
    string messageChunk = String.Empty;
    int endIndexOfMsg = -1;
    do

```

```

{
    byte[] bytesFrom = new byte[10025];
    networkStream.Read(bytesFrom, 0, tcpClient.ReceiveBufferSize);
    dataFromClient = System.Text.Encoding.ASCII.GetString(bytesFrom);
    int endIndexOf = dataFromClient.IndexOf("$");
    if (endIndexOf != -1)
        dataFromClient = dataFromClient.Substring(0,
dataFromClient.IndexOf("$"));
    else
        return null;
}
while (endIndexOfMsg != -1);

return dataFromClient;
}

private byte[] NullRemover(byte[] DataStream, int receiveBufferSize)
{
    int i;
    byte[] temp = new byte[receiveBufferSize];
    for (i = 0; i < receiveBufferSize; i++)
    {
        if (DataStream[i] == 0x00) break;
        temp[i] = DataStream[i];
    }
    byte[] NullLessDataStream = new byte[i];
    for (i = 0; i < NullLessDataStream.Length; i++)
    {
        NullLessDataStream[i] = temp[i];
    }
    return NullLessDataStream;
}
}

```

Приложение 3- клас CommandLineExecutor

```

public class CommandLineExecutor
{
    public const string MSBUILD_PATH =
@"C:\Windows\Microsoft.NET\Framework\v4.0.30319\MsBuild.exe";

    public Process ExecuteMsbuildProject(string msbuildProjPath, IPAddressSettings
ipAddressSettings, string additionalArgs = "")
    {
        string currentAssemblyLocation =
System.Reflection.Assembly.GetExecutingAssembly().Location;
        string currentAssemblyFullpath = String.Format(currentAssemblyLocation,
"\\AutomationTestAssistantCore.dll");
        string additionalArguments = BuildMsBuildAdditionalArguments(msbuildProjPath,
ipAddressSettings, additionalArgs, currentAssemblyFullpath);
        ProcessStartInfo procStartInfo = new ProcessStartInfo(MSBUILD_PATH,
additionalArguments);
        procStartInfo.RedirectStandardOutput = false;
        procStartInfo.UseShellExecute = true;
        procStartInfo.CreateNoWindow = false;
    }
}

```

```

    Process proc = new Process();
    proc.StartInfo = procStartInfo;
    proc.Start();
    return proc;
}

public Process ExecuteMsTest(MessageArgsMsTest messageArgs)
{
    messageArgs.CreateTestList();
    string additionalArgs = String.Concat("/p:TestListPath=\"",
messageArgs.TestListPath, "/p:ResultsFilePath=", "\"", messageArgs.ResultsFilePath, "\"");
    Process currentProcess = ExecuteMsbuildProject(messageArgs.ProjectPath,
messageArgs.IpAddressSettings, additionalArgs);

    return currentProcess;
}

public Process ExecuteMsTestSpecificList(MessageArgsMsTest messageArgs)
{
    messageArgs.CreateTestList();
    string uniqueTestResultName =
ATACore.Utilities.TimeStampGenerator.GenerateTrxFilePath(messageArgs.WorkingDir);
    string additionalArgs = String.Concat("/p:TestListPath=\"",
messageArgs.TestListPath, "\" /p:ResultsFilePath=", "\"", messageArgs.ResultsFilePath,
"\"", " /p:TestListName=", "\"", messageArgs.ListName, "\"");
    Process currentProcess = ExecuteMsbuildProject(messageArgs.ProjectPath,
messageArgs.IpAddressSettings, additionalArgs);

    return currentProcess;
}

public Process ExecuteTfsCreateNewWorkspace(MessageArgsWorkspaceCreation
messageArgs)
{
    string additionalArgs = String.Concat("/t:CreateWorkspace /p:TfsServerUrl=\"",
messageArgs.TfsServerUrl, "\" /p:WorkspaceName=\"", messageArgs.WorkspaceName, "\"
/p:TfsPath=\"", messageArgs.TfsPath, "\" /p:LocalPath=\"", messageArgs.LocalPath, "\"");
    Process currentProcess = ExecuteMsbuildProject(messageArgs.ProjectPath,
messageArgs.IpAddressSettings, additionalArgs);

    return currentProcess;
}

public Process ExecuteTfsDeleteWorkspace(MessageArgsWorkspaceDeletion messageArgs)
{
    string additionalArgs = String.Concat("/t>DeleteWorkspace /p:TfsServerUrl=\"",
messageArgs.TfsServerUrl, "\" /p:WorkspaceName=\"", messageArgs.WorkspaceName, "\"
/p:UserName=", "\"", messageArgs.UserName, "\"");
    Process currentProcess = ExecuteMsbuildProject(messageArgs.ProjectPath,
messageArgs.IpAddressSettings, additionalArgs);

    return currentProcess;
}

public Process ExecuteTfsGetLatest(MessageArgsTfsGettingLatest messageArgs)
{
    string additionalArgs = String.Concat("/t:GetLatest /p:PathToGet=\"",
messageArgs.PathToGet, "\"");

```

```

return currentProcess;
}

public Process ExecuteBuild(MessageArgsBuild messageArgs)
{
    string additionalArgs = "/t:Rebuild /p:Configuration=Release";
    Process currentProcess = ExecuteMsbuildProject(messageArgs.ProjectPath,
messageArgs.IpAddressSettings, additionalArgs);

    return currentProcess;
}

public void WaitForProcessToFinish(int procId)
{
    Process proc = Process.GetProcessById(procId);
    proc.WaitForExit();
}

private string BuildMsBuildAdditionalArguments(string msbuildProjPath,
IpAddressSettings ipAddressSettings, string additionalArgs, string
currentAssemblyFullpath)
{
    string additionalArguments = String.Concat(msbuildProjPath, " ",
additionalArgs, " ",
"/fileLoggerParameters:LogFile=MsBuildLog.txt;append=true;Verbosity=normal;Encoding=UTF-8
/l:AutomationTestAssistantCore.MsBuildLogger.TcpIpLogger,",
currentAssemblyFullpath, ";Ip=", ipAddressSettings.GetIPAddress(),
";Port=", ipAddressSettings.Port, ";");

    return additionalArguments;
}
}

```

Приложение 4- клас MemberManager

```

public class MemberManager
{
    public const string Salt = "5B-4B-05-AD-A5-1D-3E-C5-E5-62-8D-32-74-B4-8A-0A-49-74-
31-25-E4-8C-77-34-21-66-FE-64-E8-14-8C-74";

    public Member RetrieveMemberByCredentials(ATAEntities context, string userName,
string password)
    {
        string currentUserHashedPassword = CreatePasswordHash(password);
        Member currentMember = context.Members.Where(m => m.UserName.Equals(userName)
&& m.Password.Equals(currentUserHashedPassword)).FirstOrDefault();

        return currentMember;
    }

    public Statuses GetMemberStatus(Member member)
    {
        Statuses status = (Statuses)member.StatusId;

        return status;
    }
}

```



```

public Member GetMemberByUserName(ATAEntities context, string userName)
{
    Member member = context.Members.Where(x =>
x.UserName.Equals(userName)).FirstOrDefault();

    return member;
}

public Member CreateUser(ATAEntities context, string userName, string password,
string email, string tfsUserName, string role, List<string> teams, string comment)
{
    string hashedPassword = CreatePasswordHash(password);
    MemberRole currentMemberRole =
ATACore.Managers.MemberRoleManager.GetMemberRoleByRoleName(context, role);
    Member newMember = new Member()
    {
        UserName = userName,
        Password = hashedPassword,
        Email = email,
        Comment = comment,
        TfsUserName = tfsUserName,
        MemberRoleId = currentMemberRole.MemberRoleId
    };
    newMember.StatusId = (int)Statuses.ToBeApproved;
    AddTeamsToNewMember(context, teams, newMember);
    context.Members.Add(newMember);
    context.SaveChanges();

    return newMember;
}

private static void AddTeamsToNewMember(ATAEntities context, List<string> teams,
Member newMember)
{
    teams.ForEach(t =>
    {
        Team currentTeam = ATACore.Managers.TeamManager.GetTeamByName(context, t);
        newMember.Teams.Add(currentTeam);
    });
}

public void ApproveUser(ATAEntities context, string userName, MemberRoles
newMemberRole)
{
    Member memberToBeApproved = context.Members.Where(x =>
x.UserName.Equals(userName)).FirstOrDefault();
    ActivationCode ac = GenerateNewActivationCode();
    memberToBeApproved.ActivationCodes.Add(ac);
    memberToBeApproved.UserMemberRole = newMemberRole;
    memberToBeApproved.StatusId = (int)Statuses.PendingActivation;
    SendActivationEmail(memberToBeApproved);
    context.SaveChanges();
}

public void ActivateUser(ATAEntities context, string userName)
{
    Member memberToBeApproved = GetMemberByUserName(context, userName);
    memberToBeApproved.StatusId = (int)Statuses.Active;
}

```

```

context.SaveChanges();
}

private void SendActivationEmail(Member memberToBeApproved)
{
    var message = new MailMessage();
    message.To.Add(memberToBeApproved.Email);
    var client = new SmtpClient();
    message.Body = memberToBeApproved.ActivationCodes.Last(x => x.Code !=
null).Code;
    client.Send(message);
}

private ActivationCode GenerateNewActivationCode()
{
    return new ActivationCode()
    {
        Code = Guid.NewGuid().ToString(),
        CreationDate = DateTime.Now,
        ExpirationDate = DateTime.Now.AddDays(3)
    };
}

private byte[] GetBytes(string str)
{
    byte[] bytes = new byte[str.Length * sizeof(char)];
    System.Buffer.BlockCopy(str.ToCharArray(), 0, bytes, 0, bytes.Length);
    return bytes;
}

private string CreatePasswordHash(string password)
{
    byte[] saltBytes = GetBytes(Salt);
    PasswordDeriveBytes passProvider = new PasswordDeriveBytes(password,
saltBytes);
    string hashedPassword = BitConverter.ToString(passProvider.GetBytes(32));
    return hashedPassword;
}

public List<Member> GetAllMembersForApproval(ATAEntities context)
{
    var query = context.Members.Where(m => m.StatusId.Equals(2));
    return query.ToList();
}

public string GetTfsUserNameByUserName(ATAEntities context, string
currentUserName)
{
    Member m = GetMemberByUserName(context, currentUserName);
    return m.TfsUserName;
}
}

```

Приложение 5- клас RegistryManager

```
public class RegistryManager
{
    public const string MainRegistrySubKeyName = "AutomationTestAssistant";
    public const string DataRegistrySubKeyName = "data";
    public const string UserNameRegistrySubKeyName = "userName";
    public const string ThemeRegistrySubKeyName = "theme";
    public const string AppereanceRegistrySubKeyName = "Appereance";
    public const string LocalPathsRegistrySubKeyName = "LocalPaths";
    public const string WorkspaceNamesRegistrySubKeyName = "WorkspaceNames";
    public const string ColorRegistrySubKeyName = "color";

    public void WriterCurrentUserToRegistry(string userName)
    {
        RegistryKey ata = Registry.CurrentUser.CreateSubKey(MainRegistrySubKeyName);
        // Create two subkeys under HKEY_CURRENT_USER\AutomationTestAssistant. The
        // keys are disposed when execution exits the using statement.
        RegistryKey dataR = ata.CreateSubKey(DataRegistrySubKeyName);
        // Create data for the TestSettings subkey.
        dataR.SetValue(UserNameRegistrySubKeyName, userName);
        dataR.Close();
        ata.Close();
    }

    public void WriterCurrentThemeToRegistry(string currentUserName, string theme)
    {
        RegistryKey ata = Registry.CurrentUser.CreateSubKey(MainRegistrySubKeyName);
        RegistryKey dataR = ata.CreateSubKey(DataRegistrySubKeyName);
        RegistryKey currentUserR = dataR.CreateSubKey(currentUserName);
        RegistryKey appereanceR =
currentUserR.CreateSubKey(AppereanceRegistrySubKeyName);
        appereanceR.SetValue(ThemeRegistrySubKeyName, theme);
        appereanceR.Close();
        currentUserR.Close();
        dataR.Close();
        ata.Close();
    }

    public void WriterLocalPathToRegistry(string currentUserName, string tfsPath,
string localPath)
    {
        RegistryKey ata = Registry.CurrentUser.CreateSubKey(MainRegistrySubKeyName);
        RegistryKey dataR = ata.CreateSubKey(DataRegistrySubKeyName);
        RegistryKey currentUserR = dataR.CreateSubKey(currentUserName);
        RegistryKey localPathsR =
currentUserR.CreateSubKey(LocalPathsRegistrySubKeyName);
        localPathsR.SetValue(tfsPath, localPath);
        localPathsR.Close();
        currentUserR.Close();
        dataR.Close();
        ata.Close();
    }
}
```

```

    public void WriterCurrentColorsToRegistry(string currentUserName, byte red, byte green,
byte blue)
    {
        RegistryKey ata = Registry.CurrentUser.CreateSubKey(MainRegistrySubKeyName);
        RegistryKey dataR = ata.CreateSubKey(DataRegistrySubKeyName);
        RegistryKey currentUserR = dataR.CreateSubKey(currentUserName);
        RegistryKey appereanceR =
currentUserR.CreateSubKey(AppereanceRegistrySubKeyName);
        appereanceR.SetValue(ColorRegistrySubKeyName, String.Format("{0}&{1}&{2}",
red, green, blue));
        appereanceR.Close();
        currentUserR.Close();
        dataR.Close();
        ata.Close();
    }

    public string[] GetColors(string currentUserName)
    {
        string[] colorsStr = null;
        try
        {
            RegistryKey ata = Registry.CurrentUser.OpenSubKey(MainRegistrySubKeyName);
            RegistryKey dataR = ata.OpenSubKey(DataRegistrySubKeyName);
            RegistryKey currentUserR = dataR.OpenSubKey(currentUserName);
            RegistryKey appereanceR =
currentUserR.OpenSubKey(AppereanceRegistrySubKeyName);
            string colors = String.Empty;

            if (appereanceR != null && currentUserR != null && dataR != null && ata !=
null)
            {
                colors = (string)appereanceR.GetValue(ColorRegistrySubKeyName);
                appereanceR.Close();
                currentUserR.Close();
                dataR.Close();
                ata.Close();
            }
            if (!String.IsNullOrEmpty(colors))
                colorsStr = colors.Split('&');
        }
        catch { }
        return colorsStr;
    }

    public string GetUserName()
    {
        RegistryKey ata = Registry.CurrentUser.OpenSubKey(MainRegistrySubKeyName);
        RegistryKey data = ata.OpenSubKey(DataRegistrySubKeyName);
        string userName = (string)data.GetValue(UsernameRegistrySubKeyName);
        data.Close();
        ata.Close();

        return userName;
    }

    public string GetTheme(string currentUserName)
    {
        string theme = String.Empty;

```

```

try
{
    RegistryKey ata = Registry.CurrentUser.OpenSubKey(MainRegistrySubKeyName);
    RegistryKey dataR = ata.OpenSubKey(DataRegistrySubKeyName);
    RegistryKey currentUserR = dataR.OpenSubKey(currentUserName);
    RegistryKey appereanceR =
currentUserR.OpenSubKey(AppereanceRegistrySubKeyName);

    if (appereanceR != null && currentUserR != null && dataR != null && ata !=
null)
    {
        theme = (string)appereanceR.GetValue(ThemeRegistrySubKeyName);
        appereanceR.Close();
        currentUserR.Close();
        dataR.Close();
        ata.Close();
    }
}
catch { }
return theme;
}

public string GetProjectLocalPath(string userName, string tfspath)
{
    RegistryKey ata = Registry.CurrentUser.OpenSubKey(MainRegistrySubKeyName);
    RegistryKey dataR = ata.OpenSubKey(DataRegistrySubKeyName);
    RegistryKey currentUserR = dataR.OpenSubKey(userName);
    RegistryKey localPathsR =
currentUserR.OpenSubKey(LocalPathsRegistrySubKeyName);
    string localPath = String.Empty;
    if (localPathsR != null && currentUserR != null && dataR != null && ata !=
null)
    {
        localPath = (string)localPathsR.GetValue(tfspath);
        localPathsR.Close();
        currentUserR.Close();
        dataR.Close();
        ata.Close();
    }

    return localPath;
}

public string GetWorkspaceName(string userName, string localPath)
{
    RegistryKey ata = Registry.CurrentUser.OpenSubKey(MainRegistrySubKeyName);
    RegistryKey dataR = ata.OpenSubKey(DataRegistrySubKeyName);
    RegistryKey currentUserR = dataR.OpenSubKey(userName);
    RegistryKey workspaceNames =
currentUserR.OpenSubKey(WorkspaceNamesRegistrySubKeyName);
    string cWorkspaceName = String.Empty;
    if (workspaceNames != null && currentUserR != null && dataR != null && ata !=
null)
    {
        cWorkspaceName = (string)workspaceNames.GetValue(localPath);
        workspaceNames.Close();
        currentUserR.Close();
        dataR.Close();
    }
}

```

Приложение 6- клас ProkectSettingsLoadingView

```
public partial class ProjectSettingsLoadingView : UserControl, IContent
{
    private delegate void UpdateProgress(string currentProgress);
    private delegate void NavigateToNextPage();
    public bool IsLoaded { get; set; }
    public List<string> AlreadyCreatedWorkspaces { get; set; }

    public ProjectSettingsLoadingView()
    {
        InitializeComponent();
        IsLoaded = false;
        AlreadyCreatedWorkspaces = new List<string>();
    }

    private void ProcessSelectedProjectInfos()
    {
        try
        {
            string tfProjectPath = CurrentTfsProjectPath;
            string currentUserName = ATACore.RegistryManager.GetUserName();
            string currentTfsUserName =
            ATACore.Managers.MemberManager.GetTfsUserNameByUserName(ATACore.Managers.ContextManager.Co
            ntext, currentUserName);
            ATACore.Managers.ContextManager.Context.Dispose();
            List<Task> taskToBeExecuted = new List<Task>();
            taskToBeExecuted.Add(new Task(() => {}));
            foreach (var cTeam in MainWindow.AdminProjectSettingsViewModel.Teams)
            {
                foreach (var cProject in cTeam.ObservableProjects)
                {
                    if (!cProject.IsSelected)
                        continue;
                    string localPath = GetProjectLocalPath(currentUserName,
                    cProject.TfsPath);
                    if (String.IsNullOrEmpty(localPath))
                        continue;
                    string workspaceName =
                    ATACore.RegistryManager.GetWorkspaceName(currentUserName, localPath);
                    Task t = taskToBeExecuted[taskToBeExecuted.Count -
                    1].ContinueWith((antecedent) =>
                    {
                        if (!String.IsNullOrEmpty(workspaceName))
                        {
                            DeleteWorkSpaceIfExists(tfProjectPath,
                            currentTfsUserName, cProject.TfsUrl, MainWindow.LocalMsBuildLogIpSettings, workspaceName);
                            cProject.WorkspacesForDelete.Add(workspaceName);
                        }
                    });
                }
            }
        }
    }
}
```

```

taskToBeExecuted.Add(t);
    string tfsUrl = cProject.TfsUrl;
    string tfsPath = cProject.TfsPath;
    IPAddressSettings cIS = MainWindow.LocalMsBuildLogIpSettings;
    if (!AlreadyCreatedWorkspaces.Contains(tfsPath))
    {
        ATACore.Utilities.FilesDeleter.DeleteAllFilesAndFolders(localPath);
        string newWorkspaceName = Guid.NewGuid().ToString();
        Task t1 = taskToBeExecuted[taskToBeExecuted.Count -
1].ContinueWith((antecedent) => CreateNewWorkspace(tfsProjectPath, currentUserName,
tfsUrl, tfsPath, localPath, cIS, newWorkspaceName));
        taskToBeExecuted.Add(t1);
        AlreadyCreatedWorkspaces.Add(tfsPath);
    }
    Task t2 = taskToBeExecuted[taskToBeExecuted.Count -
1].ContinueWith((antecedent) => GetLatest(tfsProjectPath, currentTfsUserName, localPath,
cIS));
    taskToBeExecuted.Add(t2);
    PrepareAllCurrentProjectAddiotionalPaths(taskToBeExecuted,
tfsProjectPath, currentUserName, currentTfsUserName, cProject, cIS);
    string pName = cProject.Name;

    Task t3 = taskToBeExecuted[taskToBeExecuted.Count -
1].ContinueWith((antecedent) => BuildCurrentProject(localPath, pName, cIS));
    taskToBeExecuted.Add(t3);
}
}
Task finalTask = taskToBeExecuted[taskToBeExecuted.Count -
1].ContinueWith((antecedent) =>
{
    NavigateToNextPage ng = new NavigateToNextPage(Navigate);
    mainGrid.Dispatcher.BeginInvoke(ng, DispatcherPriority.Send);
});
taskToBeExecuted.Add(finalTask);
taskToBeExecuted[0].Start();
}
catch(Exception ex)
{
    return;
}
}

private void PrepareAllCurrentProjectAddiotionalPaths(List<Task> taskToBeExecuted,
string tfsProjectPath, string currentUserName, string currentTfsUserName, ProjectViewModel
cProject, IPAddressSettings msBuildLoggerIpSettings)
{
    foreach (var cAdditionalPath in cProject.ObservableAdditionalPaths)
    {
        string localPath = GetProjectLocalPath(currentUserName,
cAdditionalPath.TfsPath);
        string workspaceName =
ATACore.RegistryManager.GetWorkspaceName(currentUserName, localPath);
        string tfsUrl = cAdditionalPath.TfsUrl;
        string tfsPath = cAdditionalPath.TfsPath;

        Task t1 = taskToBeExecuted[taskToBeExecuted.Count -
1].ContinueWith((antecedent) =>

```

```

Task t1 = taskToBeExecuted[taskToBeExecuted.Count - 1].ContinueWith((antecedent) =>
{
    if (!String.IsNullOrEmpty(workspaceName))
    {
        DeleteWorkSpaceIfExists(tfsProjectPath, currentTfsUserName,
tfsUrl, msBuildLoggerIpSettings, workspaceName);
        cAdditionalPath.WorkspacesForDelete.Add(workspaceName);
    }
});

taskToBeExecuted.Add(t1);

if (!AlreadyCreatedWorkspaces.Contains(tfsPath))
{
    ATACore.Utilities.FilesDeleter.DeleteAllFilesAndFolders(localPath);
    string newWorkspaceName = Guid.NewGuid().ToString();
    Task t2 = taskToBeExecuted[taskToBeExecuted.Count -
1].ContinueWith((antecedent) => CreateNewWorkspace(tfsProjectPath, currentUser,
tfsUrl, tfsPath, localPath, msBuildLoggerIpSettings, newWorkspaceName));
    taskToBeExecuted.Add(t2);
    AlreadyCreatedWorkspaces.Add(tfsPath);
}
Task t3 = taskToBeExecuted[taskToBeExecuted.Count -
1].ContinueWith((antecedent) => GetLatest(tfsProjectPath, currentTfsUserName, localPath,
msBuildLoggerIpSettings));
taskToBeExecuted.Add(t3);
}
}

private void DisplayClientMessage(string text)
{
    rtbStatus.AppendText(String.Format("\n{0}\n", text));
    rtbStatus.ScrollToEnd();
}

private void Navigate()
{
    ModernWindow mw = Window.GetWindow(this) as ModernWindow;
    Uri u1 = new Uri("/TestsExecutionView.xaml", UriKind.Relative);
    mw.ContentSource = u1;
}

private void BuildCurrentProject(string localPath, string name, IpAddressSettings
msBuildLoggerIpSettings)
{
    Process currentlyExecutedProcess;
    MessageArgsBuild buildArgs = new MessageArgsBuild(localPath, name,
msBuildLoggerIpSettings);
    currentlyExecutedProcess =
ATACore.CommandLine.CommandLineExecutor.ExecuteBuild(buildArgs);
    currentlyExecutedProcess.WaitForExit(Int32.MaxValue);
}

private void GetLatest(string tfsProjectPath, string currentTfsUserName, string
localPath, IpAddressSettings msBuildLoggerIpSettings)
{
    Process currentlyExecutedProcess;

```



```

MessageArgsTfsGettingLatest tfsGetLatestArgs = new
MessageArgsTfsGettingLatest(Command.TFGL, tfsProjectPath, msBuildLoggerIpSettings,
localPath, currentTfsUserName);
    currentlyExecutedProcess =
ATACore.CommandLine.CommandLineExecutor.ExecuteTfsGetLatest(tfsGetLatestArgs);
    currentlyExecutedProcess.WaitForExit(Int32.MaxValue);
}

    private void CreateNewWorkspace(string tfsProjectPath, string currentUserName,
string tfsUrl, string tfsPath, string localPath, IpAddressSettings
msBuildLoggerIpSettings, string newWorkspaceName)
    {
        MessageArgsWorkspaceCreation tfsWorkspaceCreationArgs = new
MessageArgsWorkspaceCreation(Command.TFSWN, tfsProjectPath, tfsUrl,
msBuildLoggerIpSettings, newWorkspaceName, tfsPath, localPath);
        Process currentlyExecutedProcess =
ATACore.CommandLine.CommandLineExecutor.ExecuteTfsCreateNewWorkspace(tfsWorkspaceCreationA
rgs);
        currentlyExecutedProcess.WaitForExit(Int32.MaxValue);
        ATACore.RegistryManager.WriterWorkspaceNameToRegistry(currentUserName,
localPath, newWorkspaceName);
    }

    private void DeleteWorkSpaceIfExists(string tfsProjectPath, string
currentTfsUserName, string tfsUrl, IpAddressSettings msBuildLoggerIpSettings, string
workspaceName)
    {
        if (!String.IsNullOrEmpty(workspaceName))
        {
            MessageArgsWorkspaceDeletion tfsWorkspaceDeletionArgs = new
MessageArgsWorkspaceDeletion(Command.TFSWD, tfsProjectPath, msBuildLoggerIpSettings,
tfsUrl, workspaceName, currentTfsUserName);
            Process currentlyExecutedProcess =
ATACore.CommandLine.CommandLineExecutor.ExecuteTfsDeleteWorkspace(tfsWorkspaceDeletionArgs
);
            currentlyExecutedProcess.WaitForExit(Int32.MaxValue);
        }
    }

    private string GetProjectLocalPath(string currentUserName, string tfsPath)
    {
        string localPath =
ATACore.RegistryManager.GetProjectLocalPath(currentUserName, tfsPath);
        return localPath;
    }

    private void DisplayMsBuildLog()
    {
        UpdateProgress uiUpdater = new UpdateProgress(DisplayClientMessage);
        while (true)
        {
            try
            {
                if (MainWindow.messagesToBeSend.Count > 0)
                {
                    MessageArgsLogger msgArgs;
                    bool isDequeued = MainWindow.messagesToBeSend.TryDequeue(out
msgArgs);

```

```

    }
        }
    }
    catch(Exception ex)
    {
        break;
    }
}

private void On_Loaded(object sender, RoutedEventArgs e)
{
    if (!IsLoaded)
    {
        MainWindow.MsBuildLogListnerThreadWorker =
            Task.Factory.StartNew(() =>
                ATACore.TcpWrapperProcessor.TcpMsBuildLoggerProcessor.ProcessMsBuildLoggerMessage(MainWindow.messagesToBeSend,
                MainWindow.LocalMsBuildLogIpSettings),
                TaskCreationOptions.LongRunning);

        MainWindow.MessageLoggerThreadWorker = Task.Factory.StartNew(() =>
            DisplayMsBuildLog());
        MainWindow.GetBuildThreadWorker = Task.Factory.StartNew(() =>
            ProcessSelectedProjectInfos());
        IsLoaded = true;
    }
}

public void
OnFragmentNavigation(FirstFloor.ModernUI.Windows.Navigation.FragmentNavigationEventArgs e)
{
}

public void
OnNavigatedFrom(FirstFloor.ModernUI.Windows.Navigation.NavigationEventArgs e)
{
}

public void
OnNavigatedTo(FirstFloor.ModernUI.Windows.Navigation.NavigationEventArgs e)
{
}

public void
OnNavigatingFrom(FirstFloor.ModernUI.Windows.Navigation.NavigatingCancelEventArgs e)
{
}
}

```

8. Заключение

В настоящата дипломна работа беше разгледан проблемът за проектиране и реализация на приложение за стартиране на тестове на отдалечени машини. Програмата ще допринесе за повишаване качеството на разработвания софтуер, за който се пишат тестове. Критичните дефекти ще могат да се виждат по-бързо, когато тестовете се изпълняват на повече от една машина. Възможността резултатите да се пазят на централизирано място ще даде възможност на хората, управляващи екипите разработващи системата да правят анализ на резултатите във времето. Това ще им помогне при взимането на важни решения. Лесният и интуитивен интерфейс спомага работата на хора, които може да нямат достатъчно познания за езици по програмиране. Чрез работния процес, който се използва на агентските приложения се гарантира, че винаги тестовете се изпълняват на чиста среда и че няма да има проблеми, свързани с различни версии на кодови файлове.

Предвижда се добавяне на интеграция с различни видове тестови библиотеки, за да може повече екипи да се възползват от програмата. Ще бъде добавена възможност за записване на тестовете в групи(suite) за по-лесно изпълнение. Агентското приложение ще бъде разширено така че да може определени тестове да се изпълняват и без наша намеса, когато веднъж са настроени (scheduling). Панелът за резултатите ще се разшири и ще има възможност да показва най – различни графики на миналите пускания в зависимост от предпочитанията на текущия потребител.

9. Използвани източници

1. <http://gallio.org/Docs.aspx>
2. <http://www.nunit.org/index.php?p=documentation>
3. <http://www.plasticscm.com/infocenter/technical-articles/pnunit.aspx>
4. [http://msdn.microsoft.com/en-us/library/ms181710\(v=vs.90\).aspx](http://msdn.microsoft.com/en-us/library/ms181710(v=vs.90).aspx)
5. <http://www.methodsandtools.com/tools/tools.php?bromine>
6. <http://www.pssuk.com/Articles/SQLServer.htm>
7. <http://blogs.msdn.com/b/dsimmons/archive/2008/05/17/why-use-the-entity-framework.aspx>
8. http://en.wikipedia.org/wiki/Windows_Presentation_Foundation
9. <http://social.msdn.microsoft.com/Forums/en-US/wpf/thread/42636e55-a1e0-4b29-bbd1-cd8073585584>
10. <http://en.wikipedia.org/wiki/MSBuild>
11. <http://dotnetslackers.com/articles/net/IntroductionTo3TierArchitecture.aspx>
12. <http://msdn.microsoft.com/en-us/magazine/dd419663.aspx>