

**Test Case Manager – приложение за управление на тестови сценарии**

INFB01 Дипломиране

*Име:* **Борислав Ивов Вапцаров** *Научен ръководител:*

*Фак. №:* **F55118** **Гл. ас. д-р Стоян Боев**

*Специалност:* Информатика

*Модул:* Приложна информатика

*Департамент:* Информатика

**22.04.2018, София**

Table of Contents

[Уводна част 3](#_Toc522786106)

[Обзорна част 5](#_Toc522786107)

[Инструменти за управление на тестове 7](#_Toc522786108)

[TestRail 7](#_Toc522786109)

[XRay 8](#_Toc522786110)

[Zephyr 10](#_Toc522786111)

[Уеб или настолно приложение 12](#_Toc522786112)

[Backend 13](#_Toc522786113)

[Frontend 13](#_Toc522786114)

[Database 14](#_Toc522786115)

[Граници на разработка 14](#_Toc522786116)

[Проектантска част 15](#_Toc522786117)

[Изисквания 16](#_Toc522786118)

[Must have (Задълнителни за имплементация) 16](#_Toc522786119)

[Nice to have (Хубаво е да ги има) 17](#_Toc522786120)

[Архитектура 18](#_Toc522786121)

[Презентационен слой 18](#_Toc522786122)

# Уводна част

Като инженер „**автоматичен контрол качеството“** на софтуера в ежедневната си работа се сблъсквам с разнообразие от проблеми. Част от работата ми е да създавам ръчни (manual) тест кейсове и да ги използвам като основа за създаване на автоматични тестове, които кореспондират помежду си.

Първата стъпка от създаването на ръчни тест кейсове води до проблема, че имаме нужда от място, на което те да бъдат съхранявани, интуитивно и лесно да се поддържат и като цяло да се направи процеса удобен и лесен за изпълнение, както и да има възможност няколко инженери по контрола на качеството на софтуера да го използват едновременно (в един екип има повече от един такъв инженер).

Всяка свързана информация трябва да бъде записана и съхранявана по определен начин, а за тест кейсове се използват най-често Microsoft Word или Excel, но когато има функционалности с по-голям брой тест кейсове, възможностите на гореспонеатите методи са неефективни и са трудни за поддържане; не позволяват едновременно изпълнение, четене и писане

Има различни подходи към извършването на всичко гореопоменато. В повечето случаи информацията, която трябва да се съхрани, обнови и евентуално да се изтрие при нужда, включва следното: идентификационен номер (в поредност) и име на теста; стъпки за създаването на теста; очакван резултат за всяка стъпка или за теста като цяло; среда или специфики на браузъра; дата на създаване и промяна; статус (преминал успешно/провален/блокиран/пропуснат/неизпълнен); проблеми; свързани автоматични сценарии и техните характеристики; други добавки, и прочие.

Съществуващото решение за справянето с тази информация е специализирано за целта приложение, чиято пазарна стойност е висока и то невинаги дава желаният резултат. На настоящето ми работно място изпитваме нужда от такова приложение, но тези, които са на пазара са или твърде скъпи, или не са с желаното качество, или и двете. Поради тази причина реших да създам приложение, което отговаря на изискванията, които имаме и същевременно е безплатно и позволява надграждане и променяне в случай на приложени нови изисквания.

Заради времеотнемащата настояща процедура, основна цел и съответно резултат от дипломната работа ще бъде WPF приложение, наречено “**Test Case Manager**”, което ще бъде от тип „клиент - сървър с база от данни. Това приложение ще позволи интуитивен и лесен начин на създаване на йерархия от тест кейсове за нуждите на съответния софтуерен продукт, изпълнение на ръчните тест кейсове през отделен прозорец и процедура наречена   
  
  
  
„тест рън“, както и отбелязването на несъответствия от очакваният резултат, под формата на „бъгове“, които ще имат възможност да се регистрират в GitHub напълно автоматично.

Регистрираните бъгове ще съдържат в себе си основните параметри на тестовия сценарии, които е бил изпълнен: заглавие, приоритет, тестови стъпки, кога и кой е изпълнил теста. Приложението, ще предоставя както копиране в клипбоард-а на връзка към новосъздаденият бъг, така и наличието на директна препратка, която може да се достъпи с клик на мишката.

За да бъде реализиран проекта, трябва да решим следните задачи:

* Да се постигне лесно и интуитвно управление на множество от тестови сценарии, свързани с конкретни функционали области и свръзаните с тях проекти.
* Да се предостави възможност за едновременна работа и изпълнение на тестови сценарии от няколко души едновременно чрез „*оптимистично заключване*“ (*optimistic concurrency / locking*), тоест ще приложим стратегия, където четем даден запис от базата и когато искаме да го променим, ще проверим дали наново прочетената версия, отговаря на тази, която сме прочели в самото начало, ако да – извършваме промяната, ако ли не – то започваме от самото начало.
* Съхранение на информация за това как да се направи тестването, да се планират тестови дейности и да се дава справка за статуса на дейностите на контрола по качеството.
* Да се предостави начин за изпълнение, чрез пускането на избрани от потребителя тестове в тест рън и запазването на информацията от него.
* Лесно маркиране на преминалите тест кейсове, на провалените, блокираните и тези, преминали успешно настоящия test run.
* Да има възможност за маркиране кои тестове минават през съответния тест рън.
* Да позволява един тест да участва в множество тест рънове, както и един тест рън да има множество тестове.
* Интеграция с ***GitHub*** системата за проблеми, където само с няколко клика да може да създава или логва бъг спрямо изпълнен тест.
* Aвтентикация.
* Статистика, свързана с това колко теста са минали, колко са се провалили при миналия test run, колко от тестовете са автоматизирани и прочие.
* Наличие на екран, чрез която администратора ще може да определя кои потребители имат права да добавят други потребители, променят или трият такива.
* Лесно навигационно меню.
* Възможност за промяна на цветовете на основният прозорец (контури, цвят при селекция, основен фонов цвят с преливане).

# Обзорна част

Инструментите за управление на тестовете се използват за съхраняване на информация как тестовете трябва да бъдат изпълнени, планиране на дейностите по тестване и докладване за състоянието на дейностите по осигуряване на качеството. Инструментите дават различни подходи за тестване и по този начин имат различни групи характеристики. Обикновено те се използват за поддържане и планиране на ръчно тестване, изпълнение или събиране на данни за изпълнение от автоматизирани тестове, управление на множество среди и въвеждане на информация за открити дефекти, като тенденцията през последните години е да има отделна система за регистриране и следене на дефекти/бъгове. Инструментите за управление на тестовете предлагат перспективи за рационализиране на процеса на тестване и позволяват бърз достъп до анализ на данни, инструменти за съвместна работа и лесна комуникация в множество екипи по проекта. Много от инструментите за управление на тестове включват възможности за управление на изискванията, макар че все по-рядко срещаме тази възможност, тъй като има специализирани програми, които се занимават само с това.

Инструментите за управление на тестове дават възможност на екипите да консолидират и структурират процеса на тестване, като използват един инструмент за управление на тестовете, вместо да инсталират няколко приложения, предназначени да управляват само една стъпка от процеса. Инструментите за управление на тестовете може да предоставят на екипите възможност да управляват тестовата среда, автоматизираните тестове, дефекти и задачи по проекта, макар че подобни приложения се разработват години от големи софтуерни компании. Някои приложения включват разширени табла за управление и подробно проследяване на ключови показатели, което позволява лесно проследяване на прогреса и управлението на програмни грешки. Улеснение е за мениджърите да виждат графики спрямо как се движи работата, колко теста за изпълнени, колко са минали и колко са открили дефекти в тестовата среда.

Инструмент за управление на тестове, който включва всичкo необходимо за управление на тестовия процес, може да спести на тестери нервите и проблемите по инсталирането на отделни приложения, необходими за процеса на тестване. Те могат да бъдат реализирани с минимална програмна способност, което позволява лесен монтаж и мониторинг на тестовия процес в множество групи и/или проекти. След инсталирането, екипите имат незабавен достъп до потребителски интерфейс и могат веднага да започнат да работят и да записват тестови сценарии.

Тези типове приложения са предназначени да опростят процеса на управление на тестовете с вградени високи нива на автоматизация и проследяване, но все пак не изискват усъвършенствани умения за програмиране или знания, които да бъдат приложени. Те са полезни за екипи, които управляват разнообразни тестови сценарии и за по-големи екипи, които се нуждаят от всеобхватно приложение за управление на проекти, макар последната възможност да се предлага единствено и само при платени продукти.

При администриране на тестови случаи потребителите могат да получат достъп до разнообразни „прозорци“ за управление, за да получат незабавен достъп до данните, което прави процеса на тестване ефикасен и точен. Типът на използваният „прозорец“се определя от обхвата на проекта и от информацията и данните, които трябва да бъдат извлечени по време на процеса на тестване.

Данните могат да бъдат споделяни и достъпни в множество екипи на проекта, което позволява ефективна комуникация и сътрудничество през целия процес на тестване. Обикновено налични са йерархични структури в няколко нива, например: проект, който може да съдържа области, всяка област може да съдържа функционални или нефункционални ръчни тестове. Те от своя страна могат да се включват в колекция, която може да бъде изпълнявана многократно от тестъри или други роли в екипа.

Добрият инструмент за управление на тестовете трябва да позволява промяна или триене на всеки един обект от йерархичното ниво, както и регистриране на дефекти, които са породени от разлика между очакваният резултат от тестването и това, което се вижда от изпълняващият тестовата дейност. Задължително всеки тест трябва да съдържа в себе си стъпки, като всяка стъпка трябва да описва какво трябва да се направи и съответно какво трябва да се случи като очакван резултат.

Като резултат от задачите, трябва да получим приложение, което може директно да се използва от софтуерната индустрия. Проектът цели да олесни и помогне на инженерите по контрол на качеството на софтуера и на бизнес анализаторите да вършат по-добре работата си, използвайки разработеното приложение – **Test Case Manager**, което ще е просто и функционално.

На пазара има много инструменти, които до някаква степен предоставят на потребителите всички гореспоменати функционалности, естествено срещу крупна сума пари. От безплатните алтернативи, на пръсти се броят тези, които дори могат да се инсталират коректно, а какво остава да се използват по предназначение.

## Инструменти за управление на тестове

По-надолу ще разгледаме някои от най-известните инструменти за управление на тестове на пазара, техните плюсове и минуси, цена, разработка и ресурси, нужни за направата им. Ще направим съпоставка между тях и приложението, което разработваме.

### TestRail

Първият инструмент, който ще разгледаме е **TestRail**, разработен е от **Gurock Software** - основана през 2004 г. и с офиси във Франкфурт, Дъблин, Остин и Хюстън. Имат над 900+ софтуерни инжинери и се фокусират единствено и само върху изграждането и поддържането на мощни инструменти с красиви интерфейси, които помагат на софтуерните екипи по целия свят да доставят надежден софтуер. Gurock е компания на Idera, Inc. Idera, Inc. е компания-майка на световните брандове за продуктивност на софтуер за B2B, чиито решения позволяват на техническите потребители да правят повече с по-малко усилия и съответно по-бързо.Маркетите на Idera, Inc. обхващат три отдела - инструменти за бази данни, инструменти за разработчици и инструменти за управление на тестове. Тези три отдела са евангелизирани от милиони членове на общността и повече от 50 000 клиенти по целия свят, включително някои от най-големите медицински центрове в света и технологични компании.

Само от описанието на компанията разбираме, че **TestRail** е технически изпипан инструмент; ще предостави голяма гама от възможности за управление на тестовете, логване на бъгове и изкарване на статистики на потребителите, като освен функционалност, предоставя помощ в реално време, естествено срещу съответното заплащане, което не е малко.

**TestRail** помага за управлението и проследяването на усилията си за тестване на софтуер и да организирането на отдела за осигуряване на качеството. Неговият интуитивен уеб базиран потребителски интерфейс улеснява създаването на тестови сценарии, управлението на тестовете и координирането на целия процес на тестване. Добавя проекти с табла за управление и отчети за дейността. Позволява получаването на информация в реално време за напредъка в тестване, добавя филтри и известия по имейл.

Ще изброим някои от основните функционалности, които **TestRail** предоставя:

* Уеб базирано приложение с модерен дизайн
* Изпълнение на тестове и проследяване на тестовите резултати
* Метрики и репорти свързани с процеса по тестване
* Интеграция с основите системи за проследяване на бъгове, като **JIRA**
* Персонализиране списъци със задачи, изпращане на писма
* Гранулярни позволения, роли и потребители
* Гъвката и адаптивна среда

Най-големият минус на приложението е това, че е платено.

Цените започват от 25$/месец на потребител, когато приложението е хостнато на облачна среда:

* Не изисква поддръжка от специалисти, може да се конфигурира за минути
* Автоматични ъпдейти и оптимизация
* Защитени и бързи сървъри, на които приложението е хостнато
* Месечни или годишни плащания

И съответно от 20$/месец на потребител, като минимума е 5 потребителя и приложението е инсталирано на собствен сървър (обикновено големите мулти-милионни компании избират тази опция):

* Пълен контрол над инсталираното приложение, включително и базата от данни
* Позволява интеграцията със собствени приложения или сървиси
* Лесна за инсталация на Windows и Linux
* Годишен лиценз с минимум 5 потребителя

### XRay

Вторият инструмент е **XRay**, разработен е като допълнение към приложението **Jira**, което от своя страна е част от продуктовата гама на **Atlassian**. Разработката на **XRay** поверена на компанията **Xpand IT**, която за пръв път се свърза с Atlassian през 2004 г. и оттогава поемата ролята на единствен Atlassian експерт в Португалия.   
  
 След дълги години работа с JIRA, Confluence и други приложения на Atlassian, Xpand IT решава да смени посоката и от консултанство започват да развиват собствени продукти базирани на продуктите на Atlassian. В момента имат две от най-продаваните и най-високо оценени добавки JIRA в Atlassian Marketplace: **Xporter** и **Xray**.

Xray е пусната през 2013 г., като идеята за проекта идва от специфична нужда от един от клиентите на **Xpand IT**. Първоначалната положителна обратна връзка, дава мотивация да инвестират непрекъснато в създаването на нови функционалности към Xray. Успехът на продукта се оказва неочакван, приходите на компанията скачат експоненциално, клиентите се увеличават с над „8000%“ за по-малко от 3 месеца и Xray се превръща в „#1 Manual & Automated Test Management tool for JIRA“.

Тестването с **Jira** е лесно: можем да създаваме, преглеждаме, редактираме и клонираме тестове на Xray. Интеграцията с платформата на JIRA, дава възможност лесно и бързо да инсталираме допълнението и да започнем работата с него. Като всъщност, тестовете написани на Xray са реално регистрирани „дефекти“ в JIRA, което от своя страна ни позволява да използваме целият способ от инструменти за работа с „дефекти“ в Jira, но в контекста на тестове.

Xray дава възможност за конфигуриране на отчетни табла, управление на процеса и добавянето на персонализирани полета към бъговете, свързани с тестовете.

Ще изброим някои от основните функционалности, които **XRay** предоставя:

* Възможност за проследяване на изисквания, тестове, дефекти и тестови рънове
* Позволява преизползване на тестови стъпки в множество тестове
* Позволява писането и изпълението на “Cucumber” BDD тестове
* Има интеграция с Selenium, JUnit, Nunit, Robot, Behave, Calabash
* Има интеграция с Bamboo & Jenkins
* Интерфейса е преведен на Немски, Английски, Френски и Испански
* Дава добри репорти и метрики спрямо тестовете и отворените дефекти/бъгове
* Предоставя REST API, което може да се интегрира за нуждите на автоматизация или репорти
* Позволява импортване на тестове от CSV, HP ALM/QC, Zephyr.

Добавката е платена, но за рализка от **TestRail**, освен цената от 10$ за 10 потребителя (500$ за 25 потребителя, 1000$ за 50 и тн), трябва да заплати и за **Jira**, тъй като XRay не може да работи без него. Логичният избор за всички, които ползват **Jira**, като тул за проследяване на дефекти и планиране, би бил купуването и използването на **Xray**, вместо покупката на друг инструмент за управление на тестове, тъй като има най-добрата интеграция спрямо Jira. В случаите, когато това не е валидно, купуването и използването на Xray, като самостоятелен продукт е безмислено и алогично.

### Zephyr

**D Software Inc**. разработва технологии и продукти за управление на тестовете, като с тяхна помощ позволява достъп и преглед на данните, с цел вземане на решения, подобрение на процесите и развитие на изкуствен интелект в помощ на процесите за контрол на качеството. Компанията разработва **Zephyr**, платформа за управление на тестове, включително администрация, управление на ресурси и изисквания, настройка на мястото за съхранение на данните, проследяване на дефекти, показатели и персонализиране на таблото, както и създаването на тестови сценарии, както и график за изпъленение на тестовете. Компанията предлага и програми за обучение и технически, стратегически и консултантски услуги по проекти за осигуряване на качеството. Тя обслужва телекомуникации, медии, финансови услуги, високотехнологични, мобилни, здравни / медицински, правителствени, ИТ услуги и индустриални софтуерни индустрии. **D Software** е основана през 2007 г. и се намира в Фремонт, Калифорния.

**Zephyr** се предлага в две направление, *като самостоятелен продукт и като допълнение за Jira* (подобно на **XRay**).

#### Самостоятелен продукт

Zephyr е решение за тестване, предназначено да поддържа темпото на непрекъснатото предлагане на софтуер, а екипите да останат съсредоточени върху производителността и качеството на продукт. От **Agile** до **Automation**, от **DevOps** до **Analytics**, Zephyr покрива по-голямата част от деплоймънт процесите, тестването и управлението на ресурси и изисквания.

Предоставя следните фукнционалности:

* Проследява интегрирането на Agile методологията, от графики до следене на риска
* Скалиране
* Анализиране на метрики и процеси, подобряване на „прозрачността“ в организацията
* Деплоймънт на продукта в Cloud, Server или Data Center
* Независим от операционна система (Windows, Linux) или Бази данни (Oracle, MySQL, Microsoft SQL, postgres)
* Интеграция с JIRA, Confluence, Crowd, LDAP, Continuous Integration tools като Jenkins, Bamboo и Automation tools като Selenium, QTP/UFT
* SSL и микро-сървис базирана архитектура

Откъм цена, Zephyr залага на традиционната ежемесечна система, която предоставя на малки и средни екипи от 5 до 20 човека възможност да бъдат гъвкави относно това, колко и как ще плащат за предоставените услуги. Цените, от друга страна, едва ли са най-силната страна на Zephyr, тъй като за човек на месец искат (непосилните за мнозина) 50$ / човек / месец. Ценоразписа за 20+ човека и специалните пакети предлагани на големите ентърпрайс компании, Zephyr предлага по-ниски цени (макар и те да не се разграничават много от стандартните им до 20 човека) от 40 до 45$. При такова количество лицензи, вече предлаганите услуги се увеличават, като предоставят опции като: собствен сървър, на който да се хоства приложението, денонощен съпорт от страна на Zephyr, дейта център с множество нодове и *load balancers*, както и възможност за динамично скалиране на приложението, така че винаги да използва оптимално ресурсите на машината, на която е хостнат.

#### Допълнение към Jira

Подобно на XRay, Zephyr предлага и допълнение към Jira под формата на плъгин/аддон, който със сигурност не предоставя същите възможности, като стенд-алоун приложението, но въпреки това е в пряка конкуренция с XRay и не отстъпва по-назад от него.

Предоставя следните функции:

* Възможност за създаване на тестови сценарии
* Пълен life-cycle контрол, наследен от Jira
* Интеграция със CI & CD инструменти
* Agile метрики и функции
* Метрики относно тестването
* Голяма общност от ползватели и експерти.

Цените са близки до тези, които XRay предлага. В зависимост от потребителите, цените скалират експоненциално в зависимост дали говорим за сървър на който е хостната Jira или в cloud (като сървис):

При хостната Jira:

* 10 потребителя за 10$, 25 за 750$, 50 за 1500$, 100 за 3000$ и тн.

При използване на сървис:

* До 10 потребителя – 10$, oт 11 до 200 потребителя, по 4.25$ на месец/потребител. След 200 цената на месец за потребител пада пропорционално на потребителите.

## Уеб или настолно приложение

От разгледаните, в предишната секция, приложения за управление на тестовете, можем веднага да открием, че става дума за Уеб приложения с динамична възможност за скалиране. Големите компании разчитат на подобен тип архитектурно решение, за да могат максимално да обхванат пазара от потенциални клиенти, без да се съобразят от операционната система, наличните технологии и всичко, което може да ги обвърже към една единствена платформа. Макар горе-изброеното да е логично, има и компании, които предпочитат да създадът продукт, който е платформено ориентиран, например : **Microsoft** –Test Manager, **HP** - Quality Center, **XQUAL** – Xstudio, това им позволява да обхванат пазари, като: Китай, Индия и страните, където има цензура върху интернет пространството.

“**Test Case Manager**” ще бъде именно такова приложение, обвързано с платформа и операционна система.

За целта ще използваме стандартен трислоен модел с ясно обособени нива на архитектура:

* “Backend” (логически слой), където ще бъде изнесена цялата бизнес логика на приложението, моделите на различните компоненти и класове, комуникацията с базата от данни и управлението на ресурси, и интеграция с презентационния слой;
* “Frontend” (презентационен слой), ще е отговорен за визуализацията на моделите, преминаването между различните прозорци, ‘desktop responsive layout’-а и интеракциите, които ще се предават от “frontend” към “backend”;
* Традиционна база от данни, на която ще се съхранява информацията, създадена в самото приложение, ще съдържа различни видове таблици, така че да създаде йерархична структура на данните.

Технологиите, които ще се използват за направата на приложението за управление на тестови сценарии, са част от .NET платформата за създадване на софтуерни приложение и главно използвана под операционната система на Майкрософт – Уиндоус; .NET фреймуърка позволява управление на паметта и други системни услуги, както и широка набор от библиотеки, за улеснение на програмистите и възможността да използват надежден ресурс за разработка на всички основни области на разработка на приложения като: уеб, десктоп, мобилни, игрални и други видове. Езиците, с които .NET може да работи, компилира и интерпретира са: C#, F# и Visual Basic.

### Backend

В този архитектурен слой, ще се използва:

* **C#** - типово-базиран, обектно-ориентиран език, който позволява разработването на различни приложения в .NET света. Избран е като основен език за разработване на имплементацията на приложението, защото той е в основата на развитието ми като специалист и ще мога лесно и адекватно да се справя с възложената задача.
* **EntityFramework** ще бъде използван за „Database <-> Backend layer“, и e базиран на обектно-релационен модел за мапване между обекти и таблици и позволява на .Нет програмистите да работят директно с релационни бази от данни, използвайки специфични за домейна обекти. Позволява лесно да се осъществи комуникация между бекенд и дб слоя, обработка на заявки, работа с таблици и тн.
* **GitHub API** – което ще „консумираме“ за да регистрираме бъгове в системата на GitHub, възоснова на предварително избран проект, за който текущият потребителя на системата има достъп.
* **CryptoManager** – основно, КриптоМениджъра ще бъде използван за криптиране и декриптиране на поверителна информация чрез локално инсталиран сертификат с парола, на машината на ползвателя на приложението.

### Frontend

За визуализацията на потребителската част ще използваме:

* **WPF with XAML** - Windows Presentation Foundation е графична подсистема на Microsoft за рендиране на потребителски интерфейси в базирани на Windows приложения, а XAML е прост език, базиран на XML и позволява създаването и инициализирането на .NET обекти с йерархични връзки. Въпреки че първоначално е бил ексклузивно изобретен за подпомагането на WPF, той може да се използва за създаване на всякакъв вид обектни дървета.
* **ModernUI.WPF** – NuGet допълнение, което надгражда основните интерфейс елементи (предоставени от WPF), добавяйки интерактивност и модерна визуализация със заоблени ъгли и възможност за смяна на буквеният фонт в реално време.
* **Extended WPF Toolkit** – Представя набор от визуални контроли, които не са включени в списъка от предоставени от Майкрософт.

### Database

* **MSSQL DB** (SQL Server) - Microsoft SQL Server е система за управление на релационни бази данни, разработена от Microsoft. Може да се използва като сървър на база данни, чиято основна функция е съхранението и извличането на данни. Обикновено се използва заедно с **EntityFramework** при разработка на приложения.
* **Microsoft Azure SQL Database** - Microsoft Azure SQL Database е клауд база от данни, предоставена като част от Microsoft Azure. Базата данни за облаци е база данни, която се изпълнява на платформа за изчислителни системи и достъпът до нея се предоставя като услуга (тоест не ни трябва физически сървър за хостване й).

## Граници на разработка

Въпреки желанието да се разработи приложение, което ще разреши всички негативни черти на вече наличните на пазара, това ще изисква прекалено голям ресурс от хора, време и пари, а за целта на дипломната работа нямаме достъп до нито едно от тези неща, заради това се налага поставянето на постижими граници върху разработката на приложението.

Основните функционалности, които ще бъдат налични при завършване на курсовата работа, ще предоставят възможност на потребителя да се вписва в системата, с предварително създадени потребителско име и парола, като непосредствено, след успешно влизане в системата, ще се зарежда текущите проекти, към чиято база се свързва приложението.

Всяка база от своя страна ще съдържа дървовидно-йерархична структура, която ще репрезентира видимата част на екрана на приложението. В нея ще намерим множество проекти, към които има логическо свързани структури от данни, позволяващи групирането на създадените ръчни тестове в приложението (или функционалности).

Ръчните тестове трябва да съдържат стъпки, които обособяват самите тестове в завършен сценарии; тези стъпки могат да са неограничен брой с възможност за промяна и триене. Като лимитация, създадена стъпка няма да може да се преизползва в множество тестове, а участва единствено и само в теста, в който е добавена. Тази лимитация може в бъдещете да се промени в презентационният слой.

Ръчните тестове, ще могат да участват в множество тест „рънове“ и при наличието на негативен завършек от изпълнен ръчен тест, ще имаме възможността да създаваме дефект-репорти в GitHub (с който ще имаме интеграция възоснова на наличното за програмисти публично C# SDK).

Приложението ще позволява семпла възможност за създаване и триене на потребители от специален екран за управление, от който ще можем да избере измежду две възможните роли: *стандартна* и *админска*, разликата между двете се състои в това, кой може да вижда екрана за управление на потребителите и съотвено кой може да създава и трие такива. По-сложна схема с различни роли и “permissions” се обезмисля за приложение с няколко екрана и липса на различни логически необвързани функционалности.

Паролата на всеки регистриран потребител ще се съхранява криптирана в базата, чрез сертификат, който е предварително инсталиран на потребителската машина. При самото влизане, ще се определя дали потребителя има или няма достъп до екрана с потребителите. Триенето и променянето на потребители ще става само от такива, с администраторски права.

# Проектантска част

За да можем да проектираме система за управление на тестови сценарии, трябва да обърнем внимание върху потребността на потребителя и нуждата от специфична функционалност, която обособява приложение от такъв характер и възоснова на него да планираме. То от своя страна е обект на дейност, когато искаме да открием нещата, от които проектът има нужда. Важна задача в създаването на софтуерна програма е извличането на изискванията и техния анализ. Клиентите обикновено имат обща представа за това, какво искат като краен резултат, но не знаят какво трябва да прави софтуерът. В този етап умели и опитни софтуерни инженери разпознават непълните, двусмислени и понякога противоречиви изисквания.[[1]](#footnote-1)

Наличието на изисквания от своя страна, ще определи нужните функционалности, които трябва да се разработят и предоставят на потребителя / клиента. Тези изисквания за текущият проект, са вече предоставени и съдържат в себе си следните нужни за разработване и имплементиране функционалности:

## Изисквания

1. Настолно приложение, което да предоставя набор от инструменти за създаване на тестови сценарии от множество потребители.
2. В наборът от инструменти трябва да се включат следните функционалности:

### Must have (Задълнителни за имплементация)

* + Създаване на дървовидна структура на три (3) нива, включваща в себе си ниво на проект (project), ниво на функционалност (area) и ниво, на което ще се подвизават самите тестови сценарии.

***Пример:***

* + - Проект 1
      * Функционалност 1.1
      * Функционалност 1.2
        + Тест 1 – Функционалност 1.2
      * Функционалност 1. 3
        + Тест 1 – Функционалност 1.3
        + Тест 2 – Функционалност 1.3
    - Проект 2
      * Функционалност 2.1
        + Тест 1 – Функционалност 2.1
        + Тест 2 – Функционалност 2.2
        + Тест 3 – Функционалност 2.3
    - Проект 3 (и тн..)

***Край на пример****;*

* + Всеки един тест от своя страна да трябва да съдържа наборс от стъпки (неограничен брой), инкрементирани със число, отговарящо на номера на текущата стъпка.

***Пример:***

* + - Тест 1
      * Стъпка 1
      * Стъпка 2
      * Стъпка 3
    - Тест 2
      * Стъпка 1
      * Стъпка 2

***Край на пример****;*

* + Всяка стъпка може да бъде изтрита от теста (при изтриване, стъпката изчезва от екрана).
  + Всяка стъпка в себе си трябва да пази информация за това какво трябва да се изпълни от ползвателя на системата и какъв е очакваният резултат от действието. Разграничаването между двете части на стъпката трябва да се осъществи чрез така нареченият „борд разделител“.

***Пример:***

* + - Стъпка 1 (counter is visible) [ **Action** | **Expected result** ]
    - Стъпка 2 (counter is visible) [ **Action** | **Expected result** ]
    - Стъпка 3 (counter is visible) [ **Action** | **Expected result** ]

***Край на пример****;*

* + На отделен таб или прозорец в приложението, трябва да се имплементира функционалност за създаване на „тестови рънове“. „Тестови рън“ е „контейнер“, в който може да добавяме и премахваме ръчни тестове. Един „тестови рън“ може да съдържа множество различни и уникални ръчни тестове, а един ръчен тест, от своя страна, може да бъде включен в множество „тестови рънове“.
  + „Тест ръновете“ от своя страна трябва да могат да се изпълняват многократно. При пускането на един тест рън, нов прозорец или таб трябва да се отваря, където спрямо всеки един ръчен тест (включен в тестовият рън) трябва да можем да отбележим дали е “***pass***”,”***fail***”, “***blocking***” или не е изпълнен.
  + При наличие на проблем с някой от изпъленените ръчни тестове, трябва да можем да създадем „бъг“ в избрана предварително бъг-тракинг или сорс-контрол система.
  + Новосъздаденият бъг трябва да съдържа информация за теста, темплейт със стъпките за изпълнението му и заглавие, отговарящо на заглавието на изпълненият ръчен тест кейс.
  + Изпълненият „тест рън“ трябва да съдържа в себе си метрики от тип: колко теста са включени в него, колко от тях са изпъленени, колко са със статус „***pass***” или “***fail***”.
  + На отделен таб или прозорец в приложението, трябва да се имплементира функционалност за създаване и триене на потребители. Този таб или прозорец трябва да бъде достъпен единствено и само от потребители с „администраторски права“.

### Nice to have (Хубаво е да ги има)

* + Възможност за промяната на фона на приложението – цветово или директно с избрано приложение.
  + Визуално представяне на зареждането на отделните елементи или екрани (наличие на *loading* *circle*, *spinner* или друг способ).

## Архитектура

Технологичният “*стак*” избран за разработването на дипломната работа се базира на .NET технологиите и в частност, тези, свързани с разработването на настолни приложения. В исторически план, разработването на подобен тип софтуер, изпълняващ се предимно под Windows, се е извръшвало чрез WinForms (еквивалент на AWT в Java), обаче с навлизането на “Метро“ приложенията в Windows 7, Microsoft обявява много по-гъвкав и лесен за употреба фреймуърк, който трябва да надгради и замени WinForms, а това е WPF (Windows Presentation Foundation).

WPF е фреймуърк за разработване на настолни приложения и прилежащият към тях потребителски интерфейс. Поддържа широк набор от функции за разработка на приложения, включително: модел на приложението, работа с ресурси, контроли, графики, оформление, свързване на данни, документи и сигурност. WPF използва Extensible Markup Language (XAML), за да осигури декларативен модел за създаване на потребителския интерфейс.

За нуждите на разработваното приложение, ще използваме трислойна архитектура, съдържаща в себе си презентационен слой (потребителска част), логически слой (бизнес логика) и дата слой, където ще съхраняваме данните на приложението.

### Презентационен слой

XAML е декларативен „markup” език (подобен на XML) и се прилага върху „бизнес” модела за програмиране в .NET Framework. Той опростява създаването на потребителски интерфейс чрез създаване на видимите елементи посредством декларативен подход и след това да разделим дефиницията на потребителския интерфейс от бизнес логиката, която се изпълнява по време на работа.   
  
XAML позволява работен процес, при който различни фреймуърци могат да работят едновременно върху потребителския интерфейс и бизнес логиката на дадено приложение, използвайки различни инструменти.

Когато са представени като текст, XAML файловете са XML файлове, които обикновено имат разширение .xaml. Файловете могат да бъдат кодирани (encoding) от всяко XML кодиране, като UTF-8 е избран по подразбиране.

Чрез XAML, можем да моделираме и изобразяваме различни WPF контроли. Да създаваме панели, табове, анимации, промяна на резолюцията и прилежащите й елементи и свързване на елементи от екрана с бизнес логика в бек-енд частта на приложението.

Всеки един екран на приложението за управление на тестовете ще се дефинира в самостоятелен XAML файл и прилежащата към него “code-behind”. В този файл ще дефинираме по какъв начин ще бъдат подредени бутони, полета и таблици по екрана, какви са размерите на всеки един от елементите, дали са вързани с бизнес логика или не, какво се случва когато се кликне с мишката върху тях, кога се крият и кога се показват.

Спрямо зададените изисквания, можем да идентифицираме 4 основни екрана и няколко под-прозореца нужни за предоставяне на нужната функционалност, без която приложението няма да е пълноценно.

**Основни екрани:**

* Първоначален екран на приложението с налични полета за потребилско име, парола и бутон за влизане, нужен за автентикация на потребителя. От този екран трябва да са видими всички други екрана (в бар с навигация) за които потребителя не е нужно да е влязъл в системата или да има каквито и да било права.
* Непосредствено след влизане и автентикиране в системата, потребителя трябва да бъде препратен към екран, показващ йерархичната структура на проекти -> функционалност -> тест кейс. При избран тестови сценарии, трябва да може да вижда стъпките за изпълнението му; кой го е създал; с какъв приоритет е и друга информация, която може да е нужна на ползвателя на приложението. Като допълнение към горепосоченото, потребителя трябва да има възможност за създаване, променяне и триене на проекти, функционалност и тестови сценарии.
* След избиране на „Тестови рън“ потребителя трябва да бъде препратен към отделен екран, където са съхранени всички създадени и/или изпълнени тестови рънове. От този екран може да създаде нов рън и/или да добави тестови сценарии към него посредством поп-ъп прозорец. От същия екран може да избере да стартира други (вече създадени) тестови рънове, като в този случай се отваря нов поп-ъп прозорец, където всички тестови сценарии, ще могат да се изпълнят индивидуално с прилежащият към резултата от изпълението статус.

### Бизнес слой

Windows Presentation Foundation (WPF) е графична подсистема на Microsoft за рендиране на потребителски интерфейси в базирани на Windows приложения. WPF, известен преди като "Avalon", първоначално е пуснат като част от .NET Framework 3.0 през 2006 г. Използвайки иновативната за тази година технология на DirectX за рендериране на потребителският интефейс.

WPF предоставя програмен модел за изграждане на приложения и разделя потребителския интерфейс от бизнес логиката. Моделът може да бъде опреличен като XML-ориентирани обектни модели, като тези, изпълнявани в XUL и SVG.

Както е обяснено в „презентационният слой“, WPF използва XAML, базиран на XML език, за дефиниране и свързване на различни елементи на интерфейса. WPF приложенията могат да бъдат използвани като самостоятелни програми за настолни компютри или да бъдат „хоствани“ като вградени обекти в някой уебсайт. WPF има за цел да обедини множество общи елементи на потребителския интерфейс, като например: 2D / 3D рендиране, фиксирани и адаптивни документи, типография, векторни графики, анимация по време на изпълнение и предварително зададени носители. Тези елементи могат да бъдат свързани и манипулирани въз основа на различни събития (евенти), потребителски взаимодействия и/или да бъдат свързвани с данни в реално време.

WPF runtime библиотеките, от своя страна, са включени във всички версии на Microsoft Windows от Windows Vista и Windows Server 2008 към днешно време. Потребителите на Windows XP SP2 / SP3 и Windows Server 2003 могат по желание да инсталират необходимите библиотеки (посредством .NET фреймуърк с подходяща версия).

WPF се използва и в Microsoft Silverlight, който от своя страна предоставя функционалност, която е предимно подмножество на WPF контролите и модела на изграждане, за да осигури енкапсулиран вариант на приложението с вградени уеб контроли, подобни на Adobe Flash.   
  
 За целите на дипломната работа, ще използваме WPF като основен фреймуърк за играждането на десктоп приложението. Като основен архитектурен модел за играждането на проекта ще използваме частично MVVM (Model – View – ViewModel).  
  
MVVM от своя страна подпомага отделянето на графичния потребителски интерфейс чрез *markup* език или GUI код - от бизнес логиката или моделът на данните.

Моделът на презентационната част може да бъде приложен като модел на посредник, който организира достъпа до бизнес логиката от типа "backend". MVVM е реално различна форма на модела за представяне на Martin Fowler (Martin Fowler’s Presentation model). MVVM абстрахира дефиницията и логиката на визуалната част, по начин който не зависи от конкретна платформа на потребителския интерфейс.

MVVM е изобретен от архитектите на Microsoft - Кен Купър и Тед Питърс, специално за опростяване на разработването на приложения, когато се изполват „евенти“, като начин за вътрешна комуникация между различните компоненти. Моделът е включен в Windows Presentation Foundation (WPF - Microsoft .NET графична система) и Silverlight (WPF's Internet Application под-фреймуърк). Джон Госман, един от архитектите на WPF и Silverlight на Microsoft, го показва и представя като архитектурен патърн за първи път в блога си през 2005 г.

1. [Разработка на софтуер](https://bg.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0_%D0%BD%D0%B0_%D1%81%D0%BE%D1%84%D1%82%D1%83%D0%B5%D1%80) [↑](#footnote-ref-1)