СОФИЙСКИ УНИВЕРСИТЕТ „СВ.КЛИМЕНТ ОХРИДСКИ”
ФАКУЛТЕТ ПО МАТЕМАТИКА И ИНФОРМАТИКА
КАТЕДРА “ИНФОРМАЦИОННИ ТЕХНОЛОГИИ”

ДИПЛОМНА РАБОТА

на тема

Система за управление на софтуерни проекти в малки и средни фирми

Ръководител
доц. д-р. Красен Стефанов

Дипломант
Весела Стоянова Минчева
факултетен номер: M21861
специалност „Разпределени системи и мобилни технологии”

Консултант
гл. ас. Елиза Стефанова

София, 2007

Съдържание

Въведение

Повечето софтуерни фирми започват работа с няколко софтуерни разработчика и поемат предимно малки проекти изискващи кратки периоди за изпълнение. Ако фирмата завоюва място на пазара скоро след това, в повечето случаи, следва бързо разрастване и малката фирма се превръща в средна. Сложността, размера и времето за изпълнение на проектите нараства. През този период рязко нараства броя на софтуерните разработчици, появяват се нови позиции в структурата на фирмата. Една от големите промени съпътстваща разширението е строгото обособяване на екипи и поверяването на даден проект на даден екип. Това налага назначаването на ръководител за всеки екип, ръководил на проект и т.н. Целта е фирмата да се раздели на малки относително независими една от друга единици, като всяка от тях е способна да поеме отговорността за цялостната разработка на даден проект. В този момент се появява и необходимост от централизиран софтуер, който да подпомага комуникацията и разпределението на отговорностите във фирма, която вече наброява 30-40 разработчика. Това означава, че е необходим бърз достъп до информация в електронен вид за текущото състояние на проекта.

Ето защо, в повечето случаи, в тази млада средна фирма се въвеждат софтуер за управление на проекти, който се оказва твърде тежък и тромав за размерите на компанията и най-вече за размера на един екип. За предпочитане в малките и средни фирми, е да се използва по- лек продукт, който дава само най-важната информация относно текущите задачи и не изисква особено време за поддръжка.

Текущата дипломна работа се фокусира върху реализацията на такъв продукт. Системата не предлага алгоритми и методологии в посочената сфера, а работи като хранилище на информация за проектите и подпомага преди всичко ежедневната работа на разработчиците, без да отнема от времето им. Накратко, това е система обслужваща екипа като самостоятелна единица.

Изложението на дипломната работа е структурирано в четири глави.

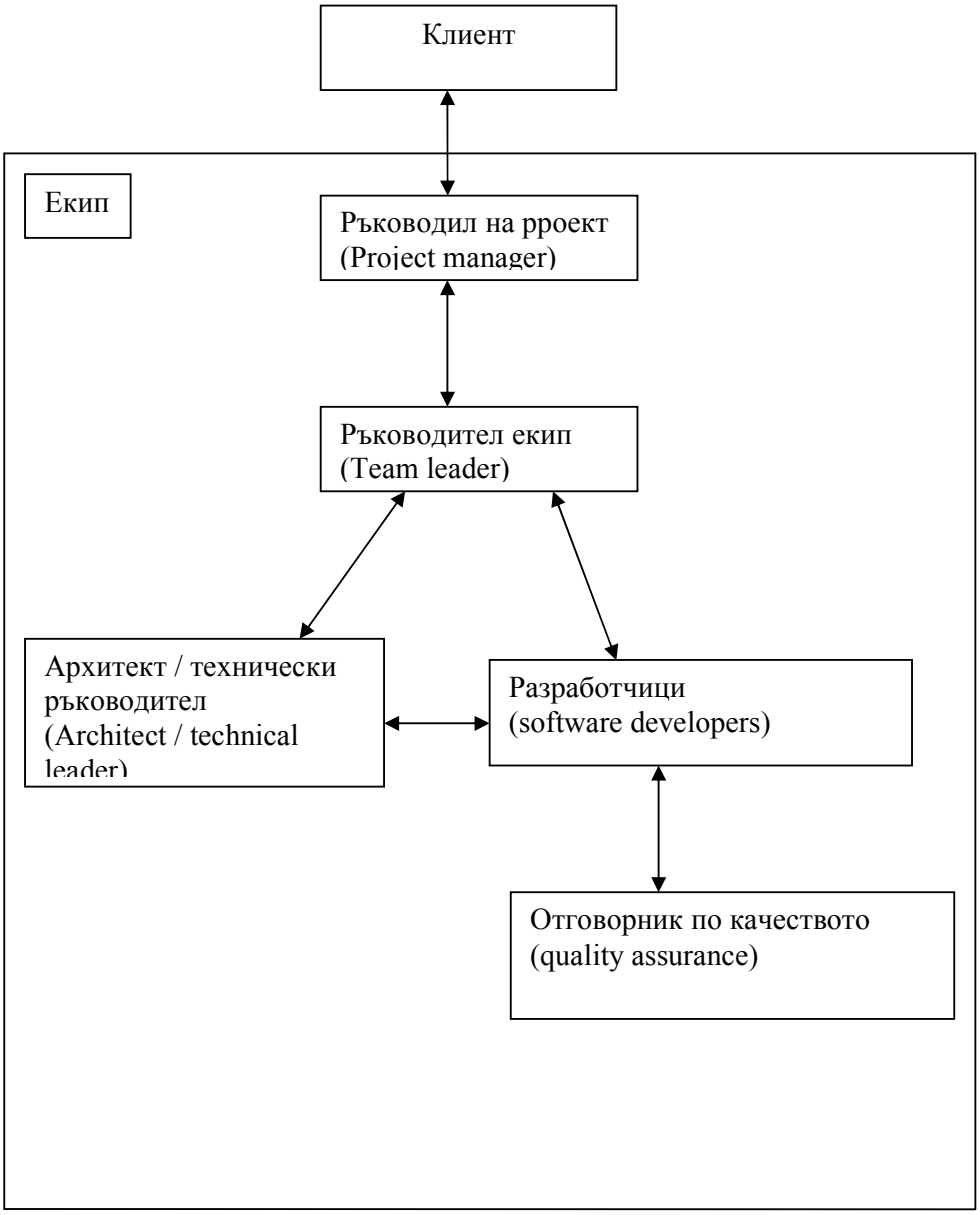
Глава 1 започва с кратко въведение в проблемната област и описание на необходимостта от такъв продукт. Описва се структурата на екип в малките и средни фирми, ролите на членовете на екипа, какви са техните отговорности и нужди. Във втората част на тази глава се разглеждат няколко от най-популярните решения и се прави сравнителна характеристика. **Глава 2** описва технологичните средства използвани за разработката на текущата дипломна работа. Също така дава обосновка за техния избор. **Глава 3** описва първоначалния план за реализация и неговото изпълнение. **Глава 4** описва техническата реализация на продукта.

Глава 1. Анализ на изискванията и обзор на възможните решения

1.1 Анализ на изискванията

Екипа, на който е поверен проекта е самостоятелно действаща единица в рамките на фирмата. При малките и средни фирми, екипа обикновено наброява 5 до 10 служителя и има следната структура(фиг. 1):

Фиг. 1 Структура на екип в малка и средна фирма



- Ръкодител на проект: технически грамотно лице, което обикновенно контактува с клиента. Той трябва да раздели проекта на функционални модули и по този начин да систематизира и подреди изискванията на клиента. Обикновенно работата му е почти изцяло свързана с писане на документи отразяващи изискванията на клиента.
- Ръководител екип: отговорен за съставянето на подробен списък от задачи, които да бъдат разпределени между разработчиците. Много е важно да разпредели задачите по време и сложност по такъв начин, че да се избегне „момента на изчакване", т.е да няма в даден интервал от време разработчик без работа, който изчаква задача възложена на друг да бъде довършена.
- Архитект / технически ръководител: определя архитектурата на крайния продукт и подбира технологиите, които ще бъдат използвани
- Разработчик: крайния производител на код (source code).
- Отговорник по качеството: лице отговорно за тестването на крайния продукт. Тестването може да се раздели на тестове за производителност, тестове за бързодействие, тестове при голямо натоварване на системата, тестове с невалидни данни, тестове проверяващи съответствие между клиентски изисквания и краен резултат и т.н.

Във всеки един момент от развитието на проекта могат да настъпят изменения. Някои примери са:

1. Клиента променя изискванията си – сменя приоритета на задачите или добавя нови изисквания.
2. Проекта се оказва зависим от проблеми в избраните технологиите – избрания набор от софтуер, като уеб и майл сървери или система за управление на база данни (СУБД) са несъвместими с операционната система.
3. Налага се промяна в архитектурата на проекта. Това може да бъде следствие от горепосочените примери.

4. Проекта е косвено свързан с друг продукт, който е възложен на друг екип. Това води до забавяне на една или няколко задачи, което рефлектира върху други аспекти на проекта. Към тази група спада следната ситуация: Даден е уеб проект, чиито дизайн е възложен на друга фирма или екип. Забавянето в дизайна води до просрочване на всички задачи свързани с графичния потребителски интерфейс
5. Непредвидени отсъствия на членове на екипа (по болест)

Както става ясно е необходима постоянна комуникация между клиента и ръководителя на проекта, както и между всички членове на екипа. Налага се всяко решение да бъде документирано и всеки в екипа да има достъп до информацията. Така грешките, в следствие на неразбиране на целите и задачите, в разработвания продукт могат да се сведат до минимум.

Изложението дотук налага следните изисквания към системата:

1. Трябва да е лесно достъпна и да не изисква исталация на допълнителен софтуер, тъй като член на екипа може да е извън офиса – например, ръководителя на проекта е в друг град за среща с клиента. Поради тази причина системата за управление е изградена като уеб приложение.
2. Системата за управление дава възможност за дефиниране на екип и проекти принадлежащи на екипа
3. Проследява се версията на документи прикачени към проект или задача
4. Системата абстрахира членовете на екипа от проблемите на ръководството. Дефинирането на цели на проекта, дава възможност на всеки член на екипа да вижда текущите си задачи и да ги разделя на по-малки елементи, които да подреди в даденото му време. Разбиването на дадена задача на по- малки остава в личното пространство на потребителя.
5. Задачата е абстрактно понятие – за ръководителя на проекта задача е изясняването с клиента на дадено изискване; за ръководителя на екипа - планиране на работата на разработчиците, за техническия ръководил - поддържането на тестова система и т.н. Това ниво на абстракция е необходимо в малките и средни фирми и поради факта, че често ръководителя на екипа и архитекта са и разработчици.
6. Следеното на статуса на всяка задача дава възможност на отговорника по качеството да тества дадена функционалност веднага след като

разработчика приключи свързаната с нея задача. Това спомага за навременното откриване на грешки и тяхното отстраняване.

7. От гледна точка на ръководителя на екип е необходимо да има лесен и интуитивен достъп до времевото разпределение на задачите от даден проект. Това налага графичен или дървовиден изглед върху множеството задачи.

1.2 Анализ на съществуващи решения

1.2.1 Gantt Project

Gantt Project [1] е инструмент само и единствено за планиране на задачи. Предоставя графично представяне на времевото разпределение, както и натоварването на всеки член на екипа. Gantt диаграмите визуализират началото и края на всяка задача, както и кои задачи са зависими една от друга. Тази система е много подходяща за малки проекти, които се изпълняват в кратки срокове. Поради ограничената информация, която може да въведе за всяка задача, тази система е неподходяща по сложни проекти. Не разполага с подсистема за следене на документи. Също така системата трябва да се инсталира за всеки член на екипа. Проекта може да се изнася (export) и внася (import) под различни формати:

- XML - Той е лесен за редактиране с помощта на обикновен текстови редактори и в него са записани списъкът със задачите и връзките между тях
- може да експортира гант графиката като PNG/JPG изображения.
- HTML/PDF – различните страници съдържат цялата информация за проекта
- Сървър – От версия 1.9.8 е възможно изтеглянето и запазването на проект на сървър поддържащ WebDAV протокол. Стандартния Apache сървър не поддържа този протокол

1.2.2 GForge

Gforge [2] е софтуер с отворен код чрез който могат да бъдат публикувани софтуерни проекти. Публикуването е такова че кода, документацията и всичко свързано с даден проект е достъпно от всеки който иска да види ресурсите. По този начин се дава възможност за обратна връзка между потребителите и разработчиците на проекта. Базиран е на PHP и Apache. Предоставя конфигурирана система за разработка включваща:

- поддръжка на версиите чрез CVS
- уеб сайт на проекта
- множество инструменти за комуникация между членовете на екипа- дискусионни форуми за членовете на екипа, система за проследяване на дефекти (bug tracking system), комуникация посредством маил групи, споделяне на документация, създаване на todo списъци, публикуване на новини и др.

От казаното дотук става ясно че GForge представлява централизирана точка за достъп до набор от полезни за развитието на проекта инструмента:

- CVS
- Майл списъци
- Дискусионни форуми
- bug tracking system
- Уеб интерфейс до CVS
- Списъци със задачи
- Статистика за членовете на екипа, броя майл списъци, брой теми на дискусионните форуми и т.н.

Като цяло GForge е мощен инструмент за управление на проекти. За съжаление предоставя твърде много функционалност която няма приложение в малките и средни фирми. GForge е особено подходящ за проекти с отворен код, в които участват разработчици от различни страни.

1.2.3 Microsoft Project Standard

Microsoft Project е набор от продукти с множество възможности в разглежданата област. Фамилията продукти включва:

- Microsoft Project Standard – настолно приложение за управление на проекти, базирано на Microsoft Windows. От версия Microsoft Project 2002 Standard е съвместим със съответния сървър Microsoft Project 2002 Server. Това дава възможност за съвместна работа с останалите членове на екипа.
- Microsoft Project Professional настолно приложение за управление на проекти, базирано на Microsoft Windows. Притежава всички характеристики на Microsoft Project Standard плюс допълнителни възможности за планиране и комуникация в екипа при използване на Microsoft Project Server.
- Microsoft Project Server – решение базирано на Интранет (Intranet), което позволява съвместна работа по проект на ниво организация (с Microsoft Project Professional) или на ниво работна група (с Microsoft Project Standard)
- Microsoft Project WebAccess – базиран на Internet Explorer интерфейс за работа с Microsoft Project Server

Едно голямо предимство на Microsoft Project е включената „машина за планиране- изчислителна машина, която може да решава проблеми, например ефекта на вълните” [3]. Това означава, че когато бъде променена началната дата на една задача или има закъснение в изпълнението й, то тези фактори ще бъдат отразени и върху останалите задачи зависи от нея.

1.2.4 Concurrent Versions System (CVS)

Concurrent Versions System [4] е една от най- разпространените системи с отворен код за контрол над набор от файлове, чрез поддържане на версии и история на за всяка промяна. Използва се предимно за съхранение на код на даден софтуерен продукт, върху който работи повече от един разработчик. В

действителност, няма причина поради, която CVS да не бъде използван за съхранението на каквито и да е файлове, както и за обслужване само на един клиент.

CVS използва типична клиент-сървър архитектура. На сървъра се съхранява текущата версия на проекта и тяхната история (history), т.е. кой, кога и какво е променил в набора файлове. Клиента се свързва към сървъра за да изтегли, редактира или качи файлове. Сред най-големите преимущества на CVS са следните:

- Нито една редакция не е окончателна. Всяка грешка може да бъде поправена, тъй като системата поддържа история на промените
- Позволява много разработчици да работят по едни и същи файлове без да рискуват загуба на информация
- Системата позволява да се подържат няколко реализации на софтуерния продукт по едно и също време без това да пречи на разработчиците да продължат работа. Това означава че не се налага работата да спира когато се спира разработката („code freeze”) точно преди релийз.
- Възможно е да се изтегли версия на проекта от дадена дата и да се провери неговото състояние. Тази характеристика на CVS подпомага бързото откриване на дефекти в продукта, и тяхното отстраняване.
- Администратора на системата може да задава права на потребителите. Възможно е да се даде и свободен достъп до файловете (например при проектите с отворен код)

1.2.5 Сравнителна характеристика

Както става ясно, CVS е мощен инструмент за контрол на съдържанието на файлове. CVS и подобните на него системи за version controll са неотменима част от контрола на разработката на софтуер в екип, но не предоставя никакви възможности за планиране и разпределение на задачите.

Фиг. 2 Сравнение на някои съществуващи решения за управление на проекти

Характеристики	Gantt Project	GForge	Microsoft Project Standard	CVN
Дефиниране на потребители	-	+	+ (дефинират се като ресурси)	+
Дефиниране на екипи	-	+	-	-
Поддържане на потребителски групи и техните права	-	+	-	+
Съхранение на информацията на общодостъпно през интернет място	- (въпреки наличието на възможност за съхранение върху web сървър, изисквания протокол е силно ограничение)	+	- (само ако се използва съвместно с Microsoft Project Server и Microsoft Project WebAccess)	+
Не изисква инсталация на всяка машина	-	+	-	-
Абстрактност на понятието задача	+	+	+	-
Управление на	-	+	-	-

задачи в личното пространство				
Следене на статуса на задачи и проекти	+	-	+	-
Дефиниране на множество проекти	-	+	-	+
Интуитивна визуализация на времевото разпределение на задачите	+	+	+	-
Прикачване на документи към задачи и проекти	-	-	-	-
Управление версията на документи	-	Да, ако се намират в CVS	-	+

Глава 2. Технологични средства и обосновка

Системата за управление е разработена за обслужване на малки и средни фирми. Това предполага, че няма да е подложена на екстремни натоварвания. Изисква се също така да е с минимална цена и да е възможно бързото и лесно нагаждане с цел да се адаптира за бъдещите нужди на фирмата. Тези условия определят избора на платформа:

1. MySQL – СУБД [5]

   MySQL е многопотребителска SQL система за управление на база от данни. Според статистиката на MySQL AB съществуват над 11 милиона инсталации на MySQL [6]. Всеки развит език за програмиране предоставя библиотека за работа с MySQL.

   MySQL е популярна на вече като компонент от най-използваните в момента платформи за уеб приложения: LAMP, MAMP и WAMP (Linux/Mac/Windows-Apache-MySQL-PHP/Perl/Python). За администриране на MySQL може да се използва включеното приложение работещо от командния ред. Също така, свободно се разпространяват и графични среди за работа с MySQL като MySQL Administrator и MySQL Query Browser.

   MySQL работи на над 20 платформи: AIX, BSDi, FreeBSD, HP-UX, GNU/Linux, Mac OS X, NetBSD, OpenBSD, OS/2 Warp, QNX, Solaris, SunOS, SCO OpenServer, Tru64, Windows 95, Windows 98, Windows 2000, Windows ME, Windows NT, Windows XP и др.

   От версия 5.0, MySQL поддържа множество функции, които са и една от причините за широкото му използване. Сред тези характеристики са:

   - Съхранени процедури
   - Тригери
   - Указатели
   - Обновяеми изгледи
   - Транзакции с InnoDB, BDB и Cluster хранилища
   - Поддръжка на SSL
   - Кеширане на заявки
   - Вложени заявки
   - Пълна поддръжка на UNICODE

- Foreing key поддръжка за InnoDB

За разлика от други системи за управление на релативна база от данни, MySQL поддържа множество хранилища(InnoDB, BDB, Cluster, MyISAM и др), давайки възможност да се избере най-ефективното за всяка таблица в приложението.

Освен гореспоменатите преимущества на MySQL, за текущата дипломна работа бе избрана именно тази СУБД поради следните причини: [7]

- MySQL може да се инсталира безплатно [8]
- MySQL е лесен за инсталиране
- MySQL има изчистен и лесен за използване интерфейс (API – Application programming Interface)
- MySQL има свободно разпространена документация с добро качество. Освен това съществуват множество други източници като книги и online статии.

2. Ruby – скриптов, обектно-ориентиран език за програмиране с отворен код [9] [10] [11] [12] [13]. Сред характеристиките поради които Ruby е толкова популярен в момент са:
   - Чиста обектно-ориентирана концепция- всичко е обект, включително и класовете
   - Обектите обменят информация чрез съобщения
   - Интерпретативен стил на програмиране- Ruby не изисква компилация на кода, т.е. дава възможност за по бърза разработка
   - Динамично програмиране- типовете на променливите и изразите се определят по време на изпълнение
   - Интуитивен синтаксис, лесен за изучаване
   - Итерирането над колекции е вградено в езика
   - Автоматично прихващане на изключения (exception handling), което при необходимост може да бъде експлицитно зададено от разработчика

- Библиотеки- Ruby притежава богат набор от стандартни библиотеки(от клас библиотеки за работа със сложни типове като стрингове и колекции до библиотеки за работа мрежови протоколи и нишки)
- Интуитивно и лесно използване на reflection
- Оценяване на код по време на изпълнение на програмата, т.е. дава свобода на писане характерна за функционалните езици
- Пространства от имена (namespace)- Ruby притежава т.нар. module, който е namespace, може да има дефинирани методи, не може да се инстанцира и може да се дефинира в рамките на клас.
- Предефинирани оператори
- Преносимост- Ruby може да работи върху широк набор от платформи като Unix, Linux, DOS, Windows, OS/2
- Автоматично изчистване на паметта (garbage collection)

3. Ruby on Rails [14] [15] [16] [17]– уеб фреймуърк (framework) с отворен код, предназначен за бързо изграждане на приложения, които са типични графични фасади на бази данни. Основната концепция е базирана на вградения Model-View-Controller pattern. Ruby on Rails става все по-популярен поради следните причини:

   - Конвенции- Ruby работи с конвенции, а не с конфигурации. По този начин се намаляват до минимум необходимите конфигурации.
   - Разработката на уеб приложения с Ruby on Rails е по-бърза отколкото с други web frameworks
   - Ruby on Rails има вградена поддръжка за AJAX (Asynchronous JavaScript and XML)
   - Ruby on Rails изисква писане на по малко код от страна на разработчика, тъй като предоставя генериран код за основните CRUD операции (create, read, update and destroy). По този начин кода става по лесен за поддръжка и по-рядко дава дефекти. Освен

това увеличава производителността на разработчика като му спестява писането на конфигурации.

- Следването на Model-View-Controller pattern прави приложението лесно за поддръжка и с ясна архитектура.

Причини поради които разработчиците използват Ruby on Rails

- Ruby има лесен и естествен синтаксис
- Използва се един framework за цялото приложение и той предоставя всичко необходимо
- Бързо нараства броя на източници, тъй като нараства популярността
- Развиваща се общност от разработчици в цял свят
- По- малко код създава повече функционалност
- Не се налага разработчика да е експерт или да използва няколко технологии за да започне разработката
- Множество техники и възможности за напреднали разработци
- Изключва цикъла compilation-build-deploy

4. WebRick [18] – HTTP server, който е стандартна Ruby библиотека

Глава 3. Реализация

3.1 План за реализацията на проекта

Реализацията започва със създаване на празно Ruby on Rails приложение. По този начин се оформя първоначалната директорна структура на крайния продукт. Следва дизайн и реализация на базата от данни, както и тестване на ограничителните условия дефинирани в нея. След като базата бъде тествана, може да се премине към конфигуриране на Rails приложението да работи именно с тази база от данни, както и генериране на основните му модули въз основа на model-view-controller pattern. На този етап може да се заключи, че етапа на подготовка за разработка е приключен и е възможно да започне работата по изграждане на функционалност на приложението. Втория етап включва дефиниране на релациите между домейн (domain) обектите, т.е. разработва се и се допълва модела на приложението. Също така се взима решение за общия изглед на страниците, за изграждането на менютата (динамични и статични), детайлна разработка на бизнес логиката, както и валидация на данните. По долу са описани основните стъпки:

1. Създаване на празно Rails уеб приложение
2. Създаване на базата от данни
3. Въвеждане на тестови данни и тестване на ограничителните условия (constraints) в модела на базата данни
4. Конфигуриране на приложението за работа с базата данни
5. Генериране на скеле (scaffolding): базов модел, контролер и изглед (model-view-controller pattern) за всяка от таблиците в базата данни
6. Тестване на генерирания код и на конфигурацията
7. Определяне на връзките между елементите на модела с цел по- лесен достъп до данните на ниво контролер и изглед
8. Създаване на основно оформление (layout) на приложението
9. Създаване и конфигуриране на динамично меню
10. Изграждане на контролер и изгледи за работа с данните на потребители, екипи, проекти, задачи, документи.
11. Създаване на оформление за начална страница и страница за грешки
12. Валидация на данните от страна на сървъра

3.2 Описание на реализацията

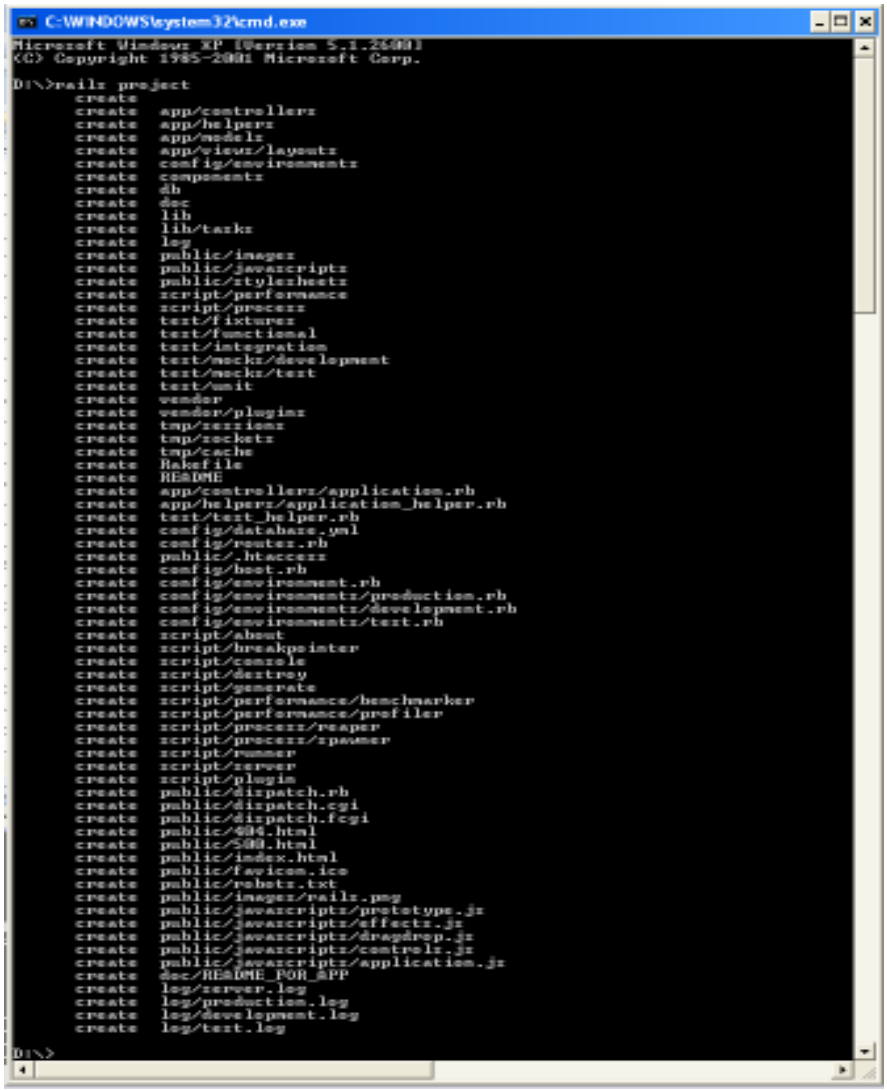
Изложението по-долу дава подробна информация за изпълнението на всяка точка от плана за реализация:

1. Създаване на празно Rails уеб приложение –

   Избираме директория, в която ще се намира приложението и я отваряме в команден ред и пускаме командата: rails project,

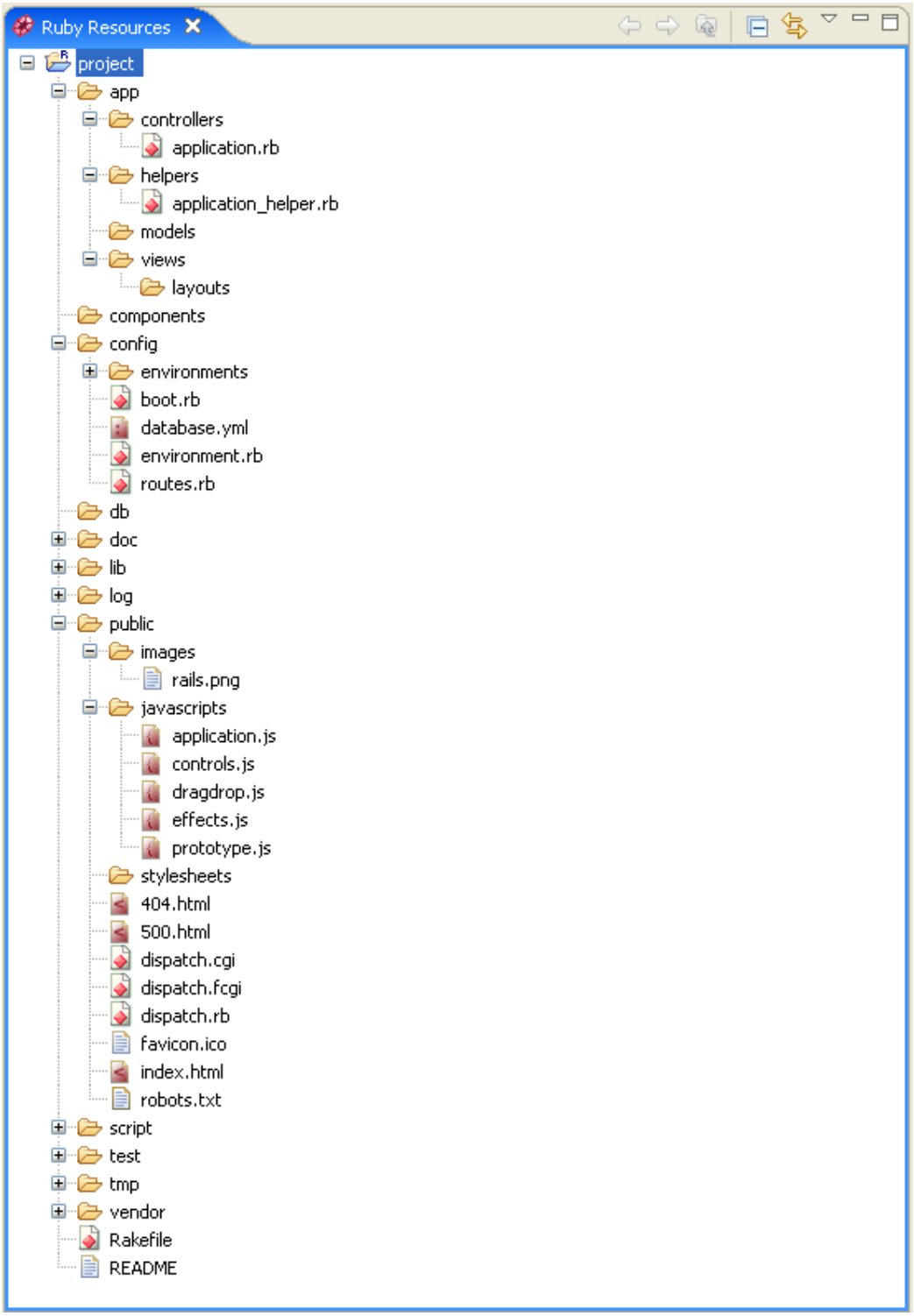
   където project е името на главната директория на приложението. Резултата е (фиг. 2):

Фиг. 3 Създаване на празно Rails приложение



Като резултат от командата се създава следната директорна структура (фиг. 3):

Фиг. 4 Директорна структура на Rails приложение



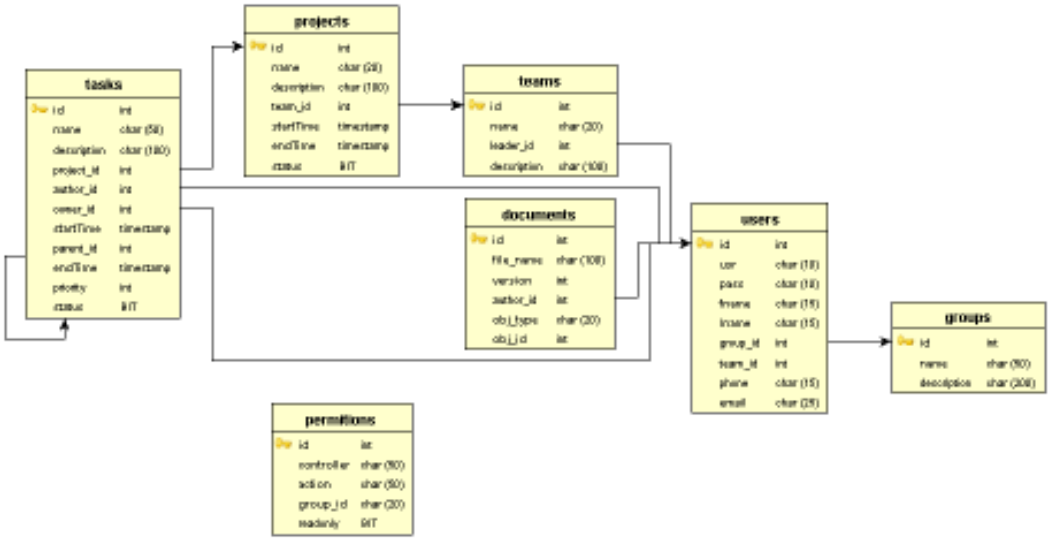
От най- голям интерес за положението са следните директории:

- app/controllers - В тази директория Rails търси класовете на контролерите (controller). Контролера прихваща уеб заявките (web request) на потребителя
- app/views – Съдържа изгледите (view) под формата на шаблони (templates), които се попълват с данни от приложението, контвертират се до HTML и се връщат до браузъра на потребителя (web response)
- app/models – Директорията съдържа класове (model), които обвиват (wrap) данните запазени в базата данни, с която работи приложението
- app/helpers – Съдържа всякакви помощни класове, които се използват в model и controller класовете, както и в view шаблоните

2. Създаване на базата от данни

   Схемата на базата от данни има следния изглед:

Фиг. 5 Структура на базата от данни



Скриптът за създаване на базата от данни е:

Фиг. 6 SQL скрипт за създаване на базата от данни

```
create database project_management;

use project_management;


create table groups
(
      id int default null auto_increment,
      name char(50) not null unique,
      description char(200) default null,
      primary key (id)
)engine=innodb;

create table permitions
(
      id int default null auto_increment,
      primary key (id),
      controller char(50) not null,
      action char(50) not null,
      group_id char(20) not null,
      readonly TINYINT(1) default false
)engine=innodb;

create table users
(
      id int default null auto_increment,
      primary key (id),
      usr char(10) not null unique,
      pass char(10) not null,
      fname char(15) not null,
      lname char(15) not null,
      group_id int,
      index ind_group_id (group_id),
      foreign key (group_id) references groups(id)
                  on delete no action,
      team_id int default null,
      phone char(15),
      email char(25)
)engine=innodb;

create table teams
(
      id int default null auto_increment,
      primary key (id),
      name char(20) not null unique,
      leader_id int,
      index ind_leader_id (leader_id),
      foreign key (leader_id) references users (id)
                   on delete no action,
      description char(100) default ""
)engine=innodb;

create table projects
(
      id int default null auto_increment,
      primary key (id),
```

```
      name char(20) not null unique,
      description char(100) default "",
      team_id int,
      index ind_team_id (team_id),
      foreign key (team_id) references teams (id) on delete no action,
      startTime timestamp,
      endTime timestamp,
      status TINYINT(1) default false
)engine=innodb;



create table documents
(
      id int default null auto_increment,
      primary key (id),
      file_name char(100) not null,
      version int(10) not null,
      author_id int,
      index ind_author_id (author_id),
      foreign key (author_id) references users (id)
                  on delete no action,
      obj_type char(20) not null,
      obj_id int not null
)engine=innodb;


create table tasks
(
      id int default null auto_increment,
      primary key (id),
      name char(50) not null unique,
      description char(100) default "",
      project_id int,
      index ind_project_id (project_id),
      foreign key (project_id) references projects (id)
                  on delete cascade,
      author_id int,
      index ind_author_id (author_id),
      foreign key (author_id) references users (id)
                  on delete no action,
      owner_id int,
      index ind_owner_id (owner_id),
      foreign key (owner_id) references users (id)
                  on delete no action,
      startTime timestamp,
      parent_id int,
      index ind_parent_id (parent_id),
      foreign key (parent_id) references tasks (id) on delete cascade,
      endTime timestamp,
      priority int(1) default 1,
      status TINYINT(1) default false
)engine=innodb;
```

3. Въвеждане на тестови данни и тестване на ограничителните условия (constraints) в модела на базата данни

4. Конфигуриране на приложението за работа с базата данни

   Инсталиране на драйвър за MySql s командата:

   gem install –remote mysql

   Редактираме файла project\config\ database.yml:

   development:

   adapter: mysql

   database: project_management

   username: root

   password: ***** (парола за MySql)

   host: localhost

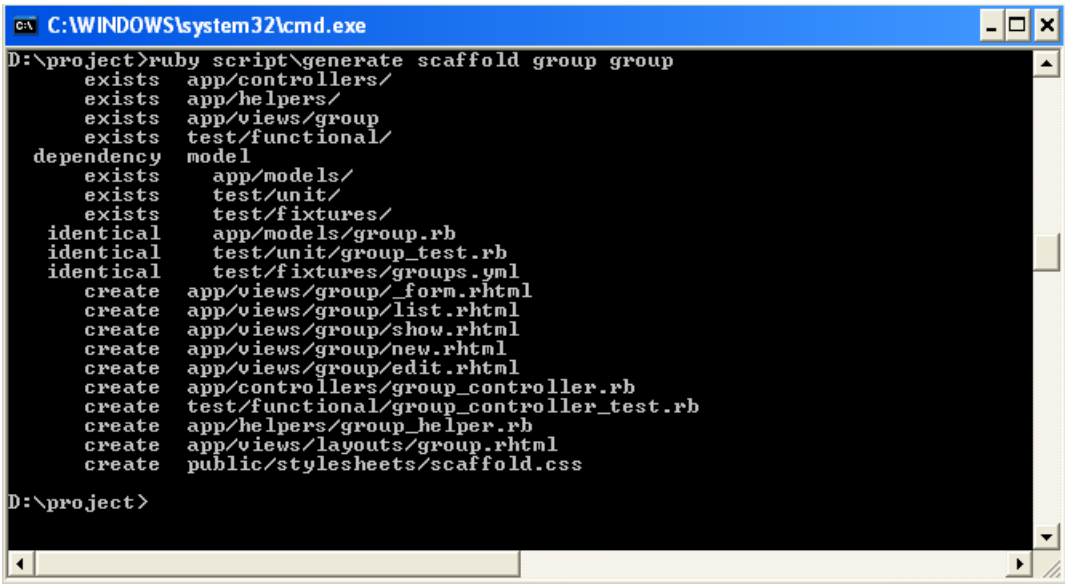
5. Генериране на скеле (scaffolding): базов модел, контролер и изглед (model-view-controller pattern) за всека от таблиците в базата данни

   В главната директория на проекта изпълняваме командата

   ruby script\generate scaffold [име_на_контролер] [име_на_модел]

   Например (фиг. 6):

Фиг. 7 Генериране на scaffolding



Основния резултата от генерацията е появата на:

app\controllers\group_controller.rb

app\helpers\group_helper.rb

app\cmodels\group.rb

app\views\group\ _form.rhtml

app\views\group\ edit.rhtml

app\views\group\ list.rhtml

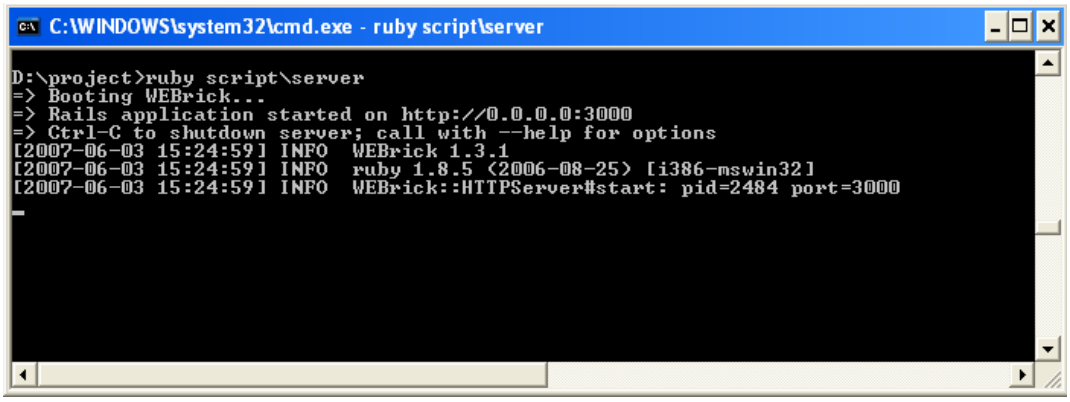
app\views\group\ new.rhtml

app\views\group\ show.rhtml

6. Тестване на генерирания код и на конфигурацията

С командата ruby script\server се стартира WebRick (фиг. 7)

Фиг. 8 Стартитране на WebRick



След това е възможен HTTP достъп до всеки от генерираните изгледи http://localhost:3000/group/list

7. Определяне на връзките между елементите на модела с цел по- лесен достъп до данните на ниво контролер и изглед

   Отношенията между елементите на модела могат да бъдат описани със средствата на Ruby on Rails. Например в модела на User се посочва, че при извличане на user от базата данни, трябва да се намерят и данните на групата му. Ruby on Rails генерира заявка, която търси групата въз основа на вторичния ключ group_id [19]

```
class User < ActiveRecord::Base

   belongs_to :group

   ..............
end
```

8. Създаване на основно оформление (layout) на приложението: Ruby on Rails предоставя лесен начин за дефиниране на layout на приложението като цяло, както и layout за отделни изгледи. Например

```
class ApplicationController < ActionController::Base
    layout "base"
..............
end
```

определя, че всяка страница в приложението ще бъде декорирана посредством дефинирано оформление във файл app\views\layouts\base.rhtml

9. Създаване и конфигуриране на динамично меню: Динамичното меню зависи от контролера и групата на потребителя, който прави request. Дефиницията на менюто се описва в xml файлове. Това прави неговата промяна лесна и бърза, т.е. това са действия които администратора на приложението може да направи без да се налага да е запознат с езика. За даден контролер има поне един файл с дефиниция на менюто му. Конвенцията за именуване на xml файловете е следната:
   [име_на_контролер].xml
   [име_на_контролер]_[име_на_потребителска_група].xml
   При прихващане на request, компонента отговорен за изграждането на менюто търси в директорията app\views\main\menu_config файл с име [име_на_контролер]_[име_на_потребителска_група].xml

Ако не открие такъв, се зарежда подразбиращото се меню за дадения контролер. Неговата дефиниция е в [име_на_контролер].xml

10. Изграждане на контролер и изгледи за работа с данните на потребители, екипи, проекти, задачи, документи (виж подробната документация по долу)
    Създаване на оформление за начална страница и страница за грешки: Страницата за вход в приложението (login page) и страницата за грешки се (error page) имат оформление различно от основното. То е дефинирано в app\views\layouts\nologin.rhtml, а регистрирането му е в

```
class MainController < ApplicationController
    ........
    layout "nologin", :only => ["login", "error"]
    ............
end
```

11. Валидация на данните от страна на сървъра: Функциите за валидация се дефинират в класовете на модела

Глава 4. Техническа реализация

4.1 Контролери

4.1.1. ApplicationController

Филтрите добавени в този контролер се изпълняват преди всеки метод на останалите контролери с изключение на експлицитно посочените. Също така всички методи добавени в ApplicationController са видими в останалите контролери на приложението.

Ключови елементи:

- `layout "base", :except => ["menu", "login", "doLogin", "todoList", "calendar"]`

  В контролера е дефинирано оформление (layout) “base”, което се използва за всички потребителски заявки с изключение на посочените

- `before_filter :authorize, :except => ["login", "error", "doLogin"]`

  Метода authorize се изпълнява преди всеки потребителски request, с посочените изключения. Метода проверява дали потребителя се е идентифицирал с потребителски име и парола. В случай на отрицателен резултат потребителя се препраща към login.rhtml

- `before_filter :permitions, :except => ["login", "error", "doLogin", "logout"]`

  Метода permitions се изпълнява преди всеки потребителски request, с посочените изключения. Проверява дали потребителя има права да достъпи данните. Правата на потребителите се въвеждат в таблицата „permitions”. При достигане на request до филтъра първо се търси запис съдържащ името на контролера (controller), неговия метод (action) и името на потребителската група, т.е. комбинация controller-action-group. Ако такъв запис не бъде открит се търси запис със данните за controller, action и група „*”. Ако и такъв не бъде открит се търси комбинация controller-*-*. Ако потребителя няма дефинирани права неговия request се препраща към

страницата за грешки. В правата на потребителя се задава и дали има право само за четене или за четене и писане.

- `def rescue_action_in_public(exception)` – Метода е отговорен за прихващането грешки от работата на контролерите. В случай на такава грешка потребителя се препраща към страницата за грешки където получава кратко съобщение за възникналия проблем.

4.1.2. MainController

MainController е отговорен за общите действия: login и logout, както и за изграждането на общите за цялото приложение компоненти. Тези компоненти са динамичното меню и календара. Ключови елементи:

- `def doLogin` – Чрез http request на потребителя, идващ от login.rhtml, до контролера достигат потребителското име и парола, които идентифицират уникално потребителя. Ако такъв потребител бъде открит в базата от данни, той ще бъде пренасочен към страницата с личните му данни и в потребителската му сесия ще бъдат записани основните му данни. Именно чрез тези данни филтъра authorize в ApplicationController разпределя оторизираните от неоторизираните заявки. Ако потребител със даденото име и парола не открит в базата, то неговия request ще бъде пренасочен към страницата за login със съобщение, че трябва да се идентифицира.
- `def logout` – инвалидира потребителската сесия и препраща потребителя към login страницата
- `def menu` – Метода е отговорен за моделирането на динамичното меню. То зависи от контролера и групата на потребителя, който прави request. Дефиницията на менюто се описва в xml файлове. Ако за даден контролер е предвидено динамично меню, то съществува поне един файл с дефиниция. Конвенцията за именуване на xml файловете е следната:

[име_на_контролер].xml

[име_на_контролер]_[име_на_потребителска_група].xml

При прихващане на request, компонента отговорен за изграждането на менюто търси в директорията app\views\main\menu_config файл с име [име_на_контролер]_[име_на_потребителска_група].xml

Ако не открие такъв, се зарежда подразбиращото се меню за дадения контролер. Неговата дефиниция е в [име_на_контролер].xml

Пример: task.xml (фиг. 8)

Фиг. 9 Дефиниция на меню

```
<list>
   <group id="Actions">
      <item id="2">
         <value>My tasks</value>
         <controller>task</controller>
         <action>list</action>
         <param>session</param>
      </item>
      <item id="3">
         <value>All Tasks</value>
         <controller>task</controller>
         <action>list</action>
      </item>
      <item id="5">
         <value>Search</value>
         <controller>task</controller>
         <action>search</action>
      </item>
   </group>
   <group id="Lists">
      <dinamic>true</dinamic>
      <controller>task</controller>
      <action>todoList</action>
   </group>
</list>
```

Разглеждания пример е дефиниция на динамично меню в 2 части. В първата група (Actions) попадат фиксирани връзки към определени страници в приложението, които изискват обръщение към въпросния

контролер. Всеки “item” е една връзка (link). Връзката се определя от името на контролера, метод в този контролер (action) и като опция може да се предаде параметър. В посочения пример като “param” е посочен “session”, който се интерпретира като текущо логнатия потребител. Втората група е определена като “`dinamic`”. Това прехвърля изграждането на остатъка от менюто в посочения контролер и метод. От там нататък, той изгражда изгледа на тази част от менюто.

- `def calendar -` метода е отговорен за изграждането на календар, който е видим в страниците за редакция, в които има възможност за въвеждане на дати. Ако бъде подаден параметър “date”, метода ще състави календар за месеца в който е подадената дата. Ако освен “date”, е подаден и параметър “shift”, то към месеца ще бъде добавен или изваден един месец , съответно за стойностите +1 и –1. В случай, че параметри не бъдат подадени, изчисленията ще бъдат направени въз основа на текущата дата.
  Извикването на метода се извършва посредством Ajax request.

4.1.3 AccountController

AccountController контролира само и единствено данните свързани с текущия потребител

- `def detail-` подготвя данните на логнатия потребител във форма подходяща за показването им в съответното view

- `def updatePassword-` формата за смяна на парола на текущия потребител препраща данните до този метод на контролера. В метода се прави проверка за валидност на старата парола като тя се сравнява с тази в базата данни. Ако съвпаднат, се обновява записа на потребителя и новите данни се отразяват в сесията му. В противен случай потребителския request се препраща на страницата с грешки с подходящо съобщение.

- `def updateUser`- метода се използва когато потребителя редактира разрешените данни, а именно телефон и емайл адрес

4.1.4 UserController

`UserController` манипулира данните на потребителите в системата. Отговорен е за изграждането на всички изгледи в секцията “Users”. Контролера предоставя както общодостъпните данни, така и тези, за които права има само администратора. Всеки потребител принадлежи към потребителска група. Потребителя може да е причислен към екип, но това не е задължително. Например администратора няма екип.

- `auto_complete_for :user, :usr` – автоматично допълване (auto complete) за потребителското име. Използва се във формата за регистриране на потребител от администратора

- `auto_complete_for :group, :name` - auto complete за името на потребителските групи. Използва се във формата за регистриране на потребител от администратора, както и за редактиране на потребителска информация.

- `auto_complete_for :team, :name` - auto complete за името на екип. Използва се във формата за регистриране на потребител от администратора, както и за редактиране на потребителска информация.

  `def list` - Функцията е отговорна за извличането на данните на потребителите, които се предоставят в табличен вид от съответния изглед (фиг. 9). Ако потребителския request е стандартен, то изгледа е app/views/user/list.rhtml. Ако потребителския request е ajax, то единствената част която се обновява в отговор на запитването е app/views/user/_list.rhtml. Причина за това е факта че както и останалите таблици в приложението и тази предоставя възможност за сортиране. Именно заявката за сортиране стига до контролера

като Ajax request. Сортирането може да стане по потребителско име, първо или последно име на потребителя, име на група или име на екип.

Фиг. 10 Имплементация на метод за търсене и сортиране

```
    def list

      sort = case params['column']
      when "usr"  then "usr #{params['order']}"
      when "fname"   then "fname #{params['order']}"
      when "lname"   then "lname #{@params['order']}"
      end
      @users = User.find(:all, :order => sort)

      if params['column'] == 'groupName'
          if params[:order] == 'DESC'
              @users = @users.sort do |a,b| b.group.name <=>
                                        a.group.name
          end
          end
          if params[:order] == 'ASC'
              @users = @users.sort do |a,b| a.group.name <=>
                                        b.group.name
          end
          end
      end
      if params['column'] == 'teamName'
          if params[:order] == 'DESC'
              @users = @users.sort do |a,b| b.team.name <=>
                                        a.team.name
          end
          end
          if params[:order] == 'ASC'
              @users = @users.sort do |a,b| a.team.name <=>
                                        b.team.name
          end
          end
      end

      if request.xml_http_request?
          render :partial => "list", :layout => false
      end
  end
```

- `def show` – Функцията е отговорна за извличането на данните на потребител с подаденото id. Ако такова няма, то се инициализира нов обект от тип User. По този начин се използва един и същ изглед за дефиниране на нов потребител и за редактиране на съществуващ. Ако

потребителя, който прави request не е администратор, той ще получи read-only достъп до изгледа.

- `def save` – От формата за редактиране, потребителския request достига до функцията save, която запазва данните в базата данни. Преди това се извършва валидация на данните. Ако не са открити грешки, потребителя се препраща към изгледа със списъка. В противен случай се връща на формата със съответно съобщение за откритата грешка

- `def delete – Изтрива от системата потребител с подаденото id и препраща потребителя към списъка.`

- `def update_group` – Функцията се използва в един конкретен случай – при създаване на екип или редактиране на данните на екипа, може да бъде определен ръководител на екипа от списъка на потребители, които не са такива. Това изисква обновяване само на групата на избрания потребител.

4.1.5 ProjectController

`ProjectController` контролера управлява данните на проектите, както и свързаната с тях информация. Всеки проект принадлежи на екип. Проекта има множество задачи. Проект може да бъде създаден, редактиран или изтрит от администратора, ръководител на екип и ръководител на проект. Всички останали имат права само за четене на данните.

- `auto_complete_for :team, :name -` auto complete за името на екип. Използва се във формата за създаване и редактиране на информацията на проект

- `auto_complete_for_user_usr` – функцията прави auto complete по първо и последно име на регистриран потребител. Поради факта, че са необходими две полета от таблицата на потребителите, не е

възможно да се използва стандартния за Ruby on Rails макрос. Ето защо тази функционалност е пренаписана в контролера. (фиг. 10)

Фиг. 11 Пример за пренаписване на базова функциаоналност

```
def auto_complete_for_user_usr
      @users = User.find_by_sql "select id,
                    concat(fname,' ',lname) as usr
                    from users
                    where LOWER(fname) LIKE
                    '%#{@params[:user][:usr]}%'
                    or LOWER(lname) LIKE
                    '%#{@params[:user][:usr]}%'
                    order by fname asc"

       render :inline =>
              "<%= auto_complete_result(@users, 'usr')%>"
end
```

- `projectForTeam`- функцията намира всички проекти, които принадлежат на екип с подаденото id. Ако потребителския request е бил Ajax request, като отговор се обновява само онази част от страницата, за която е отговорен app/views/project/_list.rhtml. В противен случай не се дава отговор. За селекцията от базата данни се използва вътрешната помощна функция find

- `list` – намира всички проекти в базата от данни. За целта използва вътрешната помощна функция find. Ако потребителския request е стандартен, то изгледа е app/views/project/list.rhtml. Ако потребителския request е бил Ajax request, като отговор се обновява само онази част от страницата, за която е отговорен app/views/project/_list.rhtml. Това се случва при сортиране на резултатите от потребителя.

- `find`- Функцията е отговорна за извличането на данните на проектите. Това е вътрешна функция, която няма съответстващ изглед. Тя се извиква от методите list и projectForTeam. Предоставя функционалност за сортиране на резултатите. Сортирането може да

стане по име на проект, време на стартиране на проект, статус на проект или име на екип.

- `show-` Функцията е отговорна за извличането на данните на проект с подаденото id. Ако такова няма, то се инициализира нов обект от тип Project. По този начин се използва един и същ изглед за дефиниране на нов проект и за редактиране на съществуващ. Ако потребителя, който прави request не е администратор, ръководител на проект или ръководител на екип, той ще получи read-only достъп до изгледа.

- `save` – От формата за редактиране, потребителския request достига до функцията save, която запазва данните в базата данни. Преди това се извършва валидация на данните. Ако не са открити грешки, потребителя се препраща към изгледа със списъка на проектите. В противен случай се връща на формата със съответно съобщение за откритата грешка. За обработката на датите, т.е. преобразуването им от стринг във формат timestamp, с който работи базата от данни, се използва помощна функция dateDBFormat.

- `update-` Функцията отговоря само на Ajax request за промяна на статуса на проект. Като параметри от формата до контролера достигат id на проекта и новия статус. Ако статуса е “open”, то endTime на проекта се записва като nil, а статуса като false. В противен случай, т.е. при затваряне на проект, за endTime се записва текущата дата, а за статуса- true. Същото се повтаря за всички и за всички задачи и подзадачи принадлежащи на този проект. Поради факта, че това действие се извършва директно от изгледа показващ списъка с проекти, то се обновява само input полето, върху което е цъкнал потребителя. (виж общо инфо за render-text). За тази цел контролера връща само подадените параметри чрез `render_text`.

4.1.6 TaskController

Контролера управлява данните на задачите и подзадачите. Всяка задача принадлежи на единствен проект. Задачата има автор, както и притежател (owner). Има също така начална дата и крайна дата, ако вече е приключила, т.е. ако статуса й е “close”. Особеност има при подзадачите. Те са собственост само и единствено на своя автор. Друг потребител няма достъп до тях нито за писане нито за четене. Подзадачите остават в личното пространство на автора си.

- `auto_complete_for :task, :name -` auto complete за името на задача. Използва се във формата за създаване и редактиране на информацията на задача.

- `auto_complete_for :project, :name -` auto complete за името на проект. Използва се във формата за създаване и редактиране на информацията на задача

- `auto_complete_for_user_usr –` функцията прави auto complete по първо и последно име на регистриран потребител. Поради факта, че са необходими две полета от таблицата на потребителите, не е възможно да се използва стандартния за Ruby on Rails макрос. Ето защо тази функционалност е пренаписана в контролера.

- `new`- Инициализира нов обект от тип Task. По подразбиране записва в него като автор и притежател текущо логнатия потребител. За старт дата поставя текущата дата. Ако сред подадените параметри има “task_id”, то задачата се причислява към проекта с даденото id. Това се случва когато действието за създаване на задача е инициирано от изгледа за детайли на проект. В резултат от изпълнението на метода се визуализира изгледа app/views/task/show.rhtml.

- `save` – От формата за редактиране, потребителския request достига до функцията save, която запазва данните в базата данни. Преди това се извършва валидация на данните. Ако не са открити грешки, потребителя се препраща към изгледа от който е бил иницииран потребителския request за създаване или редактиране на задача. В противен случай се връща на формата със съответно съобщение за откритата грешка. Автора на задача не може да се редактира. Притежателя на задача се открива по имената му, които идват от формата. Същото важи и за проекта към който се причислява задачата. Поради факта, че същата функция обработва създаването и обновяването и на задачи и на подзадачи, се проверява дали сред подадените параметри има "parent_id". За обработката на датите, т.е. преобразуването им от стринг във формат timestamp, с който работи базата от данни, се използва помощна функция dateDBFormat.

- `list` – намира всички задачи в базата от данни. За целта използва вътрешната помощна функция find. Ако потребителския request е стандартен, то изгледа е app/views/task/list.rhtml. Ако потребителския request е бил Ajax request, като отговор се обновява само онази част от страницата, за която е отговорен app/views/task/_list.rhtml. Това се случва при сортиране на резултатите от потребителя.
- `taskForProject`- функцията намира всички задачи, които принадлежат на проект с подаденото id и не са подзадачи. Ако потребителския request е бил Ajax request, като отговор се обновява само онази част от страницата, за която е отговорен app/views/task/_list.rhtml. В противен случай не се дава отговор. За селекцията от базата данни се използва вътрешната помощна функция find

- `find`- Функцията е отговорна за извличането на данните на задачите. Това е вътрешна функция, която няма съответсващ изглед. Тя се извиква от методите list и taskForProject. Предоставя

функционалност за сортиране на резултатите. Сортирането може да стане по име на задача, време на стартиране на задача, статус на задача, време на приключване на задача, име на автор, име на притежател или име на проект.

- `show-` Функцията е отговорна за извличането на данните на задача с подаденото id. Ако такова няма, то се инициализира нов обект от тип Task. По този начин се използва един и същ изглед за дефиниране на нова задача и за редактиране на съществуваща. Ако потребителския request е Ajax request, то като резултат ще се обнови само онази част от страницата, която е дефинирана в app/views/task/_show.rhtml. В противен случай ще се зареди страницата app/views/task/show.rhtml

- `newTodo`- Инициализира нов обект от тип Task, който функционира като подзадача. Автор и притежател на подзадачата е текущо логнатия потребител. Сред подадените параметри е "parent_id". Това е id на задачата към, която се причислява новата (или редактираната) подзадача. В резултат от изпълнението на метода се визуализира изгледа app/views/task/newTodo.rhtml.

- `showTodo-` Функцията извлича от базата данни всички подзадачи на задача с даденото id. В условията на заявката е включено още id на текущо логнатия потребител като притежател на подзадачите. Използва се помощната функция find, т.е. таблицата с резултатите подлежи на сортировка. Също така се изчислява какъв процент от подзадачите са завършени. Ако потребителя е направил Ajax request, то като резултат ще се обнови само онази част от страницата, която е дефинирана в app/views/task/_showTodo.rhtml. В противен случай ще се зареди страницата app/views/task/showTodo.rhtml

- `todoList`- Функцията връща списък на задачите, чиито собственик е текущо логнатия потребител. Този списък се използва за изграждане на част от менюто.

- `delete`- Функцията изтрива задача по подаденото id. Ако задачата има подзадачи, те също ще бъдат изтрити.

- `update`- Функцията отговоря само на Ajax request за промяна на статуса на задача. Като параметри от формата до контролера достигат id на задачата и новия статус. Ако статуса е "open", то endTime на задачата се записва като nil, а статуса като false. В противен случай, т.е. при затваряне на задача, за endTime се записва текущата дата, а за статуса- true. Поради факта, че това действие се извършва директно от изгледа показващ списъка със задачи, то се обновява само input полето, върху което е цъкнал потребителя. (виж общо инфо за render-text). За тази цел контролера връща само подадените параметри чрез `render_text`.

4.1.7 DocumentController

Контролера управлява данните на документите. Документите не съществуват самостоятелно. Те са прикачени към проект или задача. Поддържа се следене на версията за всеки документ.

- Потребителския request достига до метода от формата за регистриране на нов документ (upload). Функцията създава нов обект от тип Document. В него като автор на документа се записва id на текущо логнатия потребител. Отбелязват се също и типа на обекта, към който се прикачва документа (например project или task) и id на този обект. След това се проверява за съществуващ документ със същото име и за същия обект. Ако такъв бъде намерен, то версията на новия се увеличава с единица. В противен случай документа се записва с версия 0. Физически документа се запазва в

[root_dir]\files\[obj_type]\[obj_id]\[file_name]_[version].[file_ext]

където:

- [root_dir] е главната директория на сървъра
- [obj_type] е типа на обекта, към който е прикачен документа
- [obj_ id] е id на обекта
- [file_name] е оригиналното име на файла
- [version] е пресметната версия
- [file_ext] е оригиналното разширение на файла

При успешен upload и валидация на данните на документа, потребителя се връща на страницата от която е направил upload. При открита грешка се презарежда страницата за upload.

За извличане на истинското име на файла се използва помощтната функция `sanitize_filename`.

- `sanitize_filename`- Функцията първо извлича името на файла, тъй като от някои браузъри то може да дойде с целия си път. След това заменя всички символи, които не са букви и цифри с долна черта. Получения стринг се връща като резултат от функцията.
- `download`- Функцията е отговорна за свалянето на файл от сървъра на клиентската машина. Като параметър се подава id на документа. По него се извлича запис от базата данни съответстващ на този документ. По информацията в записа се открива физическия път до файла, чрез помощната функция `find_file` и се прави проверка за съществуване. Ако файла не съществува потребителя се препраща на страницата със списъка на документи. В противен случай файла се изпраща към клиентския браузър за download.
- `find_file`- Помощна функция, която се използва за намиране на физическия път до файл регистриран в системата преди това. На метода се подават като параметри версията и път, който е сформиран по следната схема: [root_dir]\files\[obj_type]\[obj_id]. В тялото на метода се пресмята пълния път във вида:

[root_dir]\files\[obj_type]\[obj_id]\[file_name]_[version].[file_ext].

След това се проверява дали файла съществува и ако да, дали е файл и дали може да бъде прочетен (read права).

- `list-` Функцията е отговорна за извличането на данните на документите регистрирани в базата данни. Ако сред параметрите подадени към метода има id, то търсеното протича по следната схема:
  - Извлича се запис на документ с даденото id
  - Извличат се всички документи с име на документ, тип на обект и id на обект съвпадащи с тези на гореспоменатия. Резултатите са сортирани по намаляващо по версията.

  Така протича търсенето ако потребителя е поискал да прегледа всички версии на даден документ.

  Ако сред параметрите на метода не е подадено id, търсеното се основава на тип на обект и id на обект, при което се извлича само последната версия. Функцията предоставя възможност за сортиране на резултатите по име на файл или по име на автор. Ако потребителя е направил Ajax request, ще се обновява само онази част от страницата, за която е отговорен app/views/document/_list.rhtml. В противен случай, при стандартен request, се изпълнява app/views/document/list.rhtml

- `delete-` Функцията получава като параметър id на документ.

  Извлича от базата записа съответстващ на този документ. Използва данните на записа за да открие физически файла, чрез функцията find_file. Ако файла съществува, той бива изтрит. В противен случай функцията връща грешка към изгледа със списъка на файловете и приключва изпълнението си. Ако файла е изтрит успешно следва триене на записа от базата данни. Така в системата остават да съществуват предишните версии на документа, ако такива съществуват. Като резултат от функцията се обновява изгледа със списъка на документи.

4.1.8 TeamController

Контролера управлява данните на екипите. Всеки екип има име, което е уникално в системата. Има единствен ръководител, като ролята на този потребител е “team_leader”. Може да има нула или повече членове. Всеки потребител може да е причислен към не повече от един екип. Екипа е притежател на проектите си.

- `auto_complete_for_user_usr` – функцията прави auto complete по първо и последно име на регистриран потребител, чиято група е различна от “team_leader”. Поради факта, че са необходими две полета от таблицата на потребителите и има ограничително условие за потребителската група, не е възможно да се използва стандартния за Ruby on Rails макрос. Ето защо тази функционалност е пренаписана в контролера.
- `list` – намира всички екипи в базата от данни. Ако потребителския request е стандартен, то изгледа е app/views/team/list.rhtml. Ако потребителския request е бил Ajax request, като отговор се обновява само онази част от страницата, за която е отговорен app/views/team/_list.rhtml. Това се случва при сортиране на резултатите от потребителя. Сортирането може да е по име на проект или по име на ръководител на проект.

- `new`- Ако сред подадените към функцията параметри има id, то функцията извлича запис на екип с дадения идентификатор. В противен случай инициализира нов обект от тип Team. Също така селектира потребителите с група различна от “team_leader”. Последното се прави поради факта, че само тези потребители могат да бъдат членове на екип. В резултат от изпълнението на метода се визуализира изгледа app/views/team/new.rhtml. До този изглед имат достъп потребители с група администратор.

- `save` – От формата за редактиране, потребителския request достига до функцията save, която запазва данните в базата данни. Преди

това се извършва валидация на данните. Ако не са открити грешки, данните на екипа се запазват. След това се прави обновяване на данните на потребителите. Това включва:

- Групата на потребител избран за ръководител на екип се променя на “team_leader”.
- От формата, като параметри, идват два списъка. Първия е на “изтритите от екипа” потребители. За тях се обновява полето “team_id”, като в него се записва недефинирана стойност- nil. Във втория списък са идентификаторите на “добавените към екипа” потребители. За тях също се обновява полето “team_id”, като в него се записва id на екипа. Съставянето на тези два списъка с идентификатори се прави с JavaScript в изгледа app/views/team/new.rhtml.

Ако няма открити грешки при изпълнение на функцията потребителя се препраща на изгледа със списъка на екипите app/views/team/лист.rhtml. В противен случай се връща на формата за редактиране/ създаване на екип със съответното съобщение за това каква грешка е била открита.

- `Show`- Функцията е отговорна за извличането на данните на екип с подаденото id, а след това и открива запис на потребителя, който е определен като ръководел на този проект. Намира и потребителите, които са членове на екипа. Ако потребителския request е Ajax request, то като резултат ще се обнови само онази част от страницата, която е дефинирана в app/views/team/_show.rhtml. В противен случай ще се зареди страницата app/views/team/show.rhtml. Изгледа е read-only.
- `delete`- Функцията изтрива от базата данни екип с подаденото id. След това намира записите на потребители причислени към този екип и обновява полето “team_id” с недефинирана стойност- nil.

4.2 Допълнителни функции

Допълнителните функции, които са помощтни за контролери и изгледи се намират в helper класовете в директорията app/helpers.

- ApplicationHelper- функциите дефинирани в този клас са достъпни в цялото приложение
  - `dateViewFormat(date)`- Подадения обект от тип Date се форматира в подходящ за изгледите формат. Дефинирания формат е глобална променлива за ApplicationHelper- ден/месец/година. За форматирането се използва стандарт на Ruby функция strftime. (фиг. 11)

Фиг. 12 Форматиране на дата във формат подходящ за изглед

```
def dateViewFormat(date)
      if(date == nil)
            return ""
       end
       date.strftime(VIEW_FORMATT[0])
end
```

  - `dateDBFormat(dateString)`- Използва се в контролерите, за преобразуване на подаден стринг в обект от тип Date. Стрингът идва от потребителския request. Именно той съответства на използвания от базата данни timestamp тип. Метода прави обръщение към стандартна Ruby функция- strptime (фиг.12)

Фиг. 13 Инициализиране на обект от тип дата въз основа на стринг

```
def dateDBFormat(dateString)
    Date.strptime(dateString, VIEW_FORMATT[0])
end
```

  - `def link_to_order(text, column, order, controller)`- Функцията се използва в таблиците с данни, като формира връзки(links) за сортиране съдържанието на таблицата посредством Ajax request. `Като параметри се подават:`

- text- текст на връзката, който потребителя вижда на страницата
- column- името на колоната в таблицата, по която се прави сортиране
- order- ред на сортиране- възходящ или низходящ
- controller- име на контролер, към който се прави Ajax request. Метода (action) в този контролер трябва да е с име list.

За формиране на Ajax link, се използва стандартна за Ruby on Rails функция- link_to_remote (фиг. 13)

Фиг. 14 Помощтна функция за изграждане на Ajax links за сортиране на таблици

```
def link_to_order(text, column, order, controller)
  options = {
    :url => {:action => @params[:action],
             :controller => controller,
             :params =>   @params.merge(
                            {:column => column,
                             :order => order
                            }
                           )
            },
             :update => 'item_table'
           }
  html_options = {
     :title => "Sort by this field",
     :href => url_for(:action => 'list',
                      :controller => controller)
                }
  link_to_remote(
     image_tag(text,
       :class => "arrow_class_#{order}"),
     options,
     html_options)
end
```

4.3. Стандартни функции

- `auto_complete_for (object, method, options = {})` – Макросът auto_complete_for е стандартен за Rails. Той прави автоматично допълване (auto complete), като object е името на обекта, а method е метода по който се прави търсене. По подразбиране auto_complete_for сортира резултатите по дадения параметър, но той може да бъде зададен експлицитно. В резултат

макроса генерира функция с име auto_complete_for_[object]_[method], който изпълнява Ajax request.

Пример:

В контролера се декларира:

```
auto_complete_for :user, :usr
```

Това означава, че търсеното ще е "User", по полето „usr"

В изгледа:

```
<%= text_field_with_auto_complete :user, :usr %>
```

- **render**(options = nil, deprecated_status = nil, &block) – Функцията е стандартна за Ruby on Rails. Интерпретира подаденото й съдържание, като го преобразува до чист html, който връща към клиентския браузър. Може да работи по няколко основни начина:
  - Интерпретиране на действие- случай, в който се подава само action е най- често използвания. Използва се автоматично от ActionController когато не са подадени други параметри. Предизвиква изпълнение на метод с името на подадения action и показване на определения от него изглед.

    Пример: в TeamController – render `:action => "new"`

    Ako не се изисква използване на текущия layout, може да се подаде параметър :layout => false или директно да се подаде името на необходимия layout.
  - Интерпретиране на част от страница (render partial)- Използва се обикновено при Ajax request за обновяване само на един или няколко елемента от страницата без да се предизвиква презареждане. По подразбиране текущия layout не се използва.

    Пример: в UserController- `render :partial => "list"`

    Така се презарежда само часта в app/views/user/_list.rhtml
  - Интерпретиране на шаблон (render template)- работи подобно на интерпретирането на action, с тази разлика, че приема релативен път от главната директория на изгледите

    Пример: в AccountController-

```
render :template => "main/error"
```

връща резултатния html от изпълнението на шаблона app/view/main/error.rhtml.

- Интерпретиране на текст- използва се връщане към потребителския браузър на предварително изготвено съдържание. Може да се изпрати съобщение или директно html код. Често се използва като отговор на Ajax request.

  Пример: в TaskController - `render :text => "some string"`

- Интерпретиране на inline шаблон- това е преходен вариант между render_text и render_action. Резултата се визуализира в клиентския браузър както при render_text, с тази разлика, че подадения параметър преди това се интерпретира като Ruby код, т.е. функционира като render_action.

  Пример: в TeamController

```
render :inline => "<%= auto_complete_result(@users, 'usr') %>"
```

- save- Ако не съществува запис в базата с даденото id или обекта няма такова, то функцията създава нов запис в базата данни. В противен случай се прави обновяване на съществуващ запис. Прави се валидация на данните, ако такава е дефинирана в модела на съответния обект.
- update(id, attributes)- Намира запис в базата от данни с подаденото id и обновява атрибутите му. Прави се валидация на данните, ако такава е дефинирана в модела на съответния обект.
- update_attribute(name, value)- Обновява единствен атрибут в дадения обект и запазва промените в базата от данни. За разлика от гореописания метод, update_attribute не валидира данните. Това го прави особено подходящ за обновяване на булеви флагове на съществуващи записи.
- delete(id)- изтрива запис от базата данни с подаденото id без да инициализира обекта преди това.

- redirect_to(options = {}, parameters_for_method_reference)- Препраща клиентския браузер към страница посочена в options. Това може да стане по няколко начина:
  - чрез генериране на url, като се посочва контролер, метод и параметри на метода-
    redirect_to :controller => "user", :action => "list"
  - чрез предаване на готов url зададен като стринг-
    redirect_to "http://www.rubyonrails.org"
  - чрез предаване на url зададен като стринг, като се използва текущия протокол и хост- redirect_to "/images/screenshot.jpg"
  - чрез подаване на глобалната променлива :back, в която се съхранява предишния потребителски request-
    `redirect_to :back`

4.4. Обобщение

Досега бяха разгледани различни класове в системата както и тяхното значение и функции. Сега ще бъде направена оценка на начина, по който си взаимодействат те в рамките на model-view-controller pattern, който е архитектурен шаблон за разделяне на логиката на приложението на няколко слоя. Тези слоеве са независими един от друг.

Използването му определя архитектурата на приложението на три-слойна. Тези слоеве са:

1. view (потребителски интерфейс)- Това е резултата който е видим за потребителя.
2. controller- В контролерите е обособена бизнес логиката
3. model (модел на данните)- Това е нивото, на което данните на приложението се обединяват в обекти, наречени домейни (domain). Именно с тези обекти работят останалите нива от архитектурата. Това е крайното ниво от архитектурата, което оперира с базата от данни (ако такава има).

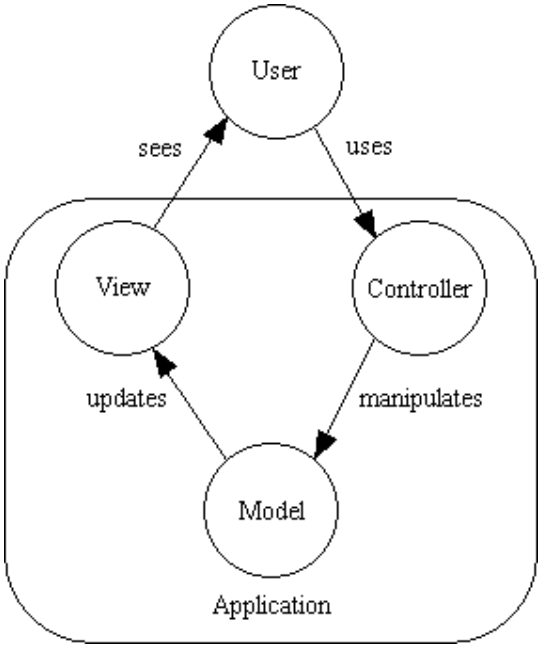
Най- общо начина на взаимодействие на нивата на архитектурата е следния:

1. Потребителя взаимодейства с потребителския интерфейс по някакъв начин (например натиска бутон или праща request за достъп до определен ресурс на системата).

2. Контролера прихваща идващото събитие от потребителския интерфейс
3. Контролера получава достъп до модела за да извърши зададеното от потребителя действие (например да обнови запис в базата от данни или да извлече такъв)
4. Данните от контролера се предават на изгледа, който генерира подходящ отговор за потребителя (например генерира html).
5. Потребителския интерфейс чака следващо действие на потребителя.

Ruby on Rails предлага именно това решение на разработчика, като генерира трите основни нива от архитектурата и връзките между тях. Тази платформа предлага и интегрирано решение за работа с релационна база от данни. Така на разработчика му се спестява времето за конфигуриране, а също така и се намалява възможността да бъде направена архитектурна грешка. Платформата на пречи йерархията от класове да се доразвива.

Фиг. 15 Взаимодействие между слоевете в Model-View-Controller pattern



Системата за управление на проекти е разработена и тестване върху следната конфигурация: Windows XP, Ruby- version 185-21, Ruby on Rails- version 1.1.6, XML-Simple- version 1.0.10, MySQL- version 5.0

Заключение

В настоящата дипломна работа е разгледан анализа, дизайна и цялостната разработка на система за управление на проекти в малки и средни фирми. Главната цел на приложението е да подпомогне ежедневната работа на екипа и неговите членове. Системата за управление на проекти не се нуждае от друг софтуер за да работи. Единственото условие е при създаване на базата от данни да се създадат потребителските групи и да се регистрира поне един администратор. Това се налага поради факта, е само администратора може да регистрира потребители. Приложението е тествано при конкурентен достъп на няколко потребители до един и същи обект, при което се запазват данните от първия пристигнал request. В този смисъл системата е подходяща за използване в малки и средни фирми.

На база на изготвеното приложение и неговото тестване може да бъде направено заключение, че са изпълнение поставените цели на дипломната работа, а именно:

- Да се определи необходимата на разработчика информация за ежедневната му работа
- Да се предостави възможност разработчика сам да организира детайлите от работата си по лесен и бърз начин
- Да се предостави информация за текущото състояние на проектите на всички заинтересувани страни, както и възможност за бързо обновяване на информацията

Като перспективи за бъдеще развитие могат да бъдат направено графично представяне на време зависимите обекти, като проекти и задачи. Също така някои компоненти са разработени само в базовата си функционалност. Например крайната дата на проект (задача) се счита датата, на която този проект (задача) бъде закрит. При бъдещо развитие на приложението може да бъде добавена препоръчителна крайна дата.

В заключение може да се каже, че създадената среда за управление на софтуерни проекти намира добра приложимост и е отворена за бъдещо усъвършенстване.

Използвана литература

[1] Gantt Project official web site: http://ganttproject.biz/

[2] Gforge web site: http://gforge.org/docman/view.php/1/34/gforge_manual.pdf

[3] Карл Чатфийлд, Тимъти Джонсън, 2003, Управление на проекти с Microsoft Project 2002, СофтПрес ООД, стр. 27

[4] Concurent Versions System: http://www.nongnu.org/cvs/

[5] MySQL: http://mysql.com/

[6] http://mysql.com/why-mysql/

[7] http://www.oreillynet.com/onlamp/blog/2004/04/why_mysql_grew_so_fast_news_fr.html

[8] Why open source and MySQL are winning corporate hearts By Jan Stafford, Editor 07 Sep 2004: http://searchenterpriselinux.techtarget.com/qna/0,289202,sid39_gci1003753,00.html

"For many applications, however, a company only needs a fairly basic database server. It has got to be fast, scalable and standard. In these situation, if an IT organization can get 90% of the functionality of an Oracle or DB2 database for 10% of the price, that is a pretty compelling argument for an open source database."

[9] Ruby: http://www.ruby-lang.org/en/

[10] Десет факта, които всеки Java програмист трябва да знае за Ruby http://www.ruby-doc.org/docs/10%20Things%20Every%20Java%20Programmer%20Should%20Know%20About%20Ruby/10things.tgz

[11] Ruby in a Nutshell, by Yukihiro Matsumoto, O'Reilly, nowember 2001

[12] http://www.randomhacks.net/articles/2005/12/03/why-ruby-is-an-acceptable-lisp - Eric Kidd, 3 dekemwri 2005

[13] http://www.ruby-lang.org/en/about/

[14] Agile Web Development with Rails, by Dave Thomas and David Hansson, July 2005, стр. 12

"...why you need a framework such as Ruby on Rails. The answer is pretty straightforward: Rails handles all of the low-level housekeeping for you—all those messy details that take so long to handle by yourself—and lets you concentrate on your application's core functionality."

[15] http://www.railsforall.org/info/why_develop_using_ruby_on_rails

[16] http://weblog.rubyonrails.org/2007/5/14/hi-i-m-ruby-on-rails

[17] http://onstartups.com/home/tabid/3339/bid/160/Reasons-Why-Your-Startup-Should-Use-Ruby-On-Rails-RoR.aspx

[18] WebRick: http://www.webrick.org/

[19] Ruby on Rails API documentation: http://api.rubyonrails.org/

Списък на използваните фигури