**ТЕХНОЛОГИЧЕСКО УЧИЛИЩЕ ЕЛЕКТРОННИ СИСТЕМИ**

**към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ**

**ДИПЛОМНА РАБОТА**

Тема: UnchainWorker

Дипломант: Научен ръководител:

*Иван Славянов Наков Марио Пешев*

СОФИЯ

2012

- **Задание** за ДР (по образец);

**УВОД**

Интернет е пространство от свързани по между си компютърни системи, които лесно, а понякога и бързо, могат да споделят информация. Най-широко разпространеният начин за предоставяне на информация в Интернет е под вида на уеб сайт. Има и други начини за споделяне на информация, като например електронната поща, но най-перспективният вариант остава уеб сайтът. През 1988 година започва навлизането на големи компании в Интернет пространството, търсещи печалба и реклама, които донасят големи инвестиции в Интернет общността. На 6 август 1991 е обявен проектът World Wide Web(W3), система от взаимно свързани хипертекстови документи. След W3 Интернет, и в частност уеб сайтовете, започват стремглаво да се развиват. Скоро след създаването на W3 статичните HTML страници стават не достатъчни и в началото на 21 век се появява понятието Богати Интернет приложения(Rich Internet applications, RIA). RIA сайтовете притежават голяма част от характеристиките на настолните приложения, по-бърз и динамичен потребителски интерфейс. Възможностите на богатите Интернет приложения се реализират с помощта на средства, като уеб браузърите. Така нареченият тънък клиент бива изместен от приложения работещи почти изцяло в потребителският браузър. Едни от най-широко разпространените платформи за разработка в момента са Adobe Flash, JavaFX и Microsoft SilverLight.

Уеб сайтовете днес са изключително много и се различават по тема и съдържание. Някои от тях са предназначени за реклама, други за бизнес, трети за забавление, четвърти за споделяне, но Интернет не е място, в което само големи корпорации могат да правят пари. Unchain Worker е приложение целящо да улеснява и осигурява работа през Интернет. В него могат да се регистрират два вида потребители, хора търсещи работа и хора предлагащи работа. Най-често в подобни приложения се публикуват обяви за решаване на софтуерни проблеми. Потребителите предлагащи работа лесно могат да обявяват задачи, като в обявата публикуват кратко описание на задачата и информация за това с колко пари и време разполагат. Потребителите търсещи работа лесно могат да предлагат оферти за задачите, които ги интересуват. Всяка оферта съдържа сума, за която даден работник би извършил определена задача. Всички оферти за дадена задача се виждат само от автора на задачата, по този начин се избягва обезценяването на труда и несъвършената конкуренция.

Всеки човек може да изпълнява задачи, за които да получава прилично възнаграждение. А хората нуждаещи се от решаване на софтуерен проблем могат бързо и на добра цена да го направят.

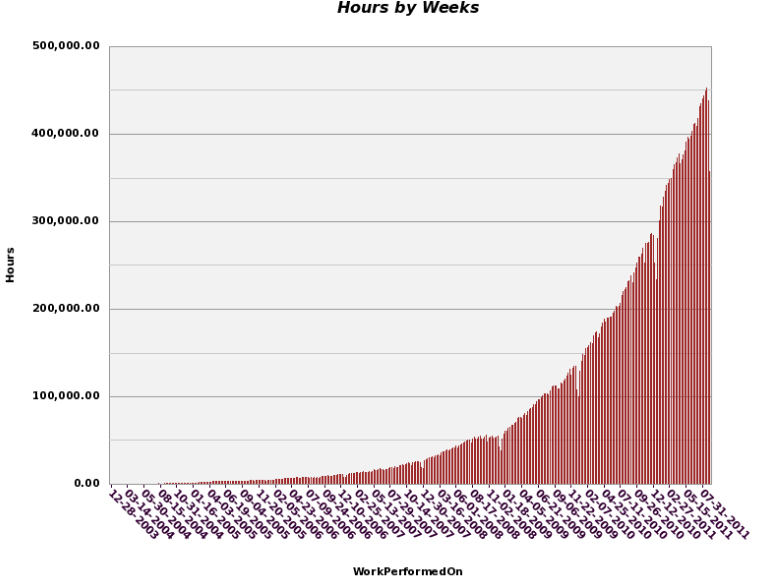
**ПЪРВА ГЛАВА**

**ПРЕГЛЕД НА ПОДОБНИТЕ ПРОДУКТИ НА „Unchain Worker” И ИЗВЕСТНИТЕ ТЕХНОЛОГИИ ЗА РАЗРАБОТКА НА УЕБ ПРИЛОЖЕНИЯ**

**1.1 Преглед на подобни продукти**

В момента в България няма добре развити подобни портали, точно това прави идеята на приложението ни доста перспективна. В световен мащаб идеята за този вид услуга е добре позната. Едни от най-големите приложения с подобна функционалност са oDesk, Elance, Freelancer и vWorker.

oDesk е най-голямото и най-бързо растящо работно място в Интернет. Само през 2011 година в приложението са публикувани около милион задачи и работници на свободна практика са изкарали над 220 милиона долара.

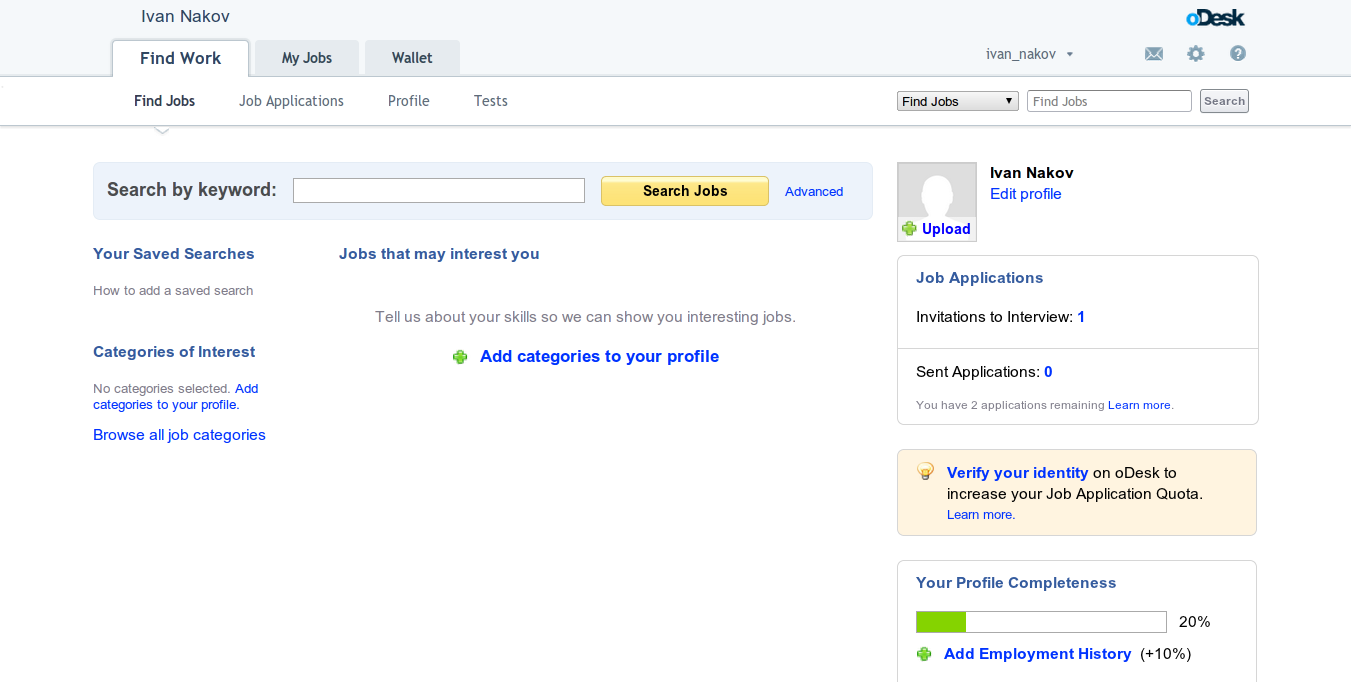


*Фигура 1.1 Диаграма на часовете работа в oDesk.*

oDesk е компания основана през 2003 от гръцките предприемачи Odysseas Tsatalos и Stratis Karamanlakis. Централата на oDesk е в Редуд Сити, Калифорния. Името на компанията идва от „no desk“, това е препратка към основната цел на компанията, а именно възможността на хората да вършат работата си без значение къде се намират. В оDesk има два вида потребители работодатели и работници (Employers и Contractors). oDesk позволява на работодателите да създават екипи от хора из целия свят, които биват координирани, чрез приложението. При свършена работа служителите получават парите си отново през системата. Работодателите могат безплатно да публикуват задачи или почасова работа, а работниците могат също безплатно да наддават за тях. Компанията прибира 10% от цената на свършената работа като комисионна. Плащанията стават през oDesk, което намалява много от счетоводната работа.

В oDesk работниците получават сигурност, която се изразява в гаранция за получаване на парите, които са си изработили. Те имат възможност да живеят навсякъде по света и да работят каквото пожелаят. Имат възможност да докажат и оценят уменията си с помощта на онлайн тестове, което им помага да печелят повече пари.

От своя страна работодателите получават възможност да платят само за часовете, в които се е извършвала реално работата, няма нужда да плащат болнични, и времето което служителите им не са оползотворили правилно. Това значително намалява техните разходи. Работодателите имат възможност да публикуват задачи и обяви безплатно. Те могат да работят с специалисти от цял свят и лесно да сформират работни екипи. Приложението съдържа както лична информация за работниците, така и богата информация за техният опит и умения, с помощта на тази информация лесно може да се намери точния служител.



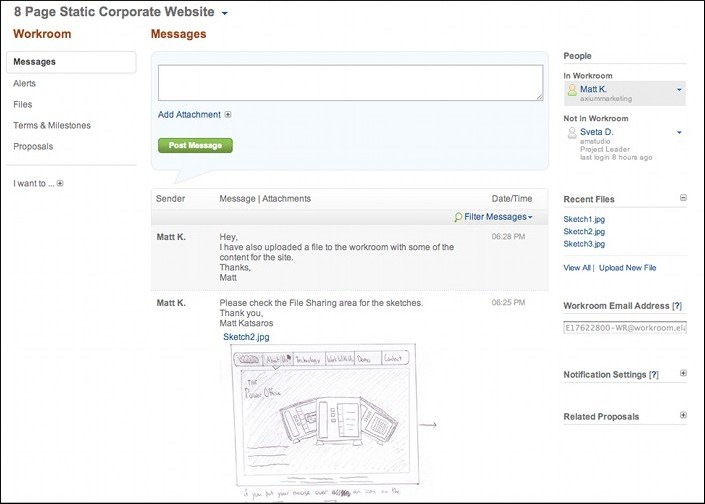
*Фигура 1.2 Начална страница на oDesk.*

Друг популярен портал е Gugu. Той предлага на бизнес партньорите си богат ресурс от талантливи програмисти на свободна практика. Тяхната система за препоръки(Guru Recommends System) прави изборът на програмист лесен и рутинен. Веднъж свързани с програмистите, работниците лесно и сигурно могат да извършват разплащания, чрез системата SafePay. За freelancer-ите Guru предлага иновативни решения, като онлайн Portfolios, които им помагат лесно да подчертаят качествата и уменията си.



*Фигура 1.3 Начална страница на Guru.*

Elance е глобална платформа за работа. Първата версия на сайта е пусната през 1999 година от Beerud Sheth и Srini Anumolu. Идеята им хрумва, когато забелязват, че по Интернет започват да се продават различни видове стоки и облигации и се замислили дали неможе да се направи същото с работата. През 2006 продават продуктът си на Click Commerce, след което е започнала разработката на сайта във вида, в който е и сега. След като веднъж се свържат работник и работодател цялата комуникация, размяна на файлове и разплащане минава през Elance Workroom. При превеждане на сумата Elance слага автоматично такса за услугата от порядъка на 7%-9%. Днес в Elance има регистрирани над 1.3 милиона потребители и има завършени задачи за около 500 милиона долара.



*Фигура 1.4 Workroom на Elance.*

**1.2. Преглед на технологиите за разработка на уеб приложения**

Едно уеб приложение може да се разработи с помощта на много и разнообразни технологии. Интернет сайт може да съдържа само HTML страници, но в нашият случай те не са достатъчни, за това трябва да изберем език за програмиране, който работи на сървъра и изпълнява заявки. Езиците, които могат да ни свършат работа са много PHP, Java, C/C++, Perl, Python, C# и други. Най-разпространените технологии, които се използват са тези на Microsoft .NET, Java и PHP.

Microsoft предлага ASP( Active Server Pages ) framework за разработка на динамични уеб сайтове, приложения и услуги. ASP .NET е изградено на Common Language Runtime (CLR), което позволява на програмистите да пишат код на всички поддържани от .NET езици (A#, Boo, C++, C#, IronPython и други). ASP .NET цели да ни предостави добрите страни на скипт-базираните технологии. Програмният код от страна на сървъра е в един или повече DDL файла (Dynamic-link library-реализацията на концепцията shared library в Microsoft Windows). ASP.NET MVC е базиран на архитектурния модел Model View Controller, който ни позволява да създаваме изчистен и разделен код. Това от своя страна прави приложенията бързи и по-лесни за тестване. Microsoft ни осигурява едно силно и удобно средство за изграждане уеб приложения, но за разработката на приложения на .NET е необходимо Visual Studio, за него е необходимо да се плати.

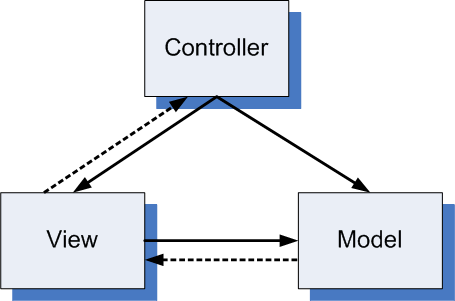
PHP това е скриптов език за програмиране, използва се предимно за изработка на сървърни приложения с динамично уеб съдържание. Авторът на езика е Размус Лердорф. Началото на PHP е поставено през 1994. През 1997 двама израелски студенти пренаписват целият език, като го изчистват от грешки и поправят цялостната концепция. PHP има множество предимства, той работи на много операционни системи (Unix, Linux, Windows, Mac OS и други) и сървъри, което от своя страна намалява значително разходите по поддръжката на приложенията. Той е лесен за разработване и се разпространява безплатно. Езикът е създаден и пригоден изцяло за разработка на уеб базирани приложения и е пригоден за лесно модифициране и адаптиране към нуждите на програмистите. Поради широкото си разпространение и опростения си синтаксис PHP се учи лесно и бързо.Това че има много програмисти на PHP води до по-евтин персонал и по ниски цени на готовите приложения. PHP поддържа много системи за управление на бази от данни, като например MS SQL, MySQL, Oracle, PostgreeSQL и други.

Заедно с всичките си облаги PHP има и много недостатъци. Езикът не поддържа напълно Unicode. Няма вградена поддръжка на нишки, което забавя и затруднява значително обработката на големи масиви от данни. Съществуват много функции от библиотеки, които вършат една и съща работа. Не се поддържат и пространства от имена до версия 5.3, което значително увеличава шанса за възникване на грешки и колизии.

Java това е обектно-ориентиран език за програмиране, чийто код не се компилира до машинен код за определен процесор, а до специфичен код, наречен байт код. Поради това за изпълнението на програма, написана на Java е необходима виртуална машина(Java Virtual Machine, JVM). Виртуалната машина има много предимства, като преносимост между различните платформи, по добър контрол на правата, по голяма сигурност. Допълнителните действия извършвани от виртуалната машина, като почистване на паметта и т.н, са много полезни. Разбира се виртуалната машина има и своите недостатъци. Главният й недостатък е необходимостта от допълнителни ресурси( като процесорно време и памет). Java се развива в три направления Standard Edition(Java SE), Enterprise Edition(Java EE) и Micro Edition(Java ME). Standard Edition е базовата версия която се използва за разработка на клиентски приложения и аплети. Enterprise Edition е базирана на SE, използва се най-често при сървърните приложения под формата на сървлети и Java Server Pages(JSP), но включва и много други софтуерни технологии. Java EE ни предлага множество от API-та за да улесни работата в многослойните приложения, като ORM, fault-tolerance, web services и други. Micro Edition е самостоятелна олекотена версия предназначена за мобилни телефони, смартфони и друг.

При разработка на софтуер е необходимо да се използва и подходяща за нуждите на програмиста развойна сред, която да улесни създаването и редактирането на програмния код. Програмният код може да се пише и в обикновен текстов редактор, но това забавя и затруднява производителността на програмистите. В момента най-използваните безплатни платформи за разработка на софтуер са NetBeans и Eclipse. NetBeans e платформа за разработка(Integrated development environment, IDE), предназначена за разработка на Java приложения, но поддържа и други езици, като PHP, JavaScript, Groovy, C++ и други. NetBeans е изработен на Java и може да работи под различни операционни системи. NetBeans лесно може да се разширява, с помощта на специални средства (plugin), за да задоволи нуждите на потребителите. Eclipse започнал като проект на IBM Канада, през ноември 2001г. продължава като проект с отворен код. Еclipse е разработен предимно за улесняване на Java програмисти, но поддържа и други езици. Eclipse Web Tools Platform (WPT) е среда базирана на стандартния Eclipse, но в нея са включени инструменти за улесняване на Web и Java EE разработчиците. В нея има вградени API-та за тестване, качване и стартиране на приложения в уеб сървър. Уеб сървърите, които поддържа Eclipse са много. Най-използваните контейнери съвместими с Eclipse са Tomcat, Jetty и други. Възможно е сървър да бъде инсталиран директно през IDE-то.

При започването на ново уеб приложение трябва да се помисли дали да се започне от нищото или да се използват фреймуърк, Content Management System (CMS). Ако се започне приложението от нулата се изискват много усиля и време, а шансовете за грешка стават големи, особено при неопитни програмисти. Фреймуърковете са създадени с цел да улеснят работата на програмистите, като им дават множество от преизползваеми библиотеки и класове. Те съдържат оптимизиран и добре проектиран код, който ни помага по-бързо и по-лесно да изграждаме големи проекти и да намалим шансовете за грешки. Фреймуърковете за уеб приложения са проектирани да намалят проблемите при често извършваните дейности в уеб програмирането, като достъп до бази данни, управление на сесиите и други. Повечето фреймлъркове за разработка на уеб приложения използват архитектурният модел Модел-Изглед-Контролер (Model-View-Controller, MVC). Те използват MVC за да изолират бизнес логиката на проекта (вътрешната му структура) от потребителския интерфейс, което от своя страна позволява едното да бъде променяно независимо от другото. В тази архитектура моделът се свързва с данните и тяхното съхранение, извличане и т.н., контролерът – с приложението, обработката на данните или вътрешната логика и изгледа (view) – с презентационния слой, т.е. с представянето на информацията на потребителя (фиг.1.5). Това разделя работата по приложението така, че да може да се работи по отделен елемент, без другите да бъдат афектирани. Чрез този подход кодирането става по-бързо и сигурно.



*Фигура 1.5 Структура на модела MVC.*

Фреймлърковете, които използват този архитектурен модел са много, някои от най-известните са Spring MVC, Struts, Symfony, CodeIgnter, CakePHP и други. В приложенията създадени от нулата има голям шанс програмистите да оставят пропуски в кода си, за това приложенията написани с помощта на фреймуърк са много по сигурни. Друг начин за създаването на уеб приложение е с помощта на CMS системи. Те се използват, за да се управлява съдържанието на даден уеб сайт. Идеята на CMS системите е да се предостави всичко необходимо на потребителя така, че без той да има технически познания дори по HTML да може само с използването на даден интерфейс да променя информацията в собствения си сайт. При тези продукти най-често се променя дизайна - спрямо изискванията на потребителя. Такива безплатни системи са Wordpress, Joomla, и Drupal.

**ВТОРА ГЛАВА**

**ФУНКЦИОНАЛНИ ИЗИСКВАНИЯ КЪМ ПРИЛОЖЕНИЕТО, АРГУМЕНТЦИЯ НА ИЗБОРА НА РАЗВОЙНИТЕ СРЕДСТВА И СРЕДИ И ОПИСАНИЕ НА СЦЕНАРИЯ ЗА РЕАЛИЗАЦИЯТА НА ПРОДУКТА**

**2.1. Функционални изисквания към „Unchain Worker”**

Приложението трябва да съдържа:

1. Два вида потребители – работници(Worker) и работодатели (Employer).
2. Възможност на регистрираните работодателите да създават задачи.
3. Възможност на регистрираните работници да предлагат оферти за дадена задача.
4. Възможност за преглед на офертите.
5. Качване на опит в различни умения при завършване на задача.

Приложението ще има проста и изчистена регистрационна форма. Тя ще изисква минимумът информация необходима на приложението да функционира правилно. След регистрация потребителите ще могат да добавят допълнителна информация за себе си като образование, опит и т.н. Всеки потребител ще притежава собствен профил, в които ще се съдържа информация за да него. Тази информация ще може да се променя. В приложението ще има два вида потребители работници и работодатели. За различните видове потребители приложението ще предлага различна функционалност. Работодателите ще могат да създават задачи и да ги възлагат на най-изгодните за тях работници. Работниците ще могат да предлагат оферти за различните задачи и да качват файлове с изпълнената работа. Работниците ще могат да разглеждат офертите и да проверяват задачите.

Началната страница на проекта трябва да съдържа поле с най-нови задачи. При избиране на задача потребителят ще може да разгледа подробна информация за нея.

При създаване на задача потребителите ще подпълват форма, чрез която ще описват работата, която им е необходима. Задължително работодателите ще трябва да избират заглавие и категория на задачата си, по този начин работниците ще се ориентират много по лесно и бързо. Всяка задача ще има списък с умения необходими за изпълняването й.

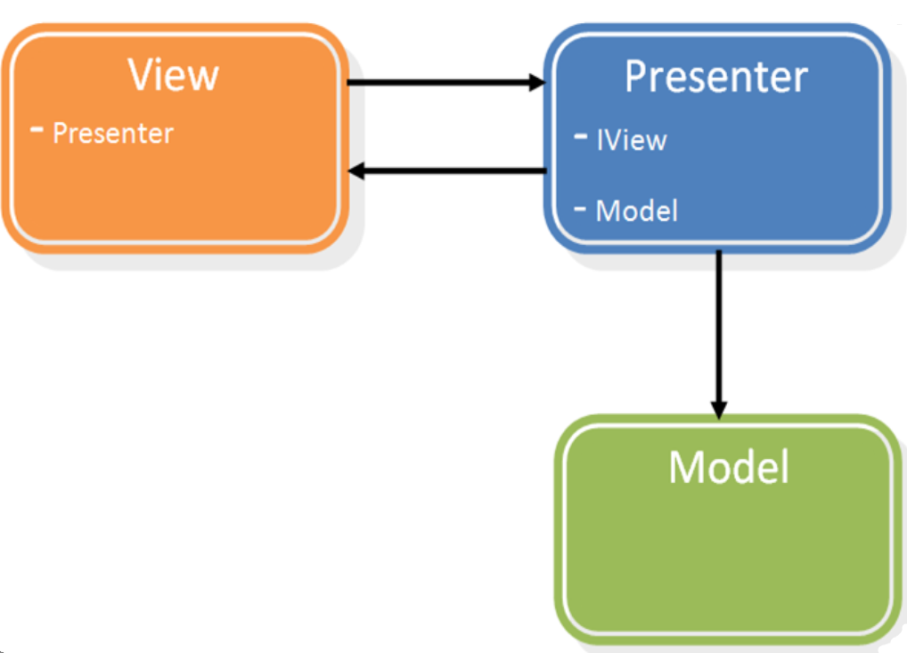
Работниците ще имат възможност за развитие. При изпълнение на задача от работник, той ще получава опит в уменията, които са му били необходими за да я завърши. Уменията ще имат нива на опит. По този начин работодателите лесно ще могат да преценяват, кой работник да назначат.

**2.2. Аргументация на избора на помощни средства**

Изборът на помощни средства за разработка на Java приложение е огромен. Съществуват много фреймуъркове и инструменти улесняващи разработката на Java приложения. Почти всички са отличен избор. Човек избира тези средства, с които има най-голям опит или са тясно свързани с това което прави. В UnchainWorker сме използвали едни от най-бързо развиващите се технологии. Те са много търсени и в тях се инвестират много пари, но най-важното е, че са с отворен код и са безплатни.

**2.2.1 Аргументация на избора на Google Web Toolkit.**

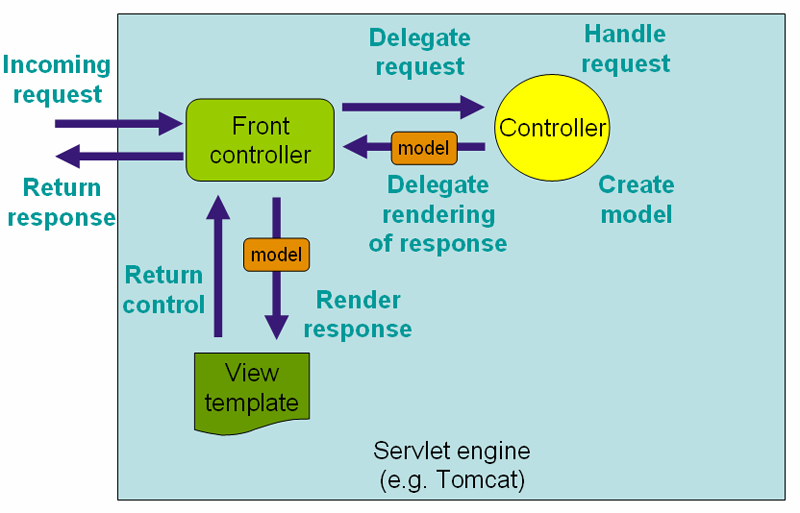
Google Web Toolkit (GWT) помага на програмистите да създават, богати Интернет приложения (RIA) много по-лесно от обикновено. За да създадем едно динамично уеб приложение, което да се изпълнява в потребителският браузър, са ни необходим HTML и JavaScript. Чрез AJAX (асинхронен JavaScript и XML) заявки, може да се достъпва информация на сървъра и да се обновява потребителският интерфейс без да се зарежда страницата отново. GWT ни предлага един вариант за програмиране на динамични сайтове, като използваме само Java. Това решава доста от проблемите със сложността на RIA сайтовете. GWT е фреймуърк, който улеснява и ускорява процеса на производство на уеб приложенията. На GWT кода се пише на Java и се компилира, оптимизира и използва като нормален JavaScript. Това че кода се пише на Java ни дава възможност да използваме съвременни средства за разработка на софтуер и откриване на грешки, като Eclipse IDE. По този начин се ускорява продуктивността и се намаляват разходите за разработка на приложения. Google Web Toolkit е с отворен код и е напълно безплатен, има добре развита общност към която може да се обърне всеки при възникване на проблем. При използване на GWT програмистите има пълна свобода да избират сами как да подреждат и как да създават кода си, но една от най-използваните методики е Model-View-Presenter(MVP). Структурата на model-view-presenter е показана на фиг.2.1. MVP е производен на model-view-controller, предимно се използва за изграждането на потребителският интерфейс. В MVP презентерът играе ролята на посредник между изгледа и модела. Изгледа приема допълнителна функционалност, а именно да се грижи за обработката на възникналите събития (натиснато копче, движение на мишката и така нататък). Моделът в повечето случай е сведен до стриктни домейн обекти.



*Фигура 2.1 Структура на модела MVP.*

**2.2.2 Аргументация на избора на Spring.**

Spring MVC е фреймуърк използващ архитектурният модел Модел-Изглед-Контролер(Model-View-Controller, MVC). Той използват MVC за да изолират бизнес логиката на проекта (вътрешната му структура) от потребителския интерфейс, което от своя страна позволява едното да бъде променяно независимо от другото. В тази архитектура моделът се свързва с данните и тяхното съхранение, извличане и т.н., контролерът – с приложението, обработката на данните или вътрешната логика и изгледа (view) – с презентационния слой, т.е. с представянето на информацията на потребителя. Това разделя работата по приложението така, че да може да се работи по отделен елемент, без другите да бъдат афектирани. Spring MVC е изграден около DispatcherServlet, който приема потребителската заявка и я насочва към правилния контролер, той обработва заявката и връща отговор обратно на DispatcherServlet-а. След това диспечера намира изглед, ако е необходим, и връща отговор на потребителя (в повечето случай това е browser). Spring контрол над този процес и начинът на откриване на точния контролер и точния изглед, което го прави много адаптивен и удобен.



*Фигура 2.2 Обработка на заявка в Spring MVC.*

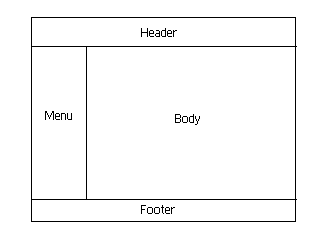
Spring Framework предоставя на потребителите си голям набор от библиотеки за работа. Те намаляват усилията в най-често извършваните дейности и драстично намаляват вероятността за грешка.

**2.2.3 Аргументация на избора на Hibernate.**

Защо Hibernate? Hibernate е технология с помощта, на която ние превръщаме таблиците от базата данни в обекти, а редовете от таблиците в полета на даден обект. Hibernate е много лесна и удобна технология. Тя има много подробна документация, която улеснява процеса на обучение. Ние използваме и Spring Framework, който ни предоставя класът HibernateDAOSupport, с чиято помощ работата с Hibernate става изключително лесна.

**2.2.4 Аргументация на избора на Apache Tiles.**

Tiles e фреймуърк предназначен изцяло за разработка на потребителския интерфейс в уеб приложения. Той ни помага да оползотворим пространството на екрана и да разделим изгледа в приложението на отделни компоненти наречени плочки. Всеки сайт се състои от компоненти (Header, Footer, Menu, Body фиг.2.3), като при преминаване от една. С Tiles ние преминаваме между страниците като променяме само съдържанието. Избрахме Apache Tiles заради простотата му и пълната му съвместимост със Spring MVC.



*Фигура 2.3 Примерно разделяне на уеб сайт.*

**2.2.5 Аргументация на избора на Maven2.**

За да съединим всички тези технологии и да ги направим лесни за употреба в UnchainWorker сме използвали Maven 2. Това е един инструмент, който в продължение на години се е утвърдил, като най-доброто решение при създаване на Java приложения. Maven е инструмент за управление на софтуерни проекти. Той е базиран на концепцията за обектния модел на проекта (project object model ,POM). Maven е страхотен инструмент с помощта, на когото можем да компилираме, пакетираме, тестваме проекти без да се налага да учим сложните и специфични файлове.

**2.2.6 Аргументация на избора на MySQL.**

При проектирането на базата данни изборът на СУБД е MySQL. Като основни причини за това могат да се посочат следните:

MySQL е силно мащабируема и лесна за администриране. Не се изисква опитен администратор на бази данни, нито за инсталирането, нито за нейното управление.

MySQL дава възможност за съхранение и управление на голямо количество данни. MySQL съхранява всяка таблица като отделен файл в директорията за базата данни. Максималният размер на една таблица може да бъде от 4GB до максималния размер на файл, поддържан от използваната операционна система.

Притежава бърза многонишкова система за заделяне на памет, като след изпълнение на заявката тя веднага се освобождава. Това е от голямо значение при продължителна работа и при много на брой свързани клиенти.

Позволява до 32 индекса за таблица като във всеки от тях могат да участват до 16 колони.

Поддържа полета с фиксирана и променлива дължина.

MySQL поддържа клиентски API (приложни интерфейси) за широк набор от програмни езици като Perl, C, PHP и др. Клиентските програми, извършващи достъп до данните в MySQL база данни могат да се напишат с помощта на тези API. В това отношение MySQL предоставя голям избор за програмен език при реализирането на среден слой.

MySQL е от около три до четири пъти по бърз от много други комерсиални сървъри за бази данни.

Това е релационна база данни с отворен код и се подчинява на лиценза GPL. Това прави MySQL финансово ефективна в сравнение с много други комерсиални релационни бази данни.

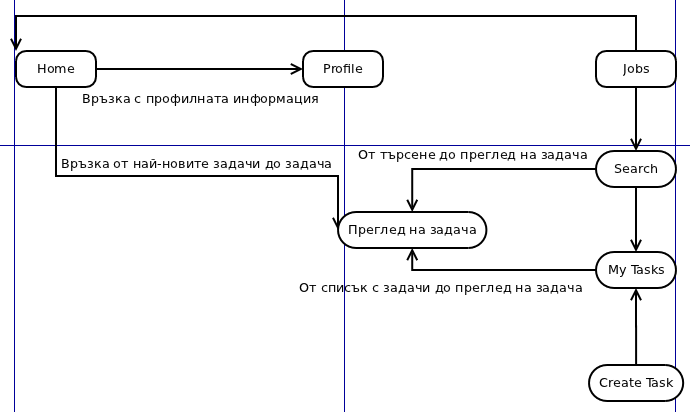
Непрестанно се усъвършенства – непрекъснато се пускат нови версии, прибавят се впечатляващи характеристики, разширява се функционалността.

**2.3. Описание на основни компоненти.**

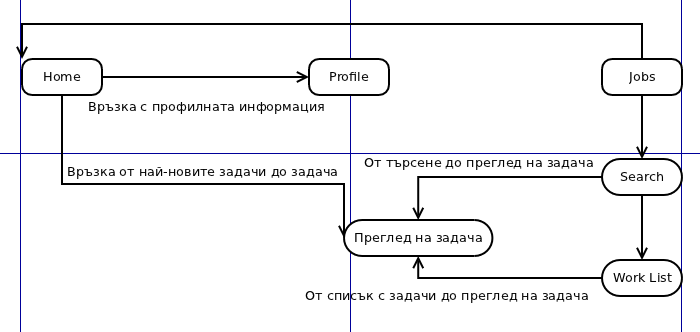
Следващите изображения представят структурата на сайта, диаграма на необходимите класове,таблици за съхранението на информацията и предварителен изглед на уеб приложението.

**2.3.1. Диаграма на връзките между секциите в сайта.**

В приложението има два вида потребители, всяка диаграма представлява връзките между секциите за различните потребители.



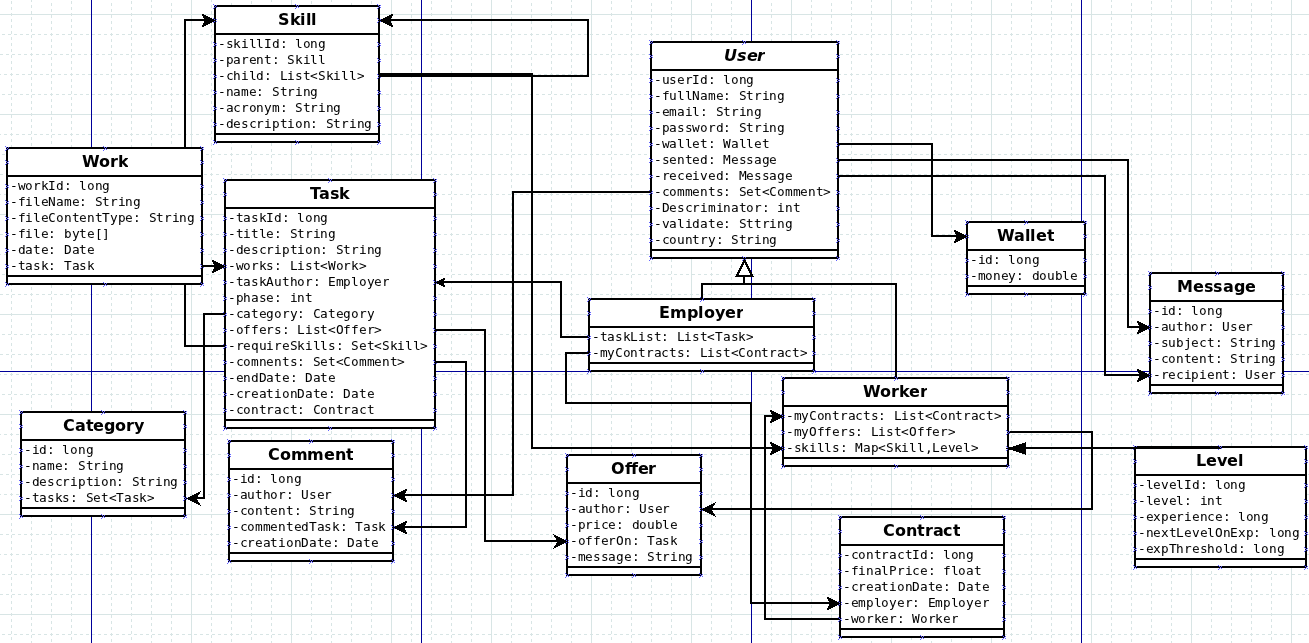
*Фигура 2.4 Връзки м/у страниците на работодателя.*



*Фигура 2.5 Връзки м/у страниците на работника.*

**2.3.2. Обобщена клас диаграма на обектите в сайта.**

Една от най-важните части на всеки Java базиран уеб проект е изграждането на подходящ набор от домейн обекти. Домейн обектите са атомите на приложенията. Те спазват стандартната Java Beans конвенция за именуване. Всички имена на полета започват с малка буква, а всяка следваща дума участваща в името започва с голяма буква(Пример: skillBuild, taskList и т.н.). Всички полета(атрибути) на даден клас са частни(private) и за всяко едно поле трябва да има методи наречени getters и setters, те също спазват конвенция. Различните полета могат да бъдат както за четене, така и писане. Полетата които могат да бъдат четени притежават метод getImeNaAtributa(), ако полето е от типа Boolean методът се кръщава isImeNaAtributa(). Атрибутите които могат да бъдат променяни притежават метод setImeNaAtributa. Те трябва да описват цялата информация, с която работи приложението и да я направят лесна за употреба. UML диаграмата на връзките между домейн обектите в UnchainWorker е следната:

*Фигура 2.6 Клас диаграма на UnchainWorker.*

**2.3.3 Структура и описание на основните таблици в базата данни.**

Основните таблици в база данни са:

|  |  |
| --- | --- |
| **Таблици** | **Съдържание** |
| User | Данни за потребителя |
| Task | Данни за задачите |
| Offer | Данни за офертите |
| Skill | Данни за уменията |
| Work | Данни за изпратените работи |

**2.3.3.1 Таблица „Task“.**

Съдържа описание и информация за задачите. Това е само основната информация без сложните връзки между обектите.

|  |  |  |  |
| --- | --- | --- | --- |
| **Име на полето** | **Тип на полето** | **Дължина** | **Описание** |
| taskId | bigint(auto inc) | 20 | Уникален номер на задача, генерира се от системата |
| title | varchar | 40 | Заглавие |
| description | text |  | Описание на задачите. |
| phase | tinyint | 1 | Период на задачата |
| creationDate | Data time |  | Дата на създаване |
| endDate | Data time |  | Краен срок |
| budget | Float | (10,2) | Бюджет. |

**2.3.3.2 Таблица „User“.**

Тя съдържа основната информация за потребителите. Връзките между обектът User и останалите обекти се изграждат с помощта на спомагателни таблици.

Полето Descriminator определя дали даден потребител е работник или работодател. Чрез него се представя наследствена връзка в една таблица. Потребителите съдържат сходни основни данни за това е по- добре да се запазват в една таблица.

|  |  |  |  |
| --- | --- | --- | --- |
| **Име на полето** | **Тип на полето** | **Дължина** | **Описание** |
| userId | bigint(auto inc) | 20 | Уникален номер на потребителя, генерира се от системата. |
| username | varchar (unique) | 30 | Потребителско име. |
| fullName | varchar | 60 | Пълно име. |
| email | varchar | 40 | Електронна поща. |
| password | varchar | 40 | Парола. |
| Descriminator | tinyint | 1 | Поле определящо типът на потребителя. |
| Validate | varchar | 40 | Валидационнен код. |
| country | varchar | 2 | Държава |

**2.3.3.3 Таблица „Offer“.**

Съдържа описание за офертите давани от работниците. Тези обекти съхраняват цялата информация необходима на един потребител да опише офертата си и да улесни работодателя максимално.

|  |  |  |  |
| --- | --- | --- | --- |
| **Име на полето** | **Тип на полето** | **Дължина** | **Описание** |
| offerId | bigint(auto inc) | 20 | Уникален номер на оферта, генерира се от системата |
| price | float | 10,2 | Цена на услугата |
| message | Varchar | 255 | Съобщение до собственикът на задачата |

**2.3.3.4 Таблица „Skill“.**

Съдържа описание и информация за всички умения регистрирани в системата. Това е таблицата, в която се въвеждат умения. Всяко умение има име и кратко описание, ако името е много дълго в повечето случаи има и съкращение от няколко букви(Пример: Java Standard Edition-JSE).

|  |  |  |  |
| --- | --- | --- | --- |
| **Име на полето** | **Тип на полето** | **Дължина** | **Описание** |
| skillId | bigint(auto inc) | 20 | Уникален номер на умението, генерира се от системата |
| name | varchar | 40 | Име |
| acronym | varchar | 10 | Кратко съкращение на името |
| description | varchar | 255 | Описание |

**2.3.3.5 Таблица „Work“.**

Съдържа описание и информация за всички изпратени файлове. В тази таблица се съхранява пълно описание на качените файлове. Файловете се запазват в базата данни като байтове. За да можем да ги възстановим запазваме информация за типът на файла и името му.

|  |  |  |  |
| --- | --- | --- | --- |
| **Име на полето** | **Тип на полето** | **Дължина** | **Описание** |
| workId | bigint(auto inc) | 20 | Уникален номер на изпратената работа, генерира се от системата |
| fileName | varchar | 50 | Име на файла |
| fileContentType | varchar | 20 | Тип на съхраняваният файл |
| file | BLОB |  | Описание |
| date | Date |  | Дата на изпращане |

**2.3.3.6 Спомагателни таблица и реализация на връзки м/у обектите.**

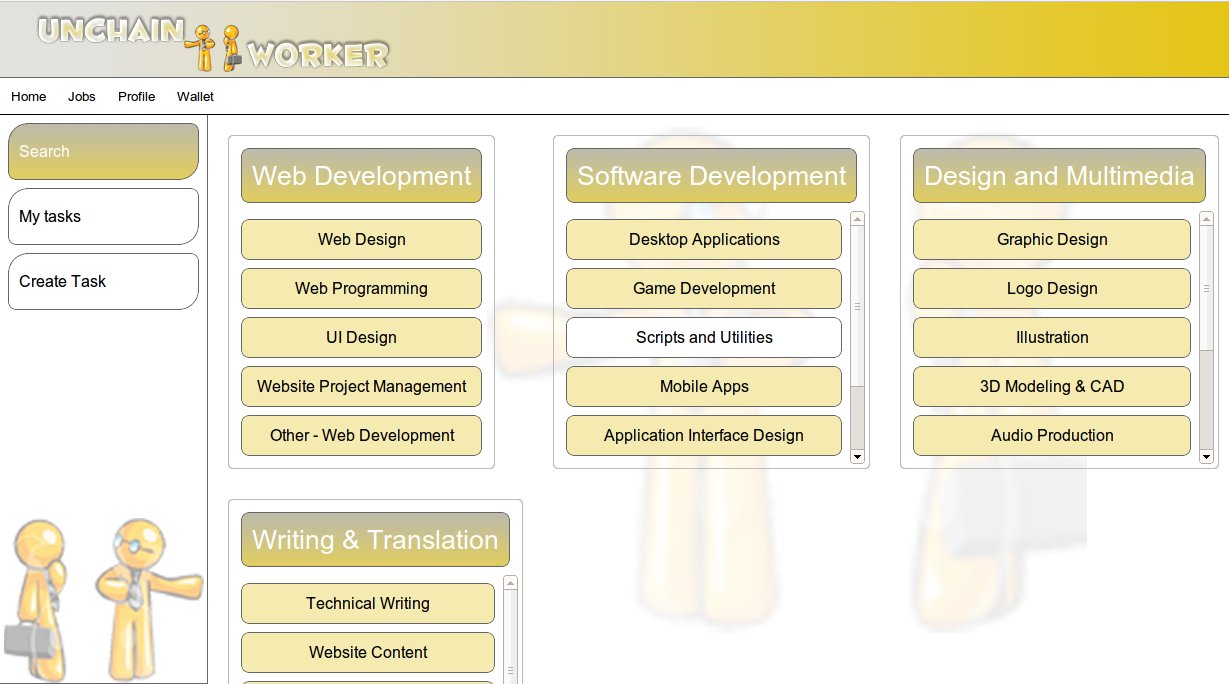
Всички връзки между обектите се реализират с помощта на спомагателни таблици. С тяхна помощ лесно може да се реализират отношения едно към много (One To Many) и много към много (Many To Many). Пример за връзка едно към много е следната таблица „Task\_Author“:

|  |  |  |  |
| --- | --- | --- | --- |
| **Име на полето** | **Тип на полето** | **Дължина** | **Описание** |
| userId | bigint | 20 | Номер на авторът на задачата взет от таблицата User |
| taskId | bigint | 20 | Номер на задачата взет от таблицата Task |

В зависимост от това кое поле ще е първичен ключ се определя посоката на връзката. Ако полето taskId е първичен ключ означава, че един работодател може да има много задачи. Ако полето userId е първичен ключ означава, че една задача може да има много автори. Ако в таблицата няма първичен ключ връзката преминава в много към много(Many To Many).

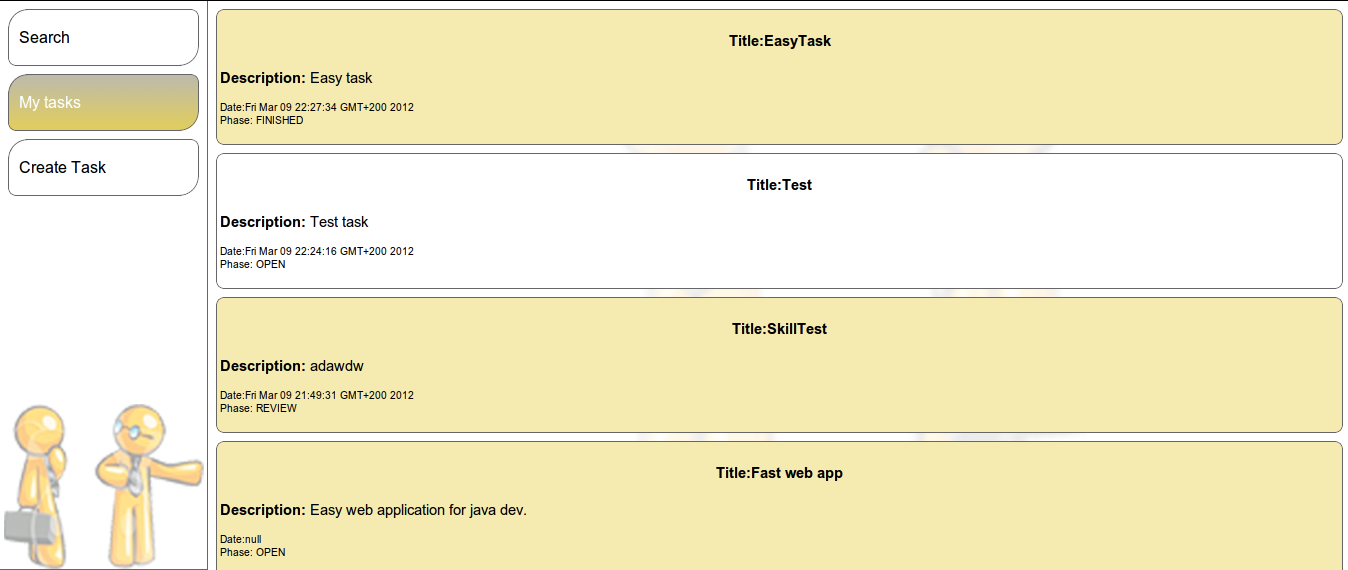
**2.3.4. Структура на интерфейса на приложението.**

Една от основните функционалности на приложението е в секцията „Търсене“ . Тя предоставя възможност за преглеждане на всички задачи по различни критерии.



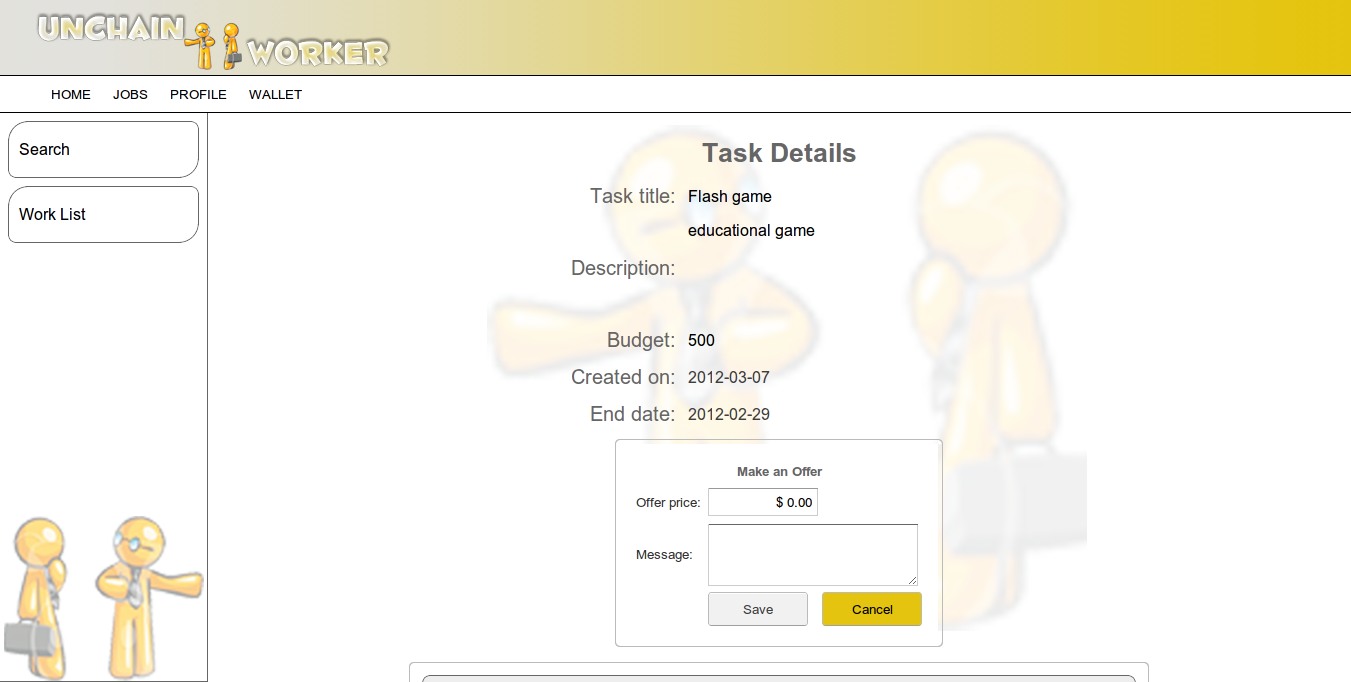
*Фигура 2.7 Секция „Search“ на UnchainWorker.*

Двата вида потребители имат списък със задачи. На работодателите списъкът съдържа всички задачи, които е създал, а на работника всички задачи, по които е работил.

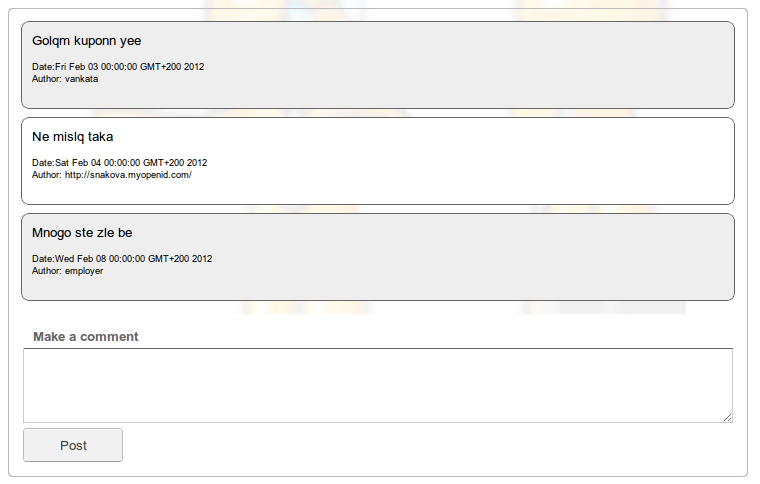


*Фигура 2.8 Секция „My tasks“ на UnchainWorker.*

При избиране на задача от списъка потребителят бива пренасочен към специален изглед за четене на задача. От него потребителите могат да пишат коментари и да комуникират.

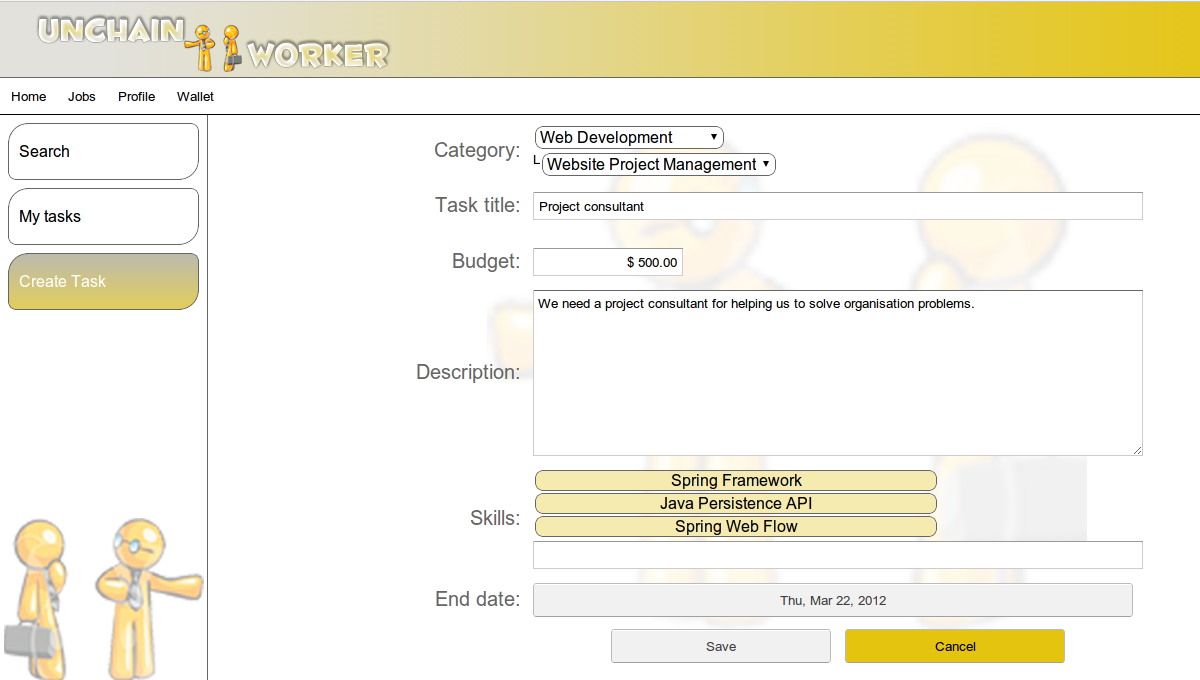


*Фигура 2.9 Изглед за четене на задача в UnchainWorker.*



*Фигура 2.10 Възможност за добавяне на коментари.*

Работодателите получават възможност за създаване на задачи, чрез удобен и бърз потребителски интерфейс.



*Фигура 2.11 Добавяне на задача в UnchainWorker.*

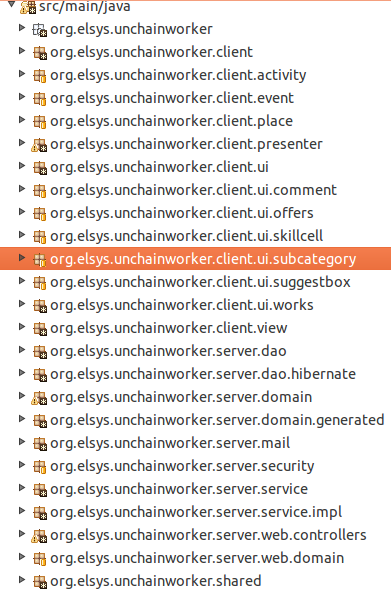
**ТРЕТА ГЛАВА**

**РАЗРАБОТКА НА УЕБ ПРИЛОЖЕНИЕТО „Unchain Worker”**

**3.1. Основна пакетна структура на приложението „Unchain Worker“.**

Основната структура на директориите в UnchainWorker (фиг.3.1), започва с конвенцията на всеки Мaven проект. Едно от най-големите предимства на Maven е използването на най-добрите и често срещани решения за структуриране и изграждане на Java проекти, като конвенции. Те могат лесно да бъдат заменени, но кодът ще загуби яснотата си. Една от най-важните Maven конвенции е винаги да държим Java кода си в директория src/main/java, а кода на тестовете в src/test/java. Конфигурационните файлове и ресурсите, като на изображения, отиват в src/main/resources, а ресурсите свързани с тестовете в src/test/resources. Тази конвенция улеснява много подреждането на кода и разбирането му от външни хора. След това сме използвали обърнатият домейн на училището и името на проекта, като начален пакет: org.elsys.unchainworker.

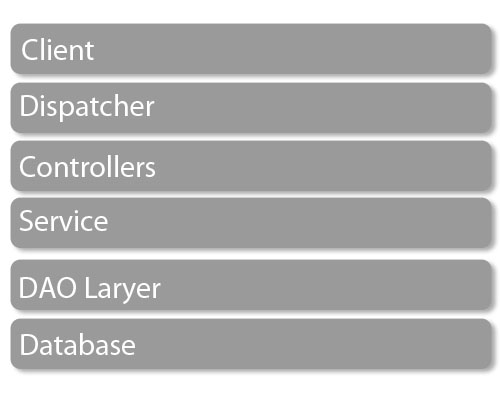
В основният пакет има още три пакета client, server и shared и UnchainWoker.gwt.xml, това са пакети с изключителна важност за Google Web Toolkit (GWT) и файл съдържащ описанието на GWT модула. Първият от под пакетите е client всичките файлове поставени в него се обръщат в JavaScript. Всичко това е много хубаво, но върви с някои ограничения, до версия 1.5 на GWT този пакет е можел да съдържа само код до Java 1.4 включително. Част от полезните Java класове и функционалности немогат да бъдат използвани, като Finalization, getStackTrace(), нишки и други. Друго ограничение е необходимостта от кода на всички външни библиотеки, който трябва да бъде предоставен на GWT компилатора. Следващия пакет е server в него сме свободни да поставяме каквото си искаме Java 1.6, Spring, Hibernate. Като цяло не сме длъжни да създаваме пакет server, той не се описва в GWT модула, което означава че всеки пакет извън client и shared се счита за сървърна част и не се компилира до JavaScript. Но по конвенция е хубаво да имаме server пакет. Последния от пакетите е shared съдържа комбинация от Java код и ресурси като HTML, CSS и така нататък.



*Фигура 3.1 Структура на пакетите.*

**3.2. Основни слоеве на приложението „Unchain Worker“.**

Сървърната част на UnchainWorker е изградена и разделена на няколко слоя, които са взаимно обвързани. Всеки слой комуникира само със слоевете над и под него. Всеки слой има собствени задължения. Сървърната част е разделена с цел да се направи лесна за тестване, разбиране и употреба.



*Фигура 3.2 Основни слоеве.*

Controllers слоят съдържа така наречените контролери. Това са Java класове, които са описани с анотации, така че Spring да ги разпознава като контролери. Те са прости и имат за цел да приемат потребителската заявка и да извлекът от нея необходимата информация, след което подават информацията на модела. След като долните слоеве си свършат работата, контролера приема обработената информацията и я предава на слоят над него, за да бъде изпратена на клиента. Слоят на контролерите управлява цялото поведение на приложението при получена заявка. С помощта на Spring MVC ние сме обвързали определени URL адреси със методи на контролерите, това може да стане по няколко начина, чрез XML конфигурация или с анотации.

Service слоя (Service Layer, SL) се грижи за цялата бизнес логика на приложението. Чрез него се предоставя цялата функционалност на приложението. В него се изпълняват алгоритмите на приложението. В него се съдържат Java класове, който са вкарани в Spring контейнера. За да се използва даден клас, чрез Spring контейнера има отново два най-често използвани варианта с помощта на XML или с анотации. С анотации е по бързият и изчистен вариант. Слоят на услугите е изграден от интерфейс и имплементация. Комуникацията между слоевете става през интерфейса, а имплементацията съдържа логиката. Това разделение прави много лесна подмяната на слоевете и тестването на приложението.

**public** **interface** UserService {

User createUser(User user);

User save(User user);

**boolean** exists(String username);

UserInfo getCurrentUserInfo();

User getCurrentUser();

**void** activateUser(String valid);

User getUser(**long** id);

Map<Skill,Level> getSkillBuild(**long** workerId);

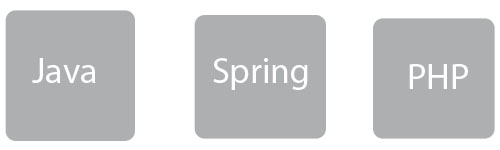
}

Това е интерфейса UserService, той определя методите, чрез които приложението ще управлява и променя User обектите. Всички класове, които реализират (имплементират) този интерфейс могат да бъдат лесно замествани и тествани без да се нарушава работата на останалата част от приложението. В Unchain Worker има няколко основни класа, които определят поведението на приложението с различните обекти.

Слой за достъп до данни(Data Access Layer, DAL) обгръща базата от данни и всички заявки към нея минават през слоят. Този слой отново е разделен на интерфейс и имплементация. Използвали сме Hibernate за Object-relational mapping, той ни помага да свържем таблиците в базата и да ги представим като обекти. В имплементацията на DAL класовете правят всички заявки към базата данни. Слоят на услугите (Service Layer) няма директна връзка с базата данни, което го прави по лесен за тестване. Трябва само да се направят Mock заместители на класовете от DAL(Data Access Layer). Слоят за достъп на данни също е регистриран в Spring контейнера.

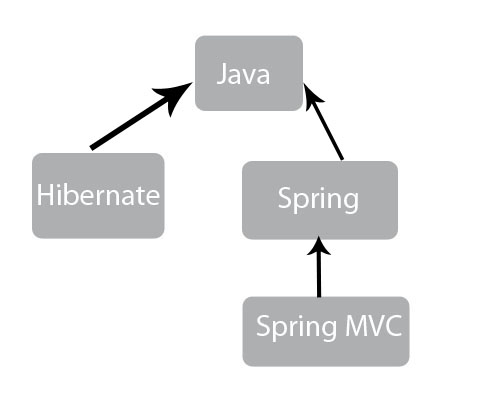
**3.3. Описание на алгоритъма за развити на умения.**

При проучването на подобни приложения, забелязахме, че всички портали предлагат защита на работниците от некоректни платци. Но много по труден е изборът на работодателите, на кого да възлагат задачите си. За да улеснят потребителите порталите предлагат рейтингови системи, чрез който да се ориентират по лесно. При рейтинговите системи потребителите трябва да са много внимателни и наблюдателни при преценката си. В обикновените приложения потребителите сами въвеждат своите умения, което позволява на работниците да злоупотребяват и подвеждат работодателите. Работниците често добавят умения, които не притежават. В UnchainWorker сме сменили подходът, което прави приложението уникално и перспективно. В нашето приложение работниците не въвеждат сами уменията си, а ги доказват. При изпълнение на дадена задача работниците получават опит в уменията, които са им били необходими за да я изпълнят. На базата на тази идея ние започнахме да съхраняваме в базата различни умения и различни технологии (Пример: Java, Spring, PHP).



*Фигура 3.3 Умения.*

Всяка технология е базирана на дадена идеология, език за програмиране и цели да направи програмирането по лесно като ни предоставя функционалност, която да използваме. Пример за това е технологията Spring MVC. Тя се базира на архитектурния модел Model-View-Controller и е базирана на Java. Скоро забелязахме, че в най-общият случай уменията имат връзки по между си. От тази връзка се предполага, че когато един програмист има опит в Spring MVC, той има опит и в Java, защото Spring е базиран на Java. Ние представим тази връзка в базата данни като мрежа от родители и деца, по този начин лесно могат да се развиват свързаните умения. Например, ако работник извърши задача, за която му е било необходимо умението Spring MVC, той не получава опит само в Spring MVC, а и в Spring Framework, Java EE, Java.



*Фигура 3.4 Връзка м/у уменията.*

При изпращане на свършена работа, ако работодател прецени, че работата не е завършена, той може да връща задачата за преработка. Това става чрез следния метод извикван в SL.

**public** **void** returnTaskBackToWork(Long taskId) {

Task task = taskDAO.getTaskById(taskId);

task.setPhase(TaskConstants.*IMPLEMENTING*);

taskDAO.save(task);

}

Методът приема параметър от типа long. Параметърът е идентификатор на задачата, която искаме да върнем в процес на разработка. В началото на метода взимаме задачата от базата данни и я преместваме във фаза за имплементация. Накрая запазваме промените в базата.

Ако задачата е завършена и работодателят я затвори се изпълнява следния метод.

**public** **void** finishTask(Long taskId) {

Task task = taskDAO.getFetchTaskForLevelUp(taskId);

Worker worker = task.getContract().getWorker();

List<Skill> skills = skillDAO.getSkillsByTaskId(taskId);

**int** exp = (**int**) (task.getContract().getFinalPrice() \* 1.3);

**for** (Skill skill : skills) {

SkillLevelingUp(worker, skill, exp);

}

task.setPhase(TaskConstants.*FINISHED*);

userService.save(worker);

taskDAO.save(task);

// Make pay to worker

}

Първо намираме задачата, работника и необходимите умения за реализацията на задачата. След това изчисляваме количеството опит, което ще получи работникът. Опитът се формира като цената на задачата се умножава с 1.3. След като сме изчислили опита обхождаме всички умения и викаме методът SkillLevelingUp(worker, skill, exp) за всяко от тях. Този метод приема работник, на когото ще бъдат качени уменията, уменията, които ще бъдат развити и с колко опит ще бъдат развити те. След това преместваме задачата във фаза „Finished“ и запазваме промените в базата данни. На края се извършват разплащанията.

**3.4. Описание на алгоритъма за качване на опит в умение.**

Метода SkillLevelingUp(worker, skill, exp) проверява дали работника вече е имал опит в дадено умение. Ако е имал го повишава, а ако не се добавя в списъкът му с умния.

**private** **void** SkillLevelingUp(Worker u, Skill s, **int** exp) {

Map<Skill, Level> skills = u.getSkillBuild();

**do** {

**if** (skills.containsKey(s)) {

Level l = skills.get(s);

l.addExperience(exp, **true**);

} **else** {

Level l = **new** Level(exp);

skills.put(s, l);

}

u.setSkillBuild(skills);

s = s.getParent();

} **while** (s != **null**);

}

Методът обхожда не само умението, което е подадено, а и всичките родителски умения по веригата. Чрез специални коефициенти може да се определи, точно кое родителско умение с колко да се повиши. С правилните изчисления на коефициентите приложението може много точно да събира информация за уменията и знанията на работниците. На базата на тази информация могат да се предлагат на потребителите задачи, които да са им по възможностите и които може да ги интересуват. Качването на опитът в дадено умение се извършва с помощта на методът addExperience(exp, **true**).

**public** **void** addExperience(**long** experience, **boolean** useThreshold) {

**if** (**this**.experience + experience < nextLevelOnExp) {

**if** (useThreshold) {

**if** (experience > expThreshold)

**this**.experience += experience;

} **else** {

**this**.experience += experience;

}

} **else** {

experience -= (**this**.nextLevelOnExp - **this**.experience);

**this**.experience = **this**.nextLevelOnExp;

**this**.level++;

**this**.nextLevelOnExp \*= 1.8;

**this**.expThreshold = (nextLevelOnExp \* 10) / 100;

addExperience(experience, **false**);

}

}

Често срещана измама от страна на работниците е извършването на множество ниско платени задачи, чрез което си повишават рейтинга. В повечето случаи те се наемат сами. За да избегнем подобен вид хитрина сме измислили алгоритъм за качване на нивото на дадено умение. Така всяко ниво си има даден минимум (праг) на опитът, който може да му се качи. И ако даден потребител е изпълнил 100 задачи за по 1$, той няма да достигне нивото на опит на друг потребител извършил 1 задачи за 100$. Опитът който получава даден потребител се базира на цената на задачата. След определен брой завършени задачи на една цена нивото на уменията ще спре да се качва.

Методът приема стойност на опитът, който искаме да качим на определеното умение и булева променлива. Булевата променлива се използва за да се включва и изключва прагът на опит. Това ни е необходимо, когато качваме голям опит на малко ниво. По този начин избягваме загубата на опит. След това правим проверка дали опитът е достатъчен за да качи нивото на умение. Ако подаденият опит не е достатъчен за качване на ниво събираме новия със старя опит. Ако е достатъчен или по голям качваме нивото на умението и продължаваме да викаме рекурсивно метода докато остатъчният опит не стане 0. Опитът необходим за качване на ниво(nextLevelOnExp) се качва с 80% за всяко следващо ниво. При качване на ниво прага(expThreshold) се качва с 10%.

**ЧЕТВЪРТА ГЛАВА**

**РЪКОВОДСТВО НА ПОТРЕБИТЕЛЯ**

**4.1. Описание на първоначалната конфигурация на продукта**

Начин на инсталиране на приложението UnchainWorker. Първата стъпка е инсталация на база данни MySQL[11]. След това се добавя базата данни на приложението, намираща се в Setups/unchain\_worker.sql или директно на диска.

Инсталиране на Java сървлет контейнер. Има богат избор на сървлет контейнери, като например Tomcat, Jetty, JBOS, GlassFish и други. Приложението е тествано на Tomcat[10] и Jetty. Инсталация на Maven 2. Приложението UnchainWorker е разработени с помощта на Maven2, което го прави лесен и удобен за употреба[6]. За да се пакетира приложението е необходимо да се използва командата *mvn:pakage.* След въвеждането на тази команда maven ще генерира пакет завършващ на .war. Последната стъпка е качването на war файлът на инсталирания сървлет контейнер. На дискът има приложен файл unchainworker.war, чрез който проекта може лесно да се постави в сървър.

В базата данни на приложението има регистрирани тестови акаунти: Работодател – потребителско име: employer, парола: employer

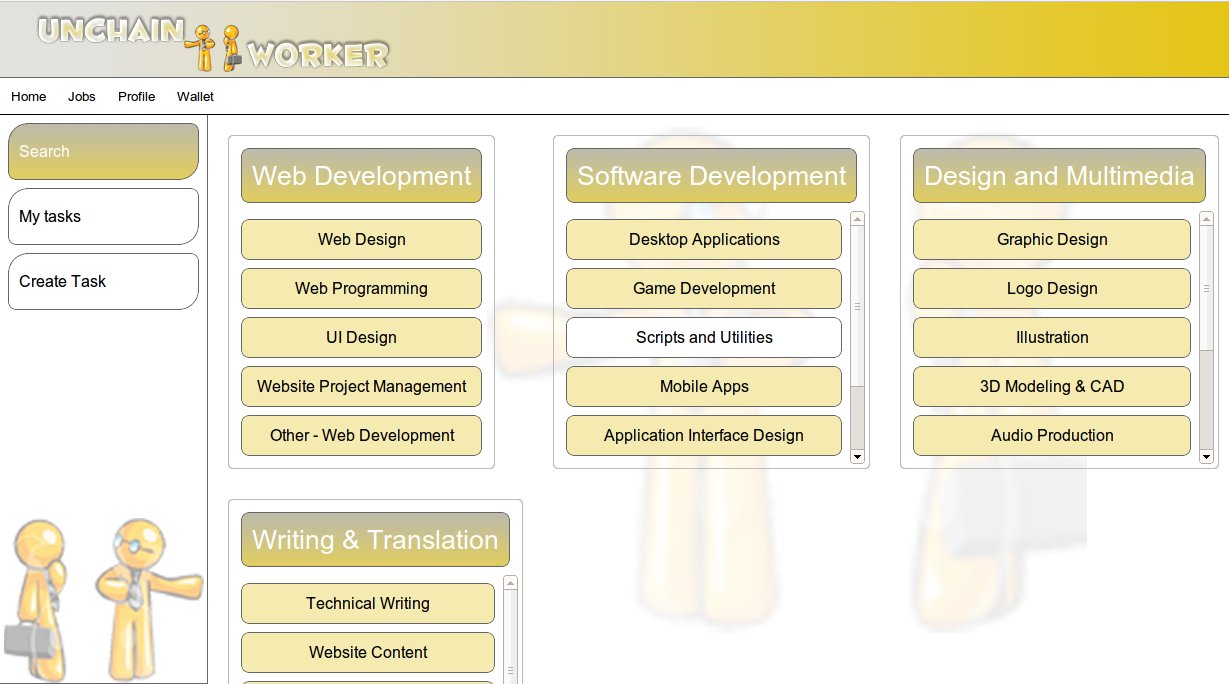
Работник – потребителско име: worker, парола: worker

Потребителят по подразбиране за MySQL използва: потребителско име: root и парола: root. Те могат да се променят лесно от src/main/resources/db.properties.

При използване на приложението от локален сървър началната страница се намира на : http://localhost:8080/unchainworker/app/home.

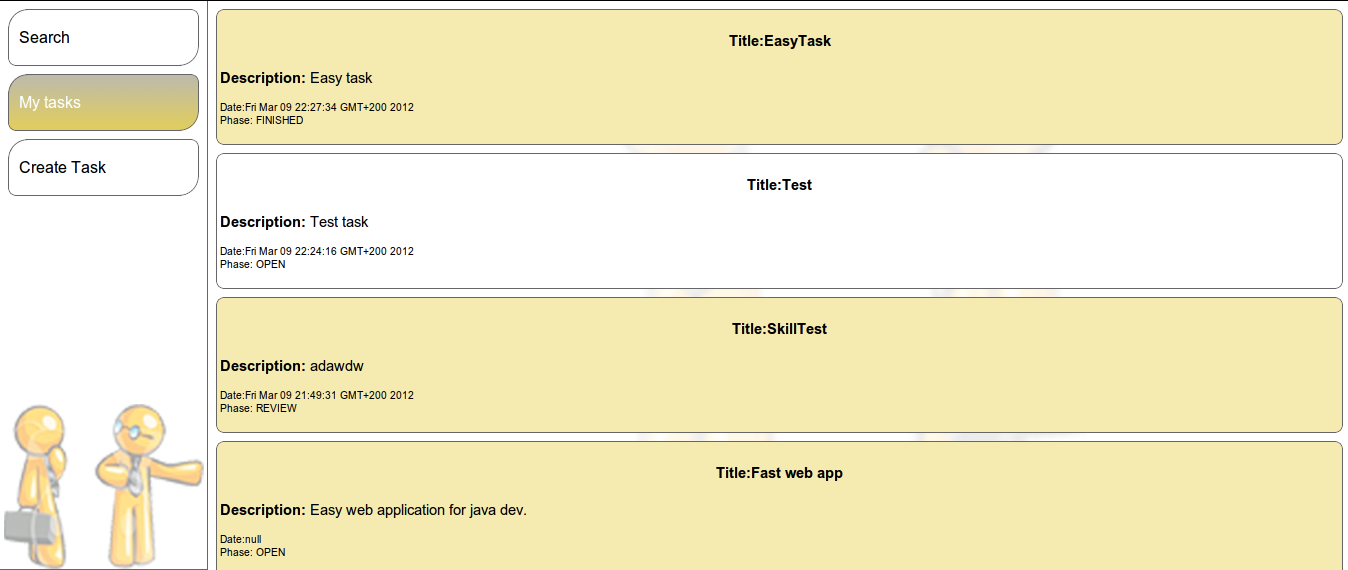
**4.2. Описание на работата с „Unchain Worker” посредством изградения потребителски интерфейс.**

В приложението има два вида потребители работници (Workers) и работодатели (Employers). При регистрация потребителят избира по какъв начин иска да използва приложението. Според това какъв потребител използва приложението, интерфейсът се променя и предлага различна функционалност. Обща за работници и работодатели е услугата за разглеждане на задачи по категории. При избиране на опцията Search от менюто потребителя се пренасочва към място(place), в което са изброени всички категории и подкатегории на приложението. При избиране на подкатегория се отваря таблица(CellTable на GWT) съдържаща всички задачи, принадлежащи на категорията. Задачите са разделени на страници. Всяка страница съдържа 15 задачи.



*Фигура 4.1 Секция “Search”.*

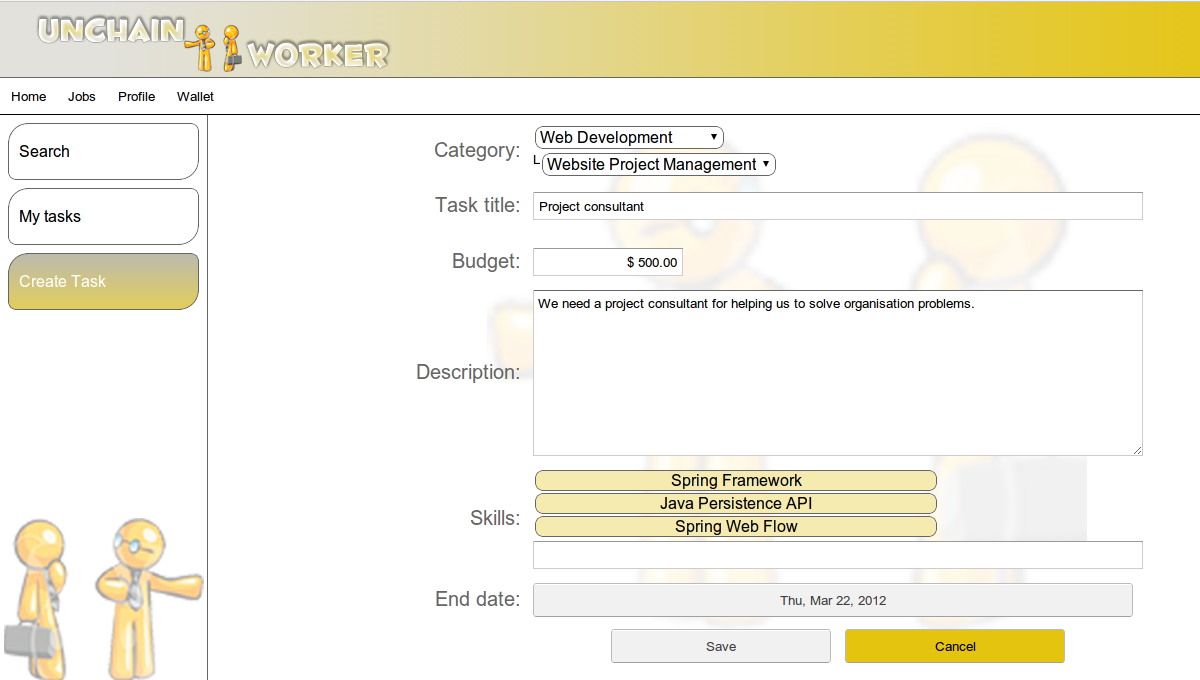
Двата вида потребители имат списък със задачи. На работодателите списъкът съдържа всички задачи, които е създал, а на работниците всички задачи, по които е работил.



*Фигура 4.2 Секция “My Tasks”.*

При избиране на задача от списъка потребителят бива пренасочен към специален изглед за четене на задача. От него потребителите могат да пишат коментари и да комуникират.

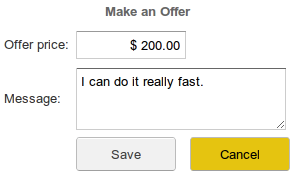
Работодателите получават възможност за създаване на задачи, чрез удобен и бърз потребителски интерфейс.



*Фигура 4.3 Секция “Create Task”.*

Първата стъпка при създаването на задача е избора на категория. Работодателите имат възможност да избират от списък с категории. Важно е да се избере възможно най-точната категория, по този начин задачата ще бъде по лесна за намиране. След категория потребителят задължително трябва да измисли заглавие за работата, която предлага. Заглавието е едно от първите неща, които виждат останалите потребители, затова заглавието трябва да е внимателно подбрано и да насочва към целта на задачата. След това работодателите въвеждат сумата, с която разполагат. Тя е ориентир за работниците, чрез нея те по-лесно могат да определят офертата си. Едно от най-важните неща, при създаване на задача, е определянето умения необходими за изпълнението й. По тези умения изпълнителите могат по лесно да преценят, дали ще се справят със задачата. Това са и уменията, в които работниците ще получат опит след завършване на задача.

*Фигура 4.4 Инструмент за създаване на оферти.*

Работниците могат безплатно да разглеждат всички задачи и да предлагат оферти. Всяка оферта съдържа цена, за която работник предлага да свърши работата и съобщение до работодателя. Офертите могат да бъдат променяни по всяко време. Оферта може да бъде видяна само от автора на задачата или от работника, чиято е офертата.

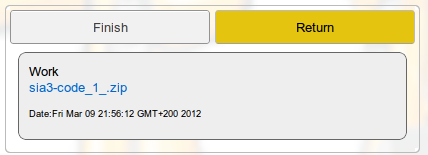
Собственикът на задача получава списък с всички предложени оферти. Той преценява, на кого да възложи задачата, като предплати за работата. При възлагане на задача приложението съхранява парите, които работодателите са предплатили, с цел да се избегнат некоректни платци.

*Фигура 4.5 Списък с всички оферти.*



F:\ScreenShots\WorkUpload.pngСлед като задача е възложена на даден работник, той може да изпраща файлове със свършената работа. Файловете се запазват в базата данни като байтове, което позволява да се правят копия на цялата информация и в случаи на проблем със сървъра да се избягва голяма загуба на информация.

*Фигура 4.6 Инструмент за качване на свършената работа.*

Когато работник качи файл за определена работа, задачата минава в процес на преглед. По време на преглед собственика на задачата преценява дали работата е свършена или не. Ако работата не е свършена той може да върне задачата в процес на разработка и да накара работника да я довърши. Ако работата е изпълнена задачата се затваря и работника получава пари и опит.

*Фигура 4.7 Списък с изпратените файлове.*

**Заключение**

Изискванията към приложението UnchainWorker бяха изпълнени и беше създадено здраво приложение, предоставящо на потребителите замислената функционалност. Всички цели и идеи, поставени в самото начало, бяха реализирани по един доста адекватен и рационален начин. Бяха разучени и овладяни нови технологии за изграждането на уеб приложения. Това включва средите за разработка, използваните програмни продукти и езици за създаването на сайта. Беше изграден основният елемент за развитие на потребителите с една новаторска идея. Ядро на приложението беше завършено и с негова помощ проекта може да расте и да се развива в една успешна посока. Една от основните ни цели ще е добавянето на система предлагаща на работниците задачи, които може да ги интересуват. Предложените задачи ще се намират на базата на уменията на потребителите. За работодателите ще се постараем да улесним максимално избора на правилния работник, като сортираме и маркираме различните оферти.

В бъдеще продуктът ще се ориентира все повече към намаляването на риска от измами и опростяването на потребителският интерфейс до степен, в която ще бъде интуитивен и лесен за употреба от всеки.

**Използвана литература:**

1. Подобни продукти на “UnchainWorker”:

<https://www.odesk.com/>

<https://www.elance.com>

<http://www.guru.com/>

1. w3c schools уроци:

<http://www.w3schools.com/>

1. Същност на CMS системите:

<http://searchsoa.techtarget.com/definition/content-management-system>

<http://plone.org/documentation/faq/what-is-a-cms>

1. GWT документация:

<http://code.google.com/intl/bg/webtoolkit/overview.html>

1. Spring MVC документация:

<http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/mvc.html>

1. Maven 2 документация:

<http://maven.apache.org/>

1. Hibernate документация:

<http://www.hibernate.org/docs>

1. Apache Tiles:

<http://tiles.apache.org/>

1. Google I/O:

<http://www.google.com/events/io/2011/sessions/high-performance-gwt-best-practices-for-writing-smaller-faster-apps.html>

<http://www.google.com/events/io/2011/sessions/highly-productive-gwt-rapid-development-with-app-engine-objectify-requestfactory-and-gwt-platform.html>

<http://www.google.com/events/io/2011/sessions/using-gwt-and-eclipse-to-build-great-mobile-web-apps.html>

1. Apache Tomcat:

<http://tomcat.apache.org/>

1. MySql database:

<http://www.mysql.com/>

1. Jeff Dwyer, Pro Web 2.0 – Application Development, 2008.
2. Federico Kereki, Essential GWT, 2010.
3. Craig Walls, Spring in Action 3, 2011.
4. Peter Mularien, Spring Security 3, 2010.
5. Dhrubojyoti Kayal, Pro Java™ EE Spring Patterns, 2008
6. Jeff Linwood and Dave Minter, Beginning Hibernate Second Edition, 2010.

**Съдържание:**

**Увод**................................................................................................................................4

1. **Първа глава:** Преглед на подобните продукти на „Unchain Worker” и известните технологии за разработка на уеб приложения.................................6
   1. Преглед на подобни продукти........................................................................6
   2. Преглед на технологиите за разработка на уеб приложения.....................10
2. **Втора глава:** Функционални изисквания към приложението, аргументация на избора на развойните средства и среди и описание на сценария за реализацията на продукта.....................................................................................15
   1. Функционални изисквания към „Unchain Worker ”...................................15
   2. Аргументация на избора на помощни средства……..................................16
      1. Аргументация на избора на Google Web Toolkit…………………..17
      2. Аргументация на избора на Spring…………………………………18
      3. Аргументация на избора на Hibernate……………………………...19
      4. Аргументация на избора на Apache Tiles…………………………..20
      5. Аргументация на избора на Maven2………………………………..20
      6. Аргументация на избора на MySQL………………………………..21
   3. Описание на основни компоненти…………………...................................22
      1. Диаграма на връзките между секциите в сайта……………………22
      2. Обобщена клас диаграма на обектите в сайта……………………..23
      3. Структура и описание на основните таблици в базата данни…….25
         1. Таблица „Task“……………..…………………………………..25
         2. Таблица „User“……….………………………………………….26
         3. Таблица „Offer“…………………………………………………27
         4. Таблица „Skill“…………………………………………………..27
         5. Таблица „Work“…………………………………………………28
         6. Спомагателни таблица и реализация на връзки м/у обектите…………………………………….…………………….28
      4. Структура на интерфейса на приложението……………………….29
3. **Трета глава:** Разработка на уеб приложението „Unchain Worker”.................31
   1. Основна пакетна структура на приложението „Unchain Worker“……….32
   2. Основни слоеве на приложението „Unchain Worker“…………………….33
   3. Описание на алгоритъма за развити на умения…………………………...36
   4. Описание на алгоритъма за качване на опит в умение…………………...39
4. **Четвърта глава:** Ръководство на потребителя.................................................58
   1. Описание на първоначалната конфигурация на продукта.........................42
   2. Описание на работата с „Unchain Worker ” посредством изградения потребителски интерфейс..............................................................................43

**Заключение**.................................................................................................................47

**Използвана литература**............................................................................................48