МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

учреждение образования

«Гродненский государственный университет имени Янки Купалы»

Факультет математики и информатики

**Кафедра современных технологий программирования**

КОЛЯГО АРТЁМ ОЛЕГОВИЧ & ГОЛИЦЫН АНДРЕЙ БОРИСОВИЧ

**Quests site**

Курсовая работа

студентов 2 курса специальности

«Управление информационными ресурсами»

дневной формы получения образования

|  |  |
| --- | --- |
|  | Научный руководитель  Гуща Юлия Вальдемаровна,  Старший преподаватель кафедры современных технологий программирования |
|  |  |

Гродно, 2015

**РЕЗЮМЕ**

Голицын Андрей Борисович & Коляго Артём Олегович

Курсовая работа – «Quests site», ?? страниц, ?? иллюстраций, ?? таблиц, ?? графиков, ?? использованных источников.

Ключевые слова – web-приложение, квесты, соревнование, игра, турнир, ASP.NET, Bootstrap, SQL, SQL Server, .Net, JavaScript, система контроля версий, GitHub.

Цель исследования – получение практических навыков совместной разработки web-приложений, основанных на технологии ASP.NET, с использованием фреймворка Bootstrap и системы управления реляционными базами данных (СУРБД) Microsoft SQL Server.

Объект исследования – создание web-сайтов.

Предмет исследования – технология создания web-приложений и web-сервисов ASP.NET, SQL, Microsoft SQL Server, Bootstrap, JavaScript

Методы исследования – теоретический (теоретический анализ технической документации и иных источников), моделирование (разработка модели сайта), лабораторный опыт (получение практических навыков путём эксперимента), индукция и дедукция (поиск и решение проблем путём рассуждений от частного к общему и наоборот), тестирование (поиск ошибок в конечном продукте).

Авторская характеристика работы – курсовая работа представляет собой творческий порыв двух пытливых умов, который вылился в web-приложение разработанное с использованием новейших технологий и фреймворков и имеющее дальнейшую перспективу развития и использования.

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ, УСЛОВНЫХ ОБОЗНАЧЕНИЙ

*~~Текст перечня должен располагаться столбцом, в котором слева в алфавитном порядке приводятся элементы перечня, справа – их детальная расшифровка.~~*

*~~Если сокращения, условные обозначения, символы, единицы и специфические термины повторяются в работе~~* ***~~не более двух раз~~****~~, отдельный перечень их~~* ***~~не составляют~~****~~, а их расшифровку приводят непосредственно в тексте при первом упоминании~~*

|  |  |  |
| --- | --- | --- |
| API | Application Programming Interface | набор готовых классов, процедур, функций и структур, предоставляемых библиотекой или сервисом для использования во внешних программных продуктах. |
| HTML | HyperText Markup Language | язык для структурирования и представления содержимого всемирной паутины. |

ВВЕДЕНИЕ

Наличие множества программных платформ – условие в которых существует современный IT рынок – это и плюс и минус: каждая из них нацелена на определённого потребителя, каждая имеет свои недостатки и достоинства. Следовательно, разработка кросплатформенных приложений – наболее актуальная проблема большинства IT компаний. Каждый производитель программного обеспечения пытается при минимальных затратах времени и ресурсов разработчиков получить продукт, который будет доступен потребителям, использующим различные платформы. Логичным выглядит подобное желание компаний, которое, однако, не всегда реализуемо на практике. Возможность воплощения стратегии «максимального покрытия рынка» зависит во многом от задачи, которую призвано выполнять ПО.

С распространением интернета новые возможности, предоставляемые пользователю, требуют наличия досупа к необходимой информации в любой момент времени, с любого устройства, имеющего выход в сеть. Кроме того, существует необходимость не только получения, но и обработки этой информации в момент поступления. Подобными требованиями явлются получене оперативной финансовой нформации, совметное редактирование документов и разработка ПО, предоставление оперативного доступа к базам данных, интернет торговля. Эти предопосылки привели к появлению нового типа ПО – web-приложений. Web-приложения не требовтельны к аппаратным возможностям и позволяют выполнять их даже на мобильных устройствах: смартфонах, планшетах, трансформерах и т.д. Единственное требование – наличие web-браузера, который выступает в качестве интерфейса между пользователем и сервером, на катором выполняется приложение. Таким образом, независимые от платформы, на которой они используются, web-приложения предоставляют едный опыт использования.

Ещё пару лет назад desktop-приложения существовали на равных с web-приложениям, чего нельзя сказать сейчас. Постепенно инициатива переходит в руки web продуктов, которые обеспечивают функционал идентичный тому, который предлагают desktop-приложения. Хотя web-приложения предлагают в большинстве случаев схожий опыт использования, desktop-приложения имеют преимущество, т.к. используют вычислительные мощности машины, на которой программа непосредственно выполняется, чего нельзя сказать о web-продуктах, которые задаюствуют в процессе вычислений и обработки запросов ресурсы серверов. Из этого следует, что приложения соновная оласть применения которых - обработка 3D графики, видеомонтаж, разработка ПО, «тяжёлые» игры, будут представлены в форме desktop продуктов. Однако стоит заметить, что все эти направления в большей или меньшей степени уже представлены web-приложениями. Ещё одним и, пожалуй, последним преимуществом desktop-приложений перед web является их бо́льшая пропускная способность. Главными же преимуществами web перед desktop-приложениями (как бы странно это ни звучало) является возможность использования предоставлемого дискового пространства и вычислительных ресурсов backend-ов. Последний пункт, как может заметить рецензент, является противоречивым, ибо выступает и как преимущество, и как недостаток. Преимущество: возможность не нагружать «железо» слабой машины, недостаток: вычислительные ресурсы предоставляются в значительно меньшем количестве.

Множество существующих социальных сервисов – представителей социальных медиа – в связке с вычислительными возможностями серверов позволяют объеденять пользователей и устаналивать между ними связи, включая обмен потоками данных, которые агрегируются в зависимости от их действий и влияют на текущее состояние других пользователей, использующих web-приложение как в данный момент, так и тех, кто находится в offline. Становится возможным установить сообщение между группами с общими целями и интересами, объединить их в сообщество, установаить контакт.

Курсовая работа представляет собой пример использования данной технологии: web-приложение. Растущая популярность рынка web-приложений подтверждает актуальность настоящей курсовой работы. При раскрытии темы курсовой работы были исследованы примеры документации, интернет-форумы, руководства по работе с базами данных Microsoft SQL Server, Microsoft .Net, Bootstrap, JavaScript и иным технологиям. Следовательно, объектмаи работы выступили технологии .NET, Java, фреймворк Bootstrap. Исследование вылиось в более детальное изучение ASP.NET, JavaScript, которые являются предметными областями исследучемых объектов.

Использованные методы исследования позвоили разностороннее проаналиировать объект курсовой работы. Теоретический метод позволил охватить наиболее ценный материал доступный в сети по исследуемому объекту; метод моделирования позволил составить архитектуру сайта, установить связи, которые возникают в процессе использования web-приложения; опытным путём на лабораторных работах был получен базовый опыт, который в дальнейшем был расширен за счёт информации полученной в результате прочтения технической документации; методом индукции и дедукции были синтезированы классы, используемые в модели сайта, вынесены обобщения, интерфейсы, классы-помощники, которые упрощают логику написания кода; методом тестирования были найдены узкие места в логике взаимодействия с СУБД, обнаружены критические ошибки и мелкие упущения ухудшающие пользовательский опыт, в результате чего была получена стройная архитектура приложения с приятным интерфейсом.

ГЛАВА 1  
АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

* 1. Основные аспекты

Web-приложение представляет собой web-сайт, на котором размещены страницы с частично либо полностью несформированным содержимым. Окончательное содержимое формируется только после того, как посетитель сайта запросит страницу с web-сервера. В связи с тем, что окончательное содержимое страницы зависит от запроса, созданного на основе действий посетителя, такая страница называется динамической.

Использование web-приложений приносит определенную пользу как посетителям web-сайтов, так и их разработчикам:

* Web-приложения позволяют посетителям быстро и легко находить требуемую информацию на web-сайтах с большим объемом информации.
* Данный вид приложений позволяет осуществлять поиск в содержимом, упорядочивать содержимое и перемещаться по нему удобным для посетителей способом. Примерами таких приложений могут служить внутренние сети компаний — Microsoft MSDN (www.msdn.microsoft.com) и Amazon.com (www.amazon.com).
* Web-приложения позволяют собирать, сохранять и анализировать данные, полученные от посетителей сайта.

1.2 Принципы работы web-приложений

Долгое время использовался метод, при котором данные, введенные в HTML-формы, отсылались для обработки CGI-приложениям (CGI (от англ. Common Gateway Interface - «общий интерфейс шлюза») - стандарт интерфейса, используемого для связи внешней программы с web-сервером) или специально назначенным работникам в виде сообщений электронной почты. Web-приложение позволяет сохранять данные непосредственно в базе данных, а также получать данные и формировать отчеты на основе полученных данных для анализа. В качестве примера можно привести интерактивные страницы банков, страницы для контроля товарных запасов, социологические исследования и опросы, а также формы для обратной связи с пользователями.

Web-приложение может использоваться для обновления web-сайтов с периодически меняющимся содержимым.

Web-приложение освобождает web-дизайнера от рутинной работы постоянного обновления HTML-страниц сайта. Поставщики содержимого, например редакторы новостей, отвечают за наличие свежего материала, а web-приложение следит за автоматическим обновлением сайта. В качестве примера можно привести web-версию журнала «The Economist» (www.economist.com) и службы новостей CNN (www.cnn.com).

1.2 Обзор существующих решений

Любое web-приложение представляет собой набор статических и динамических web-страниц. Статическая web-страница — это страница, которая всегда отображается перед пользователем в неизменном виде. Web-сервер отправляет страницу по запросу web-браузера без каких-либо изменений. В противоположность этому при использовании динамических страниц, сервер вносит изменения в web-страницу перед отправкой ее браузеру. По причине того, что страница меняется, она называется динамической.

Например, можно создать страницу, на которой будут отображены результаты программы оздоровления. При этом некоторая информация (например, имя сотрудника и его результаты) будет определяться в момент запроса страницы сотрудником.

1.2.1 Решение первое: Обработка статических web-страниц

Статический web-сайт содержит набор соответствующих HTML-страниц и файлов, размещенных на компьютере, на котором установлен web-сервер.

Web-сервер — это программное обеспечение, которое предоставляет web-страницы в ответ на запросы web-браузеров. Обычно запрос страницы создается при щелчке ссылки на web-странице, выборе закладки в браузере либо вводе URL-адреса в адресной строке браузера.

Окончательное содержимое статической web-страницы определяется разработчиком и остается неизменным в процессе запроса страницы (см. Листинг 1).

Весь HTML-код создается разработчиком до того момента, когда страница будет размещена на сервере. Поскольку HTML-код не меняется после размещения страницы на сервере, данная страница называется статической.

Примечание. Если подходить строго, «статическая» страница может на самом деле таковой не являться. Например, замещающее изображение либо Flash-содержимое (в виде SWF-файла) позволяет «оживить» статическую страницу. Однако в данном контексте страница является статической, поскольку отсылается браузеру без изменений.

Когда web-сервер получает запрос на выдачу статической страницы, то, после анализа запроса, сервер находит нужную страницу и отправляет ее браузеру, как показано ниже:

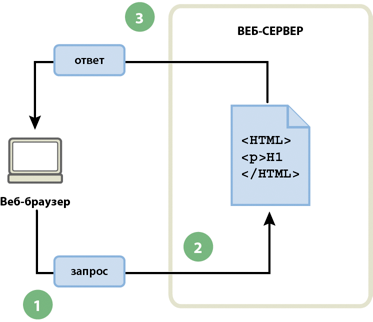


Рисунок 1.1 – Взаимодействие между клиентом и сервером при статической обработке web-страниц

1. Web-браузер запрашивает статическую страницу.
2. Web-сервер находит страницу.
3. Web-сервер отправляет страницу запросившему ее браузеру.

В случае web-приложений некоторые участки кода страницы отсутствуют до момента запроса страницы посетителем. Отсутствующий код формируется с помощью некоторого механизма, и только после этого страница может быть отправлена браузеру. Данный механизм формирования кода рассматривается в следующем разделе.

1.2.2 Решение второе: Обработка динамических страниц

Когда web-сервер получает запрос на выдачу статической web-страницы, он отправляет страницу непосредственно браузеру. Однако, когда запрашивается динамическая страница, действия web-сервера не столь однозначны. Сервер передает страницу специальной программе, которая и формирует окончательную страницу. Такая программа называется сервером приложений.

Сервер приложений выполняет чтение кода, находящегося на странице, формирует окончательную страницу в соответствии с прочитанным кодом, а затем удаляет его из страницы. В результате всех этих операций получается статическая страница, которая передается web-серверу, который в свою очередь отправляет ее клиентскому браузеру. Все страницы, которые получает браузер, содержат только HTML-код. Схематическое изображение процесса:

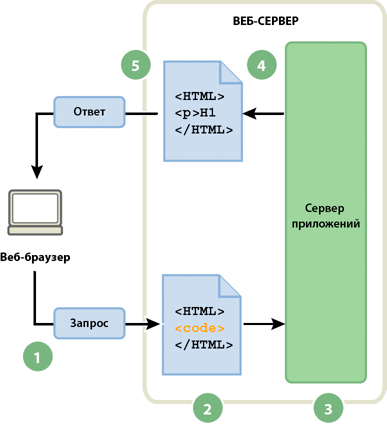


Рисунок 1.2 – Взаимодействие между клиентом и сервером при статической обработке web-страниц

1. Web-браузер запрашивает динамическую страницу.
2. Web-сервер находит страницу и передает ее серверу приложений.
3. Сервер приложений просматривает страницу на наличие инструкций и выполняет ее создание.
4. Сервер приложений возвращает подготовленную страницу на web-сервер
5. Web-сервер отправляет подготовленную страницу запросившему ее браузеру

1.3 Доступ к базе данных

Сервер приложений предоставляет возможность использовать такие ресурсы сервера, как базы данных. Например, динамическая страница может содержать программные инструкции для сервера приложений, следуя которым серверу необходимо получить определенные данные из базы данных и поместить их в HTML-код страницы.

Хранение содержимого в базе данных позволяет отделить оформление web-сайта от содержимого, которое будут видеть пользователи. Вместо того чтобы создавать все страницы в виде отдельных HTML-файлов, пишутся только шаблоны страниц для каждого вида представляемой информации. Затем содержимое загружается в базу данных, после чего web-сайт будет извлекать его при запросах пользователей. Кроме того, можно обновить информацию в одном источнике и продублировать это изменение на всем web-сайте без редактирования каждой страницы вручную. Adobe® Dreamweaver® позволяет создавать web-формы для вставки, обновления и удаления информации в базе данных.

Программная инструкция, предназначенная для получения данных из базы данных, называется запросом к базе данных. Запрос состоит из критериев поиска, выраженных с помощью языка баз данных, называемого SQL (язык структурированных запросов). Текст SQL-запроса располагается в сценариях страниц на стороне сервера либо в тегах.

Сервер приложений не может непосредственно получить данные из базы, поскольку базы данных используют специфические форматы хранения данных, в результате чего попытка получения таких данных будет напоминать попытку открытия документа Microsoft Word с помощью текстового редактора Notepad или BBEdit. Поэтому для подключения к базе данных сервер приложений использует посредника — драйвер базы данных. Драйвер базы данных представляет собой программный модуль, с помощью которого устанавливается взаимодействие между сервером приложений и базой данных.

После того как драйвер установит соединение, выполняется запрос к базе, в результате чего формируется набор записей. Набор записей представляет собой множество данных, полученных из одной или нескольких таблиц базы данных. Набор записей возвращается серверу приложений, который использует полученные данные для формирования страницы.

Следующий пример демонстрирует процесс выполнения запроса к базе данных и возвращение полученных данных браузеру.

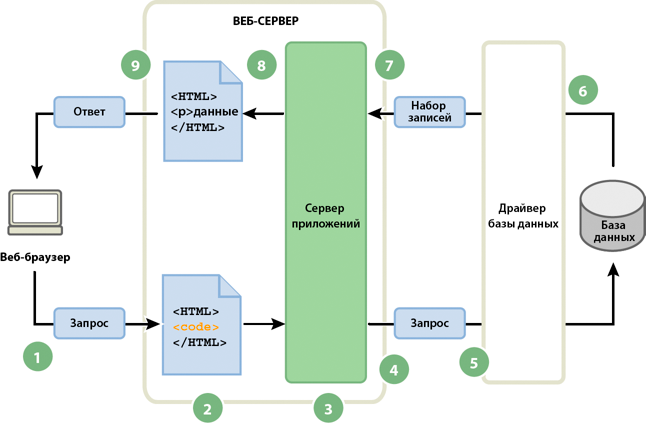


Рисунок 1.3 – Взаимодействие между клиентом и сервером при динамической обработке с использованием базы данных

1. Web-браузер запрашивает динамическую страницу.
2. Web-сервер находит страницу и передает ее серверу приложений.
3. Сервер приложений просматривает страницу на наличие инструкций и выполняет ее подготовку.
4. Сервер приложений отправляет запрос драйверу базы данных.
5. Драйвер выполняет запрос в базе данных.
6. Драйверу возвращается набор записей.
7. Драйвер передает набор записей серверу приложений
8. Сервер приложений вставляет данные в страницу и передает страницу web-серверу
9. Web-сервер отправляет подготовленную страницу запросившему ее браузеру.

Для использования в web-приложении пригодна любая база данных при условии, что на сервере установлен соответствующий драйвер базы данных.

Для создания малобюджетных приложений можно использовать файловую базу данных, например базу данных, созданную с помощью Microsoft Access. Если планируется создание надежных корпоративных приложений, рекомендуется использовать серверную базу данных, например, на основе серверов Microsoft SQL Server, Oracle 9i или MySQL.

Если база данных и web-сервер располагаются на разных компьютерах, следует обеспечить скоростное подключение между системами, поскольку от этого будет зависеть эффективность и скорость работы всего web-приложения.

1.4 Выводы по главе 1

Web-приложения являются удобным решенем при написании сайтов, содержимое каторых должно динамичски менятся в зависимости от изменений в базе данных, на основе которой функциониркет приложение, либо на основе действий пользователя, которые вносят изменения в базу данных или содержимое страницы. Использование базы данных в связке с динмической обработкой страниц позволяет конфигурировать страницы в соответствии с действиями пользователя и на основе информации, которая хранится в хранилище, и требований в представлении со стороны клиента. Таким образом форма web-приложения является наиболее оптимальной с учётом модели (использование множеством пользователей) и темы данной курсовой работы.

ГЛАВА 2  
ПРОЕКТИРОВАНИЕ WEB-ПРИЛОЖЕНИЯ

2.1 Архитектура сайта

Многослойная архитектура (n-tier client-server architecture) - клиент-серверная архитектура, где процессы представления, обработки и управления данными являются логически отделенными друг от друга процессами. Модель многослойной архитектуры помогает создать гибкое и многократно используемое программное обеспечение. В случае изменений надо их делать лишь в отдельных слоях, а не сразу во всем приложении. Это сулит меньше работы, меньших затрат времени и меньше потенциальных ошибок.

Более типичным и более используемым вариантом является трехслойная архитектура (three-tier client server architecture), которой и представоена настоящая курсовая работа. В случае трехслойного приложения каждый слой располагается в разных местах в компьютерной сети и может располагаться также ​​на разных платформах.

К пользователю ближе всего находящееся в его компьютере программное обеспечение рабочей станции (так называемый слой логики представления). Этот слой может отводиться формам ввода и типичному на данной платформе графическому пользовательскому интерфейсу. Не исключено существование этого слоя на различных платформах. Слой логики представления взаимодействует со слоем логики приложения (также слой бизнес-логики, средний слой).

Задача бизнес-логики (компании) - управлять функциональностью, обрабатывая для этого полученные с нижнего слоя данные в соответствии с запросами пользователя, пришедшими с верхнего слоя. Этот слой обычно находится на сервере локальной сети.

Слой данных - это третий слой. Он включает в себя базу данных и необходимое для ее управления программное обеспечение и может располагаться на некотором мейнфрейме. Слой отвечает среди прочего за сохранение данных независимо от приложения и логики представления.

В настоящее время у широкораспространенных веб-приложений, как правило, аналогичная архитектура. В этом случае слой логики представления связан с веб-браузером (веб-обозревателем) и с визуализацией пользовательского интерфейса (рендеринг) в окне браузера.

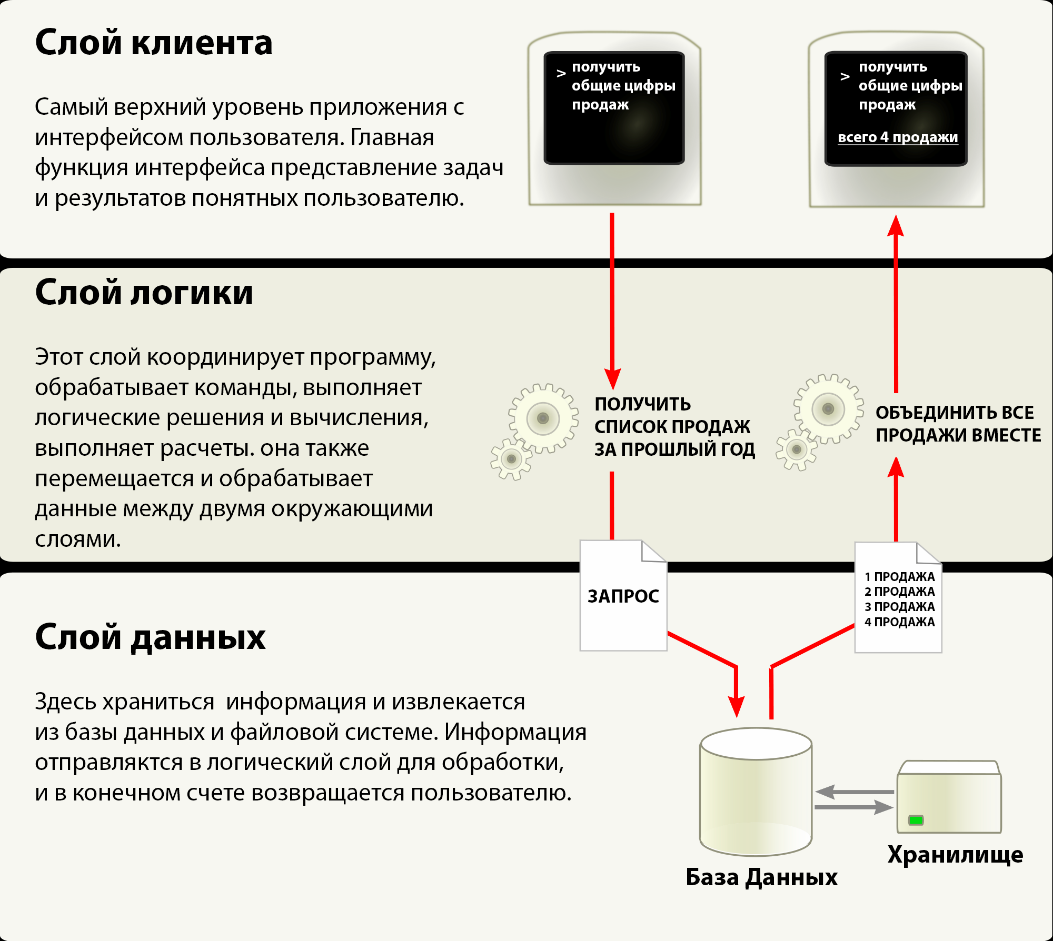


Рисунок 2.1 – Пример трехуровневой схемы/аритектуры web-приложения

Обобщением является многослойная архитектура (n-tier architcture). Здесь различные слои могут добавляться по мере необходимости (или описанные слои могут быть разделены, в свою очередь, на части). В ситуации, когда все большая ценность имеется не у одиночных приложений или баз данных, а у взаимодействующих информационных систем, узкое место в разработке переместилось в сторону создания интерфейсов и интеграции систем. Немаловажно использовать проверенные практики (например, шаблоны для разрешения аналогичных проблем), сервис-ориентированный подход, семантическое описание информационных систем.

2.2 Case диаграма

Определение базовых функций при построении архитектуры – процесс, который на первоначальном этапе кнтруирования каркаса приложения позволяет установить какие компоненты архитектуры программного обеспечения должны быть задействованы и включены в проект. Функционал разрабатываемой системы может дополнятся в случае появления новых требований, которые могут быть обусловлены необходимостью удовлетворять подребности дополнительных групп пользователей.

Вышесказанное подтверждает необходимость использования даиграмы прецедентов, также известной как диаграмма вариантов использования или же case-диаграма. Варианты использования предназначены в первую очередь для определения функциональных требований к системе и управляют всем процессом разработки. Все основные виды деятельности такие как анализ, проектирование, тестирование выполняются на основе вариантов использования. Во время анализа и проектирования варианты использования позволяют понять, как результаты, которые хочет получить пользователь, влияют на архитектуру системы и как должны себя вести компоненты системы, для того чтобы реализовать нужную для пользователя функциональность.

В процессе тестирования, описанные ранее варианты использования позволяют проще оценить точность реализации требований пользователей и позволяют провести пошаговую проверку этих требований.

Стратегия использования прецедентов при определении требований определяет необходимость дополнительно к вопросу "что пользователи ждут от системы?" задавать вопрос "что система должна сделать для конкретного пользователя?". Такой подход позволяет искать функции, которые нужны многим пользователям и исключать те возможности, которые не могут помочь пользователям выполнять свои повседневные задачи.

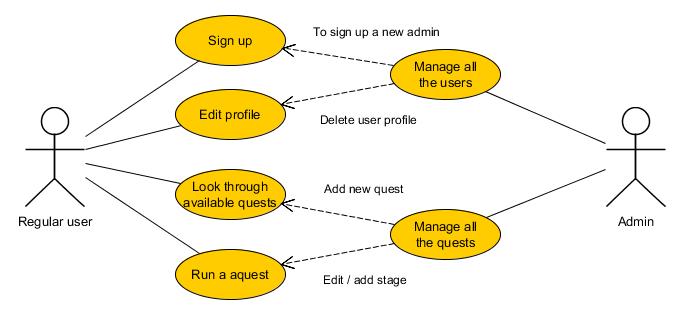


Рисунок 2.2 – Case-диаграмамма взаимодействия пользователя и администратора

2.3 Диаграмма состояний

При использовании web-приложения пользователь посредством интерфейсов промежуточного уровня архитектуры, взаимодействует с содержимым реляционной базы данных, оказывает влияние на её содержимое. Для обработки пользовательских воздействий и адекватного отклика в виде внесения изменений в хранилище, на этапе проектирования был составлен ряд диаграмм состояний, которые позволили получиьт наглядное представление того, как меняют своё представление те или инные объекты базы данных.

Главное назначение диаграммы состояний - описать возможные последовательности состояний и переходов, которые в совокупности характеризуют поведение моделируемой системы в течение всего ее жизненного цикла. Диаграмма состояний представляет динамическое поведение сущностей, на основе спецификации их реакции на восприятие некоторых конкретных событий. Системы, которые реагируют на внешние действия от других систем или от пользователей, иногда называют реактивными. Если такие действия инициируются в произвольные случайные моменты времени, то говорят об асинхронном поведении модели.

Диаграммы состояний чаще всего используются для описания поведения отдельных систем и подсистем. Они также могут быть применены для спецификации функциональности экземпляров отдельных классов, т.е. для моделирования всех возможных изменений состояний конкретных объектов. Диаграмма состояний по существу является графом специального вида, который служит для представления конечного автомата.

Диаграммы состояний могут быть вложены друг в друга, образуя вложенные диаграммы для более детального представления состояний отдельных элементов модели. Для понимания семантики конкретной диаграммы состояний необходимо представлять особенности поведения моделируемой сущности, а также иметь общие сведения из теории конечных автоматов.

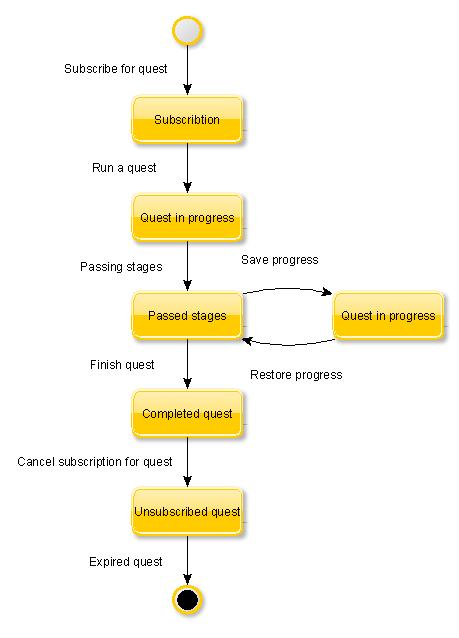


Рисунок 2.3 – Диаграмма состояний подписки пользователя на квест

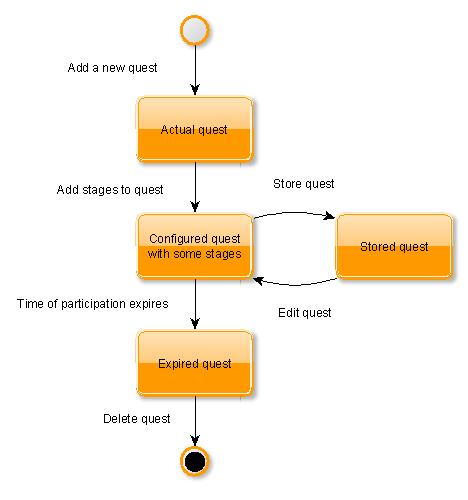


Рисунок 2.4 – Диаграмма состояний квеста

ГЛАВА 3  
РЕАЛИЗАЦИЯ WEB-ПРИЛОЖЕНИЯ

3.1 Модели бизнес-объектов

Разрабатываемое web-приложение представляет собой сайт с квестами, в которых может принимать участие пользователь, путём прохождения этапов. Прохождение предполагает, что пользователь отвечает на вопрос текущего этапа. Один квест может состоять из множества этапов. Исходя из вышеописанной логики потребовалось реализовать классы: пользователя, квеста и этапа (см. Листинг 2).

Класс пользователя является общим для администратора и игрока, т.к. на текущем этапе реализации проекта, по мнению разработчиков приложения, не имеет смысла выносить абстрактный класс и наследовать от него два класса: игрока и пользователя, что несомненно в будущем при расширении полномочий администратора потребует существенный дополнительный фугкционал. Отличить обычного пользвателя от администратора можно по соответствующему флагу в базе данных.

3.2 Util классы

Использование вспомогательных классов – повсеместная практика в программировании. Util-классы применяются для выполнения часто используемых процедур, что делает код чище, упрощает его визуальное восприятие, даёт программисту возможность уделять внимание не мелочам а сконцентрироваться непосредственно над реализацией поставленной задачи. В проекте было использовно два Util-класса: первый – DatabaseUtil – предоставляет возможность удобной работы с SqlDataReader, позволяет в одну строчку кода получить строку соединения с сервером базы данных (см. Листинг 3).

3.3 Использование lambda-выражений

При реализации проекта, а именно при написании динамически формируемых страниц активно использовался механизм C# 3.0 – lambda-выражения. Lambda-выражение наиболее прямо заменяет анонимный метод, представляет собой блок кода, заключенный в фигурные скобки это — прямая замена анонимных методов. Лямбда-выражения предоставляют еще более сокращенный способ объявлять анонимный метод и не требуют ни кода в фигурных скобках, ни оператора return. В итоге обработчики событий находятся непосредственно в том месте, где и ожидаешь их видеть: в месте оформления подписки элемента управления на событие (см. Листинг 4).

* 1. Модель базы данных

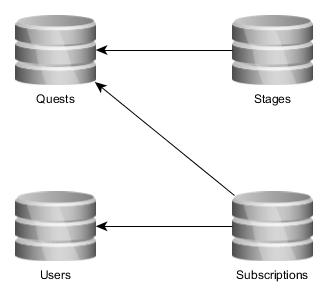
При написании работы невозможно было обойти необходимость использования реляционной базы данных для хранения информации о пользователях, квестах, этапах и подписках.

При выборе базы данных стоял выбор между базой данных Microsoft SQL Server и Oracle MySQL. Выбор был сделан в пользу решения компании Microsoft, которое является встроенным в среду разработки Microsoft Visual Studio Ultimate решением.

Также стоял вопрос использования платформы ADO.NET Entity Framework (EF) — программной модели, которая пытается заполнить пробел между конструкциями базы данных и объектно-ориентированными конструкциями, но пообная идея была отвергнута вследствии недостаточной компетенции в использовании баз данных. Используя EF, можно взаимодействовать с реляционными базами данных, не имея дело с кодом SQL (при желании). Исполняющая среда EF генерирует подходящие операторы SQL, при применении запросов LINQ к строго типизированным классам.

Подобная высокоуровневая «обёртка» помогает обстрагироваться от хранения и представления объектов в базе, но не даёт понимания процесса использования базы данных.

Таблица Stages связана с таблицей Quests внешним ключом, также при удалении квеста все связанные с ним этапы будут удалены. То же касается и связи таблиц Quests с Subscriptions, Users и Subscriptions (см. Листинг 5).



**Рисунок 2.4 – Диаграмма состояний квеста**

3.5 Проблема динамически добавляемых элементов управления

~~Как не бывает бочки мёда без ложки дёгтя, так и не бывает кода без костылей.~~ При реализации курсвой работы, команда столкнулась с довольно распространённой проблемой в программировании на ASP.NET: отсутствием отклика динамически добавленных элементов управления. Проблема проявлялсь в момент отправки страницы на сервер, в случае, когда пользователь воздействовал на динамически добавленный элемент управления (нажимал на кнопку, например). Сервер принимал страницу, и ряд блоков с проверкой if не давал элементам управления вновь добавиться на странцу, вследствие чего не происходил вызов соответствующих событий нажатия на кнопку. Проблема была решена лишь за счёт ~~прямых рук программиста~~ продолжительного тестирования кода и изучения схожих тем на сайте stackoverflow.

ЗАКЛЮЧЕНИЕ

Текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст.

Текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст. Текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Текст
2. Текст
3. Текст
4. Текст
5. текст

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ 1

Листинг 1 Пример статической web-страницы

<html>

<head>

<title>Trio Motors Information Page</title>

</head>

<body>

<h1>About Trio Motors</h1>

<p>Trio Motors is a leading automobile manufacturer.</p>

</body>

</html>

Листинг 2 Модели бизнес-объектов

public class UserModel

{

public int Id { get; set; }

public string NickName { get; private set; }

public string FirstName { get; private set; }

public string SecondName { get; private set; }

public string Password { get; private set; }

public long BirthDate { get; private set; }

public string AvatarPath { get; private set; }

public Sex Gender { get; private set; }

public bool IsLoggedIn { get; private set; }

public uint Balance { get; private set; }

public List<string> CompletedQuests { get; private set; }

public UserModel(string nickName, string firstName, string secondName,

string password, long birthDate, string avatarPath, Sex gender)

{

NickName = nickName;

FirstName = firstName;

SecondName = secondName;

Password = password;

BirthDate = birthDate;

AvatarPath = avatarPath;

Gender = gender;

Id = (nickName + firstName + secondName + birthDate.ToString()).GetHashCode();

}

public UserModel(int id, string nickName, string firstName, string secondName,

string password, long birthDate, string avatarPath, Sex gender)

{

Id = id;

NickName = nickName;

FirstName = firstName;

SecondName = secondName;

Password = password;

BirthDate = birthDate;

AvatarPath = avatarPath;

Gender = gender;

}

public UserModel(string nickName, string password, SqlCommand authCommand)

{

Id = (int)authCommand.Parameters[DatabaseConst.ParameterId].Value;

NickName = nickName;

FirstName = (string)authCommand.Parameters[DatabaseConst.ParameterFirstName].Value;

SecondName = (string)authCommand.Parameters[DatabaseConst.ParameterSecondName].Value;

Password = password;

BirthDate = (long)authCommand.Parameters[DatabaseConst.ParameterBirthDate].Value;

AvatarPath = DatabaseUtil.GetString(authCommand, DatabaseConst.ParameterAvatarPath);

Gender = (bool)authCommand.Parameters[DatabaseConst.ParameterGender].Value ? Sex.Female : Sex.Male;

}

public override string ToString()

{

return FirstName + " " + NickName + " " + SecondName;

}

}

public class QuestModel

{

public int Id { get; private set; }

public string Name { get; private set; }

public string Description { get; private set; }

public long StartDate { get; private set; }

public long ExpirationDate { get; private set; }

public bool Opened { get; private set; }

public List<UserModel> RegisteredCompetotors { get; private set; }

public QuestComplexityLevel ComplexityLevel { get; private set; }

public List<StageModel> Stages { get; set; }

public int LastStage { get; set; }

public QuestModel(int id, string name, string description, long startDate, long expirationDate, bool opened,

List<UserModel> registeredCompetotors, QuestComplexityLevel complexityLevel, List<StageModel> stages)

{

Id = id;

Name = name;

Description = description;

StartDate = startDate;

ExpirationDate = expirationDate;

ExpirationDate = expirationDate;

//calculate Opened value while performing request to database

Opened = opened;

RegisteredCompetotors = registeredCompetotors;

ComplexityLevel = complexityLevel;

Stages = stages;

}

public QuestModel(int id, string name, string description, long startDate, long expirationDate, QuestComplexityLevel complexityLevel)

{

Id = id;

Name = name;

Description = description;

StartDate = startDate;

ExpirationDate = expirationDate;

ExpirationDate = expirationDate;

ComplexityLevel = complexityLevel;

}

public QuestModel(SqlCommand selectCommad)

{

Id = (int)selectCommad.Parameters[DatabaseConst.ParameterId].Value;

Name = (string)selectCommad.Parameters[DatabaseConst.ParameterName].Value;

Description = (string)selectCommad.Parameters[DatabaseConst.ParameterDescription].Value;

StartDate = (long)selectCommad.Parameters[DatabaseConst.ParameterStartDate].Value;

ExpirationDate = (long)selectCommad.Parameters[DatabaseConst.ParameterExpirationDate].Value;

//Opened = (string)selectCommad.Parameters[PARAMETER\_AVATAR\_PATH].Value;

ComplexityLevel = (QuestComplexityLevel)selectCommad.Parameters[DatabaseConst.ParameterComplexityLevel].Value;

}

}

public class StageModel

{

public string Title { get; private set; }

public string Question { get; private set; }

public string ImagePath { get; private set; }

public string Answer { get; private set; }

public int Ordinal { get; private set; }

public StageModel(string title, string question, string imagePath, string answer, int ordinal)

{

Title = title;

Question = question;

ImagePath = imagePath;

Answer = answer;

Ordinal = ordinal;

}

}

Листинг 3 Util-классы

public static class DatabaseUtil

{

public static string GetConnectionString()

{

return WebConfigurationManager.ConnectionStrings["ConnectionString"].ConnectionString;

}

public static string GetString(SqlDataReader dataReader, int i)

{

return dataReader.IsDBNull(i) ? null : dataReader.GetString(i);

}

public static int GetInt32(SqlDataReader dataReader, int i)

{

return dataReader.IsDBNull(i) ? -1 : dataReader.GetInt32(i);

}

public static string GetString(SqlCommand sqlCommand, string field)

{

return Convert.IsDBNull(sqlCommand.Parameters[field].Value) ? null : (string)sqlCommand.Parameters[field].Value;

}

}

public static class FileUtil

{

public static string GetFileExtension(FileUpload fileUpload)

{

return System.IO.Path.GetExtension(fileUpload.FileName).ToLower();

}

}

Листинг 4 Использование lambda-выражений

editButton.Click += (sender, args) =>

{

int selectedStageOrdinal = int.Parse((sender as Button).ID);

foreach (StageModel stageModel in QuestModel.Stages)

{

if (stageModel.Ordinal == selectedStageOrdinal)

{

SelectedStageModel = stageModel;

break;

}

}

};

Листинг 5 SQL запросы создания таблиц

CREATE TABLE [dbo].[Quests] (

[Id] INT IDENTITY (1, 1) NOT NULL,

[name] NVARCHAR (MAX) NULL,

[description] NVARCHAR (MAX) NULL,

[start\_date] BIGINT NULL,

[expiration\_date] BIGINT NULL,

[opened] BIT NULL,

[complexity\_level] INT NULL,

PRIMARY KEY CLUSTERED ([Id] ASC)

);

CREATE TABLE [dbo].[Stages] (

[related\_quest] INT NOT NULL,

[title] NVARCHAR (MAX) NULL,

[question] NVARCHAR (MAX) NULL,

[image\_path] NVARCHAR (MAX) NULL,

[answer] NVARCHAR (MAX) NULL,

[ordinal] INT NOT NULL,

CONSTRAINT [StagePrimaryKey] PRIMARY KEY CLUSTERED ([related\_quest] ASC, [ordinal] ASC),

CONSTRAINT [Key\_Stages\_To\_Quests] FOREIGN KEY ([related\_quest]) REFERENCES [dbo].[Quests] ([Id]) ON DELETE CASCADE ON UPDATE CASCADE

);

CREATE TABLE [dbo].[Subscription] (

[questId] INT NOT NULL,

[userId] INT NOT NULL,

[last\_stage] INT NOT NULL,

CONSTRAINT [SubscriptionPrimaryKey] PRIMARY KEY CLUSTERED ([questId] ASC, [userId] ASC),

CONSTRAINT [Confirmed\_competitors\_To\_Users] FOREIGN KEY ([userId]) REFERENCES [dbo].[Users] ([Id]) ON DELETE CASCADE ON UPDATE CASCADE,

CONSTRAINT [Confirmed\_competitors\_To\_Quests] FOREIGN KEY ([questId]) REFERENCES [dbo].[Quests] ([Id]) ON DELETE CASCADE ON UPDATE CASCADE

);

CREATE TABLE [dbo].[Users] (

[Id] INT NOT NULL,

[nick\_name] NVARCHAR (MAX) NULL,

[first\_name] NVARCHAR (MAX) NULL,

[second\_name] NVARCHAR (MAX) NULL,

[password] NVARCHAR (MAX) NULL,

[birth\_date] BIGINT NULL,

[avatar\_path] NVARCHAR (MAX) NULL,

[gender] BIT NULL,

[admin] BIT DEFAULT ((0)) NULL,

CONSTRAINT [PK\_Users] PRIMARY KEY CLUSTERED ([Id] ASC)

);