Министерство образования Республики Беларусь
Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерного проектирования

Кафедра проектирования информационных компьютерных систем

Дисциплина "Объектно-ориентированное программирование"

К защите допустить:
Руководитель курсовой работы
старший преподаватель
кафедры
_____ А.В.Михалькевич
30.01.2025

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему

Разработка литературного сайта

БГУИР КР 1-40 05 01-10 № 198 ПЗ

Студент (подпись студента)

Курсовая работа
представлена на проверку
30.01.2025
_____
(подпись студента)

2025

Реферат

БГУИР КР 1-40 05 01-10 № 198 ПЗ, гр. 814302

, Разработка литературного сайта, Минск: БГУИР - 2025.

Пояснительная записка 75316 с., 0 рис., 0 табл.

Ключевые слова:

Предмет Объектно-ориентированное программирование, А.В.Михалькевич

Содержание

Введение

Сайты хранят в себе массу полезной и важной информации, они находятся в открытом доступе для любого пользователя, а потому так широко пользуются спросом на сегодняшний день. Каждый сайт ориентирован на группу пользователей, которые объединяются одними интересами и ищут информацию определенного характера, а если эта информация хранится в одном месте, собрана по крупицам и может дать развернутый ответ на вопрос, который ставит перед собой посетитель – такой сайт будет цениться вдвойне. Исходя из этого, можно с уверенностью предположить, что сайты создаются именно для сбора и хранения информации, которая находит своего конечного потребителя, а чем больше таких потребителей посещает сайт, тем большую выгоду из своего проекта, извлекает его владелец. Во всем мире превыше всего ценится качество, поэтому информация на сайтах обязательно должна быть качественной, интересной, развернутой и полезной. На какую бы тему ни был создан сайт, он должен раскрывать ее максимально обширно, от самого начала и до малейших мелочей. К примеру, если вы создали сайт на компьютерную тематику, не достаточно просто описать что такое сайт, персональный компьютер и для чего создаются сайты, расскажите о разработке сайтов в самых разных его проявлениях, тогда пользователи обязательно заинтересуются вашим проектом и посетят его снова и снова, зная, что с каждым днем, информационный объем вашего сайта будет пополняться чем-то интересным, и в то же время полезным для посетителей. Конечно, информацию можно черпать с различных источников, к примеру из книг, средств массовой информации, из слов специалистов, но ведь именно интернет и сайты, позволяют сохранять эту информацию для длительного пользования, делать закладки и возвращаться к необходимому источнику в любой момент. Именно это и делает разработку сайтов такой востребованной, а качественные сайты – успешными, приносящими доход своим владельцам. Сайтов существует невероятное количество, их невозможно сосчитать даже при огромном желании, ведь новые проекты запускаются ежедневно, но только единицы достигают поставленных целей, другие же, уходят в никуда. Почему это происходит – однозначно ответить нельзя, просто кто-то готов полностью отдаваться своему делу и вкладывать в него не

только свободное время, но также финансовые средства, другие же, останавливаются еще в самом начале пути, бессмысленно выжидая, когда сайт начнет приносить ему прибыль из пустоты. Если говорить об актуальности создания сайта, то можно утверждать что любой качественный проект будет актуален в своем информационном сегменте, если его довести до ума, работать над ним и постоянно наполнять качественным контентом. Ведь информация, которая хранится на этом сайте, если она полезна и ориентирована на конечного пользователя, обязательно найдет своих посетителей. Интернет – сравнительно молодой, инновационный источник информации, который только набирает свои масштабы во всем мире. С каждым днем все дальше распространяется интернет-покрытие, практически в каждой семье уже имеются интернет-проводники, такие как современные мобильные телефоны, персональные компьютеры, ноутбуки, планшеты и другие коммуникационные устройства. А это значит, что количество активных пользователей в сети-интернет также стремительно возрастает. Благодаря тому, что интернет является неисчерпаемым источником предоставления разнообразной информации, создание сайтов очень актуально, вне зависимости от выбранной тематики. Но стоит уделять внимание не столько созданию своего сайта, сколько его наполнению, потому что актуальным будет только качественный сайт, который пробьется в топ выдачи поисковой системы, заинтересует пользователя и принесет определенную выгоду своему владельцу. Таким образом, в данной курсовой работе была поставлена цель – научиться разрабатывать веб-сайты на примере разработки сайта, посвященного литературе.

1 Описание проекта

Данная курсовая работа представляет собой тематический сайт, посвященный литературе, который содержит в себе информацию (и, соответственно, выводит ее для пользователя) об авторах (в частности, даны их биографии) и их стихотворениях (разумеется, они разбиты по авторам). Также есть список всех стихотворений по годам (представлен в виде таблицы на странице, содержащей вкладку «стихотворения») и список тематических подборок книг, которые помогут пользователю определиться с тем, что же именно он желает прочитать.

Таким образом, для каждой статьи отображается ссылка для перехода к источнику информации и возможности более детального ознакомления с ней. Данный сайт обеспечивает пользователем уникальную возможность доступа к разнообразной информации, касающейся литературы.

Данный сайт был разработан при помощи языка PHP, с использованием языков разметки HTML и CSS, а также языка сценариев JavaScript [1][2].

Навигация по сайту довольно проста и интуитивно понятна.

При загрузке сайта пользователь попадает на главную страницу сайта, которая выглядит следующим образом:

Рисунок 1 – Главная страница

Меню есть сверху и сбоку – это сделано для дополнительного удобства пользователя. Также есть поле со случайной цитатой – при перезагрузке страницы она будет другой.

Когда пользователь зайдет во вкладку «Авторы», он увидит следующую веб-страницу:

Рисунок 2 – вкладка «Авторы»

Здесь приведен список авторов; каждое наименование является ссылкой, при нажатии на которую пользователь попадет на веб-страницу с биографией выбранного автора.

Рисунок 3 – веб-страница с биографией выбранного автора

Следующая вкладка «Стихотворения» также содержит в себе список ссылок: при нажатии на имя автора пользователь попадет на страницу с его стихотворениями.

Рисунок 4 – вкладка «Стихотворения»

Рисунок 5 – вкладка со стихотворениями выбранных авторов

Помимо этого на странице со вкладкой «Стихотворения» есть список всех стихотворений, имеющихся на сайте, с указанием автора и года написания стихотворения. Это выполнено в виде таблицы.

Следующая, последняя вкладка – «Подборки». Она содержит в себе перечень тематических подборок книг на самые разные темы. Перечень этот выполнен в виде ссылок.

Рисунок 6 – вкладка «Подборки»

Рисунок 7 – веб-страница, содержащая тематическую подборку книг

2 Обоснование выбора технологий

Событие в объектно-ориентированном программировании – это сообщение, которое возникает в различных точках исполняемого кода при выполнении определённых условий.

События предназначены для того, чтобы иметь возможность предусмотреть реакцию программного обеспечения.

Для решения поставленной задачи создаются обработчики событий: как только программа попадает в заданное состояние, происходит событие, посылается сообщение, а обработчик перехватывает это сообщение. В общем случае в обработчик не передаётся ничего, либо передаётся ссылка на объект, инициировавший (породивший) обрабатываемое событие. В особых случаях в обработчик передаются значения некоторых переменных или ссылки на какие-то другие объекты, чтобы обработка данного события могла учесть контекст возникновения события.

Самое простое событие – это событие, сообщающее о начале или о завершении некоторой процедуры. Событие, по сути, сообщает об изменении состояния некоторого объекта. Наиболее наглядно события представлены в пользовательском интерфейсе, когда каждое действие пользователя порождает цепочку событий, которые, затем обрабатываются в приложении.

В объектно-ориентированном анализе для описания динамического поведения объектов принято использовать модель состояний.

Иными словами, событие – это переход объекта из одного состояния в другое.

Взаимодействие объектов также осуществляется при помощи событий: изменение состояния одного объекта приводит к изменению состояния другого объекта, а событие оказывается средством связи между объектами.

Также событие можно определить как «абстракцию инцидента или сигнала в реальном мире, который сообщает нам о перемещении чего-либо в новое состояние».

Далее, выделяются четыре аспекта события:

– метка – это уникальный идентификатор события.

– значение – это текстовое сообщение о сути произошедшего.

– предназначение – представляет собой модель событий, которая принимает событие.

– данные – данные, которые переносятся от одного объекта к другому.

Первый ряд примеров событий доставляет собственно сам жизненный цикл объекта:

– создание объекта;

– уничтожение объекта.

Более сложные примеры событий возникают тогда, когда у объекта появляются внутренние состояния, которые описываются соответствующей диаграммой переходов (из одного состояния в другое). [3]

3 Инструментарий

1. **Обоснование используемых инструментов**

Для написания данной курсовой работы был использован текстовый редактор Notepad++, а также серверная платформа Open Server.

Notepad++ – это свободный текстовый редактор с открытым исходным кодом для Windows с подсветкой синтаксиса большого количества языков программирования и разметки, а также языков описания аппаратуры VHDL и Verilog. Поддерживает открытие более 100 форматов. Базируется на компоненте Scintilla, написан на C++ с использованием STL, а также Windows API и распространяется под лицензией GNU General Public License. Базовая функциональность программы может быть расширена как за счёт плагинов, так и сторонних модулей, таких как компиляторы и препроцессоры.

Базовые возможности этого текстового редактора (заявлены на официальном сайте):

– подсветка синтаксиса;

– сворачивание кода;

– автодополнение и автоматическое закрытие скобок и тэгов (если активировано);

– закладки;

– регулярные выражения для поиска и замены;

– запись и воспроизведение макросов;

– сравнение файлов;

– менеджер проектов;

– карта документа;

– переопределение любых горячих клавиш;

– резервное копирование сохраняемых файлов (включается в настройках);

– трансформация текста при помощи подключённого плагина TextFX;

– поддержка и конвертирование кодировок ANSI, UTF-8 и UCS-2;

– блоковое выделение текста, одновременное выделение нескольких разных мест (с Ctrl);

– многострочное редактирование (с использованием Alt);

А при установке дополнительных плагинов добавляются следующие возможности:

– шаблоны текста (сниппеты), вводимые с помощью сокращений (плагин SnippetPlus);

– FTP-менеджер (плагин NppFTP);

– Hex-редактор;

– автосохранение (при потере фокуса; через настраиваемый промежуток времени);

– проверка орфографии (с использованием GNU Aspell);

– симметричное и асимметричное шифрование текста (при установке плагина NppDarkCrypt);

– поддержка Zen Coding;

– поддержка автоматизации с помощью скриптов: Python, JScript, Lua, и других;

– поддержка сохранения в OneDrive и Dropbox;

Также в данном редакторе имеется функция подсветки синтаксиса для следующих языков программирования: ActionScript, Ada, ASN.1, ASP, Assembly, AutoIt, Скрипты AviSynth, BaanC, batch files, Blitz Basic, C, C#, C++, Caml, CMake, Cobol, CoffeeScript, Csound, CSS, D, Diff, Erlang, escript, Forth, Fortran, FreeBASIC, Gui4Cli, Haskell, HTML, ini-файлы, Intel HEX, скрипты Inno Setup, Java, JavaScript, JSON, JSP, KiXtart, LaTeX, LISP, Lua, Makefile, Matlab, MMIX, Nimrod, nnCron, скрипты NSIS, Objective-C, OScript, Pascal, Perl, PHP, PostScript, PureBasic, Python, R, Rebol, REG-файлы, Resource file, Ruby, Rust, Scheme, Shell script, Smalltalk, SPICE, SQL, Swift, S-Record, Tcl, Tektronix HEX, TeX, txt2tags, Visual Basic, Visual Prolog, VHDL, Verilog, XML, YAML.

Кроме того, пользователи могут задавать собственные правила подсветки и сворачивания для других языков, что очень удобно.[4]

Open Server — это портативный локальный WAMP/WNMP сервер, имеющий многофункциональную управляющую программу и большой выбор подключаемых компонентов. Это первый полноценный профессиональный инструмент, созданный специально для веб-разработчиков с учётом их рекомендаций и пожеланий.

Для отладки скриптов в различном окружении Open Server предлагает на выбор сразу два вида HTTP серверов, различные версии PHP и СУБД модулей, а так же возможность быстрого переключения между ними.

HTTP модули: Apache 2.2.21 и Nginx 1.0.11;

СУБД модули: MySQL 5.1.61, MySQL 5.5.20 и PostgreSQL 9.1.1;

PHP модули: PHP 5.2.17 (IMagick 2.2.1, Zend Optimizer 3.3.3, IonCube Loader 4.0.7, Memcache 2.2.4) и PHP 5.3.9 (IMagick 2.3.0, Xdebug 2.1.3, IonCube Loader 4.0.10, Memcache 2.2.6);

Набор инструментов: HeidiSQL, Adminer, PHPMyAdmin, PHPPgAdmin, PgAdmin.

В состав пакета так же включены Perl, FTP сервер, Sendmail, Memcached сервер.

Open Server — это целиком и полностью портативный сервер. В случае отсутствия на компьютере нужных системных компонентов Open Server установит их сам.

Безусловно, с помощью Open Server можно запустить/остановить сервер или открыть нужный домен. Однако также он имеет ряд специфических возможностей, которые делают Open Server по настоящему особенным:

– подробный просмотр логов всех компонентов в реальном времени;

– выбор HTTP, СУБД и PHP модулей в любом сочетании;

– поддержка SSL и кириллических доменов из коробки;

– поддержка алиасов или по другому доменных указателей, а так же удобная форма их настройки;

– создание локального поддомена без потери видимости основного домена в сети интернет;

– доступ к доменам (в один клик) и быстрый доступ к шаблонам конфигурации модулей;

– мультиязычный интерфейс (Русский, Украинский, Белорусский, Английский).[5]

1. **Использование системы контроля версий GIT**

Git – это распределённая система управления версиями. Проект был создан Линусом Торвальдсом для управления разработкой ядра Linux, первая версия выпущена 7 апреля 2005 года. На сегодняшний день его поддерживает Джунио Хамано. Среди проектов, использующих Git — ядро Linux, Swift, Android, Drupal, Cairo, GNU Core Utilities, Mesa, Wine, Chromium, Compiz Fusion, FlightGear, jQuery, PHP, NASM, MediaWiki, DokuWiki, Qt, а также ряд дистрибутивов Linux.

Программа является свободной и выпущена под лицензией GNU GPL версии 2. По умолчанию используется TCP порт 9418.

Как было сказано выше, все началось с разработки ядра для операционной системы Linux. Разработка велась на проприетарной системе BitKeeper, которую автор, – Ларри Маквой, сам разработчик Linux, — предоставил проекту по бесплатной лицензии. Разработчики, высококлассные программисты, написали несколько утилит, и для одной Эндрю Триджелл произвёл реверс-инжиниринг формата передачи данных BitKeeper. В ответ Маквой обвинил разработчиков в нарушении соглашения и отозвал лицензию, и Торвальдс взялся за новую систему: ни одна из открытых систем не позволяла тысячам программистов кооперировать свои усилия (тот же конфликт привёл к написанию Mercurial). Идеология была проста: взять подход CVS и перевернуть с ног на голову, и заодно добавить надёжности. Начальная разработка велась меньше, чем неделю: 3 апреля 2005 года разработка началась, и уже 7 апреля код Git управлялся неготовой системой. 16 июня Linux был переведён на Git, а 25 июля Торвальдс отказался от обязанностей ведущего разработчика.

Система спроектирована как набор программ, специально разработанных с учётом их использования в сценариях. Это позволяет удобно создавать специализированные системы контроля версий на базе Git или пользовательские интерфейсы. Например, Cogito является именно таким примером оболочки к репозиториям Git, а StGit использует Git для управления коллекцией исправлений (патчей).

Git поддерживает быстрое разделение и слияние версий, включает инструменты для

визуализации и навигации по нелинейной истории разработки. Как и Darcs, BitKeeper, Mercurial, Bazaar и Monotone[en], Git предоставляет каждому разработчику локальную копию всей истории разработки, изменения копируются из одного репозитория в другой.

Удалённый доступ к репозиториям Git обеспечивается git-демоном, SSH- или HTTP-сервером. TCP-сервис git-daemon входит в дистрибутив Git и является наряду с SSH наиболее распространённым и надёжным методом доступа. Метод доступа по HTTP, несмотря на ряд ограничений, очень популярен в контролируемых сетях, потому что позволяет использовать существующие конфигурации сетевых фильтров.

Ядро Git представляет собой набор утилит командной строки с параметрами. Все настройки хранятся в текстовых файлах конфигурации. Такая реализация делает Git легко портируемым на любую платформу и даёт возможность легко интегрировать Git в другие системы (в частности, создавать графические git-клиенты с любым желаемым интерфейсом).

Репозиторий Git представляет собой каталог файловой системы, в котором находятся файлы конфигурации репозитория, файлы журналов, хранящие операции, выполняемые над репозиторием, индекс, описывающий расположение файлов, и хранилище, содержащее собственно файлы. Структура хранилища файлов не отражает реальную структуру хранящегося в репозитории файлового дерева, она ориентирована на повышение скорости выполнения операций с репозиторием. Когда ядро обрабатывает команду изменения (неважно, при локальных изменениях или при получении патча от другого узла), оно создаёт в хранилище новые файлы, соответствующие новым состояниям изменённых файлов. Существенно, что никакие операции не изменяют содержимого уже существующих в хранилище файлов.

По умолчанию репозиторий хранится в подкаталоге с названием «.git» в корневом каталоге рабочей копии дерева файлов, хранящегося в репозитории. Любое файловое дерево в системе можно превратить в репозиторий git, отдав команду создания репозитория из корневого каталога этого дерева (или указав корневой каталог в параметрах программы). Репозиторий может быть импортирован с другого узла, доступного по сети. При импорте нового репозитория автоматически создаётся рабочая копия, соответствующая последнему зафиксированному состоянию импортируемого репозитория (то есть не копируются изменения в рабочей копии исходного узла, для которых на том узле не была выполнена команда commit).

Нижний уровень git является так называемой контентно-адресуемой файловой системой. Инструмент командной строки git содержит ряд команд по непосредственной манипуляции этим репозиторием на низком уровне. Эти команды не нужны при нормальной работе с git как с системой контроля версий, но нужны для реализации сложных операций (ремонт повреждённого репозитория и так далее), а также дают возможность создать на базе репозитория git своё приложение.

Для каждого объекта в репозитории вычисляется SHA-1-хеш, и именно он становится именем файла, содержащего данный объект в каталоге .git/objects. Для оптимизации работы с файловыми системами, не использующими деревья для каталогов, первый байт хеша становится именем подкаталога, а остальные – именем файла в нём, что снижает количество файлов в одном каталоге (ограничивающий фактор производительности на таких устаревших

файловых системах).

Все ссылки на объекты репозитория, включая ссылки на один объект, находящийся внутри другого объекта, являются SHA-1-хешами.

Кроме того, в репозитории существует каталог refs, который позволяет задать читаемые человеком имена для каких-то объектов Git. В командах Git оба вида ссылок – читаемые человеком из refs, и нижележащие SHA-1 — полностью взаимозаменяемы.

В классическом обычном сценарии в репозитории git есть три типа объектов – файл, дерево и «коммит» (англ. commit – фиксация). Файл есть какая-то версия какого-то пользовательского файла, дерево – совокупность файлов из разных подкаталогов, «коммит» – дерево и некая дополнительная информация (например, родительские коммиты, а также комментарий).

В репозитории иногда производится сборка мусора, во время которой устаревшие файлы заменяются на «дельты» между ними и актуальными файлами (то есть, актуальная версия файла хранится неинкрементально, инкременты используются только для возврата к предыдущим версиям), после чего данные «дельты» складываются в один большой файл, к которому строится индекс. Это снижает требования по ёмкости хранения.

Репозиторий Git бывает локальный и удалённый. Локальный репозиторий – это подкаталог .git, создаётся (в пустом виде) командой git init и (в непустом виде с немедленным копированием содержимого родительского удалённого репозитория и простановкой ссылки на родителя) командой git clone.

Практически все обычные операции с системой контроля версий, такие, как коммит и слияние, производятся только с локальным репозиторием. Удалённый репозиторий можно только синхронизировать с локальным как «вверх» (push), так и «вниз» (pull).

Наличие полностью всего репозитория проекта локально у каждого разработчика даёт Git ряд преимуществ перед SVN. Так, например, все операции, кроме push и pull, можно осуществлять без наличия интернет-соединения.

Очень мощной возможностью git являются ветви, реализованные куда более полно, чем в SVN: по сути, ветвь git есть не более чем именованная ссылка, указывающая на некий коммит в репозитории (используется подкаталог refs). Коммит без создания новой ветви всего лишь передвигает эту ссылку на себя, а коммит с созданием ветви – оставляет старую ссылку на месте, но создаёт новую на новый коммит, и объявляет её текущей. Заменить локальные девелоперские файлы на набор файлов из иной ветви, тем самым перейдя к работе с ней – так же тривиально.

Также поддерживаются субрепозитории с синхронизацией текущих ветвей в них.

Команда push передаёт все новые данные (те, которых ещё нет в удалённом репозитории) из локального репозитория в репозиторий удалённый. Для исполнения этой команды необходимо, чтобы удалённый репозиторий не имел новых коммитов в себя от других клиентов, иначе push завершается ошибкой, и придётся делать pull и слияние.

Команда pull – обратна команде push. В случае, если одна и та же ветвь имеет независимую историю в локальной и в удалённой копии, pull немедленно переходит к слиянию.

Слияние в пределах разных файлов осуществляется автоматически (всё это поведение настраивается), а в пределах одного файла – стандартным трёхпанельным сравнением файлов.

После слияния нужно объявить конфликты как разрешённые.

Результатом всего этого является новое состояние в локальных файлах у того разработчика, что осуществил слияние. Ему нужно немедленно сделать коммит, при этом в данном объекте коммита в репозитории окажется информация о том, что коммит есть результат слияния двух ветвей и имеет два родительских коммита.

Кроме слияния, Git поддерживает ещё операцию перемещения. Эта операция есть получение набора всех изменений в ветви A, с последующим их «накатом» на ветвь B. В результате ветвь B продвигается до состояния AB. В отличие от слияния, в истории ветви AB не останется никаких промежуточных коммитов ветви A (только история ветви B и запись о самом rebase, это упрощает интеграцию крупных и очень крупных проектов).

Также Git имеет временный локальный индекс файлов. Это – промежуточное хранилище между собственно файлами и очередным коммитом (коммит делается только из этого индекса). С помощью этого индекса осуществляется добавление новых файлов (git add добавляет их в индекс, они попадут в следующий коммит), а также коммит не всех изменённых файлов (коммит делается только тем файлам, которым был сделан git add). После git add можно редактировать файл далее, получатся три копии одного и того же файла – последняя, в индексе (та, что была на момент git add), и в последнем коммите.

Имя ветви по умолчанию: master. Имя удалённого репозитория по умолчанию, создаваемое git clone во время типичной операции «взять имеющийся проект с сервера себе на машину»: origin.

Таким образом, в локальном репозитории всегда есть ветвь master, которая есть последний локальный коммит, и ветвь origin/master, которая есть последнее состояние удалённого репозитория на момент завершения исполнения последней команды pull или push.

Команда fetch (частичный pull) – берёт с удалённого сервера все изменения в origin/master, и переписывает их в локальный репозиторий, продвигая метку origin/master.

Если после этого master и origin/master разошлись в стороны, то необходимо сделать слияние, установив master на результат слияния (команда pull есть fetch+merge). Далее возможно сделать push, отправив результат слияния на сервер и установив на него origin/master.

В Windows-версии (официальная Windows-версия называется mSysGit) используется пакет mSys – порт POSIX-совместимой командной строки под Windows из проекта MinGW. Под mSys перенесены все необходимые для Git библиотеки и инструменты, а также сам Git. При работе с удалёнными репозиториями по протоколу SSL используется хранилище сертификатов из mSys, а не из Windows.

Существует немало графических оболочек для Git для Windows, например, TortoiseGit. Все они реализованы через вызовы mSysGit и требуют его установки на машину. Не исключение и SourceTree, решение компании Atlassian, но mSysGit оно содержит внутри себя, что имеет свои плюсы и минусы (так установка в глубокий подкаталог затрудняет добавление в mSys нужных SSL-сертификатов).

Поскольку в Windows используется отличный от большинства Unix-подобных систем символ конца строки, для работы коллективов, использующих разные операционные системы,

предусматриваются параметры (как для клиентов, так и уровня репозитория), обеспечивающие унифицированное представление конца строки.

Git использует сеть только для операций обмена с удалёнными репозиториями.

Возможно использование следующих протоколов:

– git-протокол (схема URI — git:) – открытый протокол, требующий наличия на сервере запущенного git-домена (поставляется вместе с Git), протокол не имеет средств аутентификации пользователей;

– SSH (ssh:) – использует аутентификацию пользователей с помощью пар ключей, а также встроенный в Unix-систему «основной» SSH-сервер (sshd), со стороны сервера требуется создание учётных записей с git в качестве оболочки;

– HTTP и HTTPS (http:, https:) – использует инструмент curl (для Windows – поставляется вместе с git), и его возможности HTTP-аутентификации, как и его поддержку SSL и сертификатов.

В последнем случае требуется работа на серверной стороне веб-приложения, исполняющего роль прослойки между командами Git на сервере и HTTP-сервером (среди таковых WebGitNet, разработанный на ASP.NET MVC 4). Кроме поддержки серверной стороны команд push и pull, такие веб-приложения могут также давать доступ только на чтение к репозиторию через веб-браузер.

Разработано множество графических интерфейсов для системы, среди них – GitKraken (кроссплатформенный условно бесплатный клиент Git), SmartGit (кроссплатформенный интерфейс на Java), gitk (простая и быстрая программа, написана на Tcl/Tk, распространяемая с самим Git), Giggle (вариант на Gtk+), TortoiseGit (интерфейс, реализованный как расширение для проводника Windows), SourceTree (бесплатный Git-клиент для Windows и Mac), Github-клиент и ряд других.

Кроме того, разработано множество веб-фронтендов, в числе которых GitWebAdmin, GitLab, Gitblit, Gerrit, Gitweb, Kallithea, Gitea.

Ряд сервисов предоставляют хостинг для git-репозиториев, среди наиболее известных – GitHub, Codebase, SourceForge, SourceHut, Gitorious, Bitbucket, GitLab.

В стандартной поставке Git поддерживается взаимодействие с CVS (импорт и экспорт, эмуляция CVS-сервера) и Subversion (частичная поддержка импорта и экспорта). Стандартный инструмент импорта и экспорта внутри экосистемы – архивы серий версионированных файлов в форматах .tar.gz и .tar.bz2.

Ссылка на онлайн-репозиторий GitHub с данной курсовой работой: https://github.com/AnnGerwel/World_of_Verses.

4 Архитектурный шаблон проектирования MVC

Шаблон проектирования MVC (Model-View-Controller, Модель-Представление-Контроллер) — это шаблон программной архитектуры, построенный на основе сохранения представления данных отдельно от методов, которые взаимодействуют с данными.

Не смотря на то, что схема MVC была первоначально разработана для персональных

компьютеров, она была адаптирована и широко используется веб-разработчиками из-за точного разграничения задач и возможности повторного использования кода. Схема стимулирует развитие модульных систем, что позволяет разработчикам быстро обновлять, добавлять или удалять функционал. [6]

Иными словами, шаблон MVC описывает простой способ построения структуры приложения, целью которого является отделение бизнес-логики от пользовательского интерфейса. В результате, приложение легче масштабируется, тестируется, сопровождается и конечно же реализуется.

Идеи MVC сформулировал Трюгве Реенскауг во время работы в Xerox PARC в конце 70-х годов. В те времена для работы с ЭВМ было не обойтись без ученой степени и постоянного изучения объемной документации.

Задача, которую Реенскауг решал совместно с группой очень сильных разработчиков, заключалась в том, чтобы упростить взаимодействие рядового пользователя с компьютером. Необходимо было создать средства, которые с одной стороны были бы предельно простыми и понятными, а с другой — давали бы возможность управлять компьютером и сложными приложениями. Реенскауг работал в команде, которая занималась разработкой портативного компьютера "для детей всех возрастов" — Dynabook, а также языка SmallTalk под руководством Алана Кея.

Именно тогда и там закладывались понятия дружелюбного интерфейса. Работа Реенскауга совместно с командой во многом повлияла на развитие сферы IT.

Проект, над которым работал Реннскауг велся на протяжении 10 лет; первая публикация об MVC от его создателей вышла в свет еще через 10 лет. Мартин Фаулер, автор ряда книг и статей по архитектуре ПО, упоминает, что он изучал MVC по работающей версии SmallTalk.

Поскольку информации об MVC из первоисточника долго не было, а также по ряду других причин, появилось большое количество различных трактовок этого понятия.

В результате многие считают MVC схемой или шаблоном проектирования. Реже MVC называют составным шаблоном или комбинацией нескольких шаблонов, работающих совместно для реализации сложных приложений.

Но на самом деле, как было сказано ранее, MVC — это прежде всего набор архитектурных идей/принципов/подходов, которые можно реализовать различными способами с использованием различных шаблонов. [7]

Концептуальная схема шаблона MVC:

Рисунок 8 – модель MVC

В архитектуре MVC модель предоставляет данные и правила бизнес-логики, представление отвечает за пользовательский интерфейс, а контроллер обеспечивает взаимодействие между моделью и представлением.

Типичную последовательность работы MVC-приложения можно описать следующим

образом:

1. При заходе пользователя на веб-ресурс, скрипт инициализации создает экземпляр приложения и запускает его на выполнение.

   При этом отображается вид, скажем главной страницы сайта.

1. Приложение получает запрос от пользователя и определяет запрошенные контроллер и действие. В случае главной страницы, выполняется действие по умолчанию (index).
2. Приложение создает экземпляр контроллера и запускает метод действия, в котором, к примеру, содержаться вызовы модели, считывающие информацию из базы данных.
3. После этого, действие формирует представление с данными, полученными из модели и выводит результат пользователю.

Стоит также отметить, что реализация шаблона MVC может отличаться в зависимости от задачи. Например, в веб-разработке модель и вид взаимодействуют друг с другом через контроллер, а в приложениях модель может сама уведомлять вид, что нужно что-то изменить.[8]

Как было сказано выше, шаблон MVC включает в себя три составные части: модель, контроллер и представление (или же вид).ё

Модель содержит бизнес-логику приложения и включает методы выборки (это могут быть методы ORM), обработки (например, правила валидации) и предоставления конкретных данных, что зачастую делает ее очень толстой, что вполне нормально.

Модель не должна напрямую взаимодействовать с пользователем. Все переменные, относящиеся к запросу пользователя должны обрабатываться в контроллере.

Модель не должна генерировать HTML или другой код отображения, который может изменяться в зависимости от нужд пользователя. Такой код должен обрабатываться в видах.

Одна и та же модель, например: модель аутентификации пользователей может использоваться как в пользовательской, так и в административной части приложения. В таком случае можно вынести общий код в отдельный класс и наследоваться от него, определяя в наследниках специфичные для подприложений методы.

Вид используется для задания внешнего отображения данных, полученных из контроллера и модели.

Виды содержат HTML-разметку и небольшие вставки PHP-кода для обхода, форматирования и отображения данных. Они не должны напрямую обращаться к базе данных. Этим должны заниматься модели. Также они не должны работать с данными, полученными из запроса пользователя. Эту задачу должен выполнять контроллер.

Вид может напрямую обращаться к свойствам и методам контроллера или моделей, для получения готовых к выводу данных.

Виды обычно разделяют на общий шаблон, содержащий разметку, общую для всех страниц (например, шапку и подвал) и части шаблона, которые используют для отображения данных выводимых из модели или отображения форм ввода данных.

Контроллер – это связующее звено, соединяющее модели, виды и другие компоненты в рабочее приложение. Контроллер отвечает за обработку запросов пользователя. Контроллер

не должен содержать SQL-запросов, их лучше держать в моделях. Также контроллер не должен содержать HTML и другой разметки – её стоит выносить в виды.

В хорошо спроектированном MVC-приложении контроллеры обычно очень тонкие и содержат только несколько десятков строк кода. Чего, не скажешь о Stupid Fat Controllers (SFC) в CMS Joomla. Логика контроллера довольно типична и большая ее часть выносится в базовые классы.

Модели, наоборот, очень толстые и содержат большую часть кода, связанную с обработкой данных, т.к. структура данных и бизнес-логика, содержащаяся в них, обычно довольно специфична для конкретного приложения.

Именно контроллер является самым главным элементом, с которого все начинается и на котором все, как правило, заканчивается, в модели MVC.

Задача контроллера – принять запрос пользователя и решить, что делать с этим запросом далее, если требуется, то перенаправить запрос в модель, который обработает информацию и возвращает ее контроллеру.

После того, как информация обработана, контроллер решает, что с ней делать дальше, если пользователю достаточно предоставить просто набор каких-то данных, не в виде html-странице, а например, в формате json, контроллер этот набор данных ему выдает.

Если необходимо сформировать html-страницу, контроллер передает эти данные в вид и внутри вида шаблонизатор формирует каркас страницы, выдает ее назад контроллеру и контроллер уже выдает этот каркас пользователю в виде html-страницы.

Из 3 частей MVC модели контроллер является обязательной частью. Остальные части являются опциональными. Если пользователю достаточно только отдать какой-то набор данных, то можно обойтись без вида. Если не нужно обрабатывать данные, то можно обойтись без модели. [9]

5 Разработка базы данных и системы управления базой данных

Безусловно, для правильного функционирования сайта нужны не только файлы с кодом страниц, но и базы данных. Для взаимодействия с базами данных используются системы управления базами данных (СУБД).

База данных (БД) – это совокупность массивов и файлов данных, организованная по определённым правилам, предусматривающим стандартные принципы описания, хранения и обработки данных независимо от их вида. Также понятие базы данных можно определить так: это совокупность организованной информации, относящейся к определённой предметной области, предназначенная для длительного хранения во внешней памяти компьютера и постоянного применения.

Иными словами, база данных представляет собой определенный набор данных, которые, как правило, связаны объединяющим признаком либо свойством (их может быть несколько). Эти данные упорядочены, например, по алфавиту. Обилие различных данных, которые могут быть помещены в единую базу, ведет к множеству вариаций того, что может быть записано: личные

данные пользователей, записи, даты, заказы и так далее.

В первую очередь это удобно тем, что информацию можно быстро заносить в базу данных и так же быстро ее извлекать при необходимости. Если на заре развития web-разработки все необходимые данные нужно было прописывать в коде страницы, то теперь такая необходимость отсутствует – нужная информация может быть запрошена из базы данных при помощи скриптов. Специальные алгоритмы хранения и поиска информации, которые используются в базах данных, позволяют находить нужные сведения буквально за доли секунд – а при работе в виртуальном пространстве скорость работы ресурса важна как ничто другое.

Немаловажной является и взаимосвязь информации в базе данных: изменение одной строчки может привести к значительным изменениям других строк. Работать с данными таким образом гораздо проще и быстрее, чем, если бы изменения касались только одного места в базе данных.

Одно из основных свойств БД – это независимость данных от программы, использующих эти данные.

Для разработки программ, систем программ, работающих с базами данных, используются специальные средства – системы управления базами данных (СУБД).

СУБД включает, как правило, специальный язык программирования и все прочие средства, необходимые для разработки указанных программ.

В настоящее время наиболее известными СУБД являются: Oracle Database, MS SQL Server, MySQL (MariaDB) и ACCESS. Последняя входит в состав профессионального офисного пакета Microsoft Office. Это современные системы с большими возможностями, предназначенные для разработки сложных программных комплексов, и знакомство с ними для пользователя ЭВМ исключительно полезно, но в рамках настоящего пособия осуществить его затруднительно.

Иными словми, главная функция СУБД – это управление данными. СУБД обязательно поддерживает языки баз данных, а также отвечает за копирование и восстановление данных после каких-либо сбоев.

Классифицировать базы данных можно по разным признакам. К примеру, можно разделить базы по модели данных: иерархические (они имеют древовидную структуру), сетевые (они по своей структуре похожи на иерархические), реляционные (они используются для управления реляционными базами данных), объектно-ориентированные (эти используются для объектной модели данных) или же объектно-реляционные (представляют собой некое слияние реляционного и объектно-ориентированного вида баз данных).

Также их можно классифицировать по тому, где размещается СУБД. Здесь можно выделить локальные – вся СУБД размещается на одном компьютере, и распределенные – части системы управления базами данных находятся на нескольких компьютерах.

СУБД, которые можно разделить по способу доступа к базам данных: файл-серверные, клиент-серверные и встраиваемые. Файл-серверные СУБД на данный момент уже считаются устаревшими; в основном идет использование клиент-серверных (СУБД, которые располагаются на сервере вместе с самой базой данных) и встраиваемых (не требующих отдельной установки) систем.

Что касается информации, которая хранится в базах данных, то она не ограничивается

только текстовыми или графическими файлами – современные версии СУБД поддерживают также форматы аудио и видеофайлов.

СУБД, помимо основной своей функции – хранения и систематизации огромного количества информации – позволяют быстро обрабатывать клиентские запросы и выдавать свежую и актуальную информацию.

Реляционные и объектно-реляционные СУБД являются одними из самых распространенных систем. Они представляют собой таблицы, у которых каждый столбец упорядочен и имеет определенное уникальное название. Последовательность строк определяется последовательностью ввода информации в таблицу. При этом обрабатывание столбцов и строк может происходить в любом порядке. Таблицы с данными связаны между собой специальными отношениями, благодаря чему с данными из разных таблиц можно работать – к примеру, объединять их – при помощи одного запроса.

Для управления реляционными базами данных применяется особый язык программирования – SQL (Structured query language, или же Язык структурированных запросов).

Команды, которые используются в SQL, делятся на те, которые манипулируют данными, те, которые определяют данные, и те, которые управляют данными.

Схема работы с базой данных представлена на рисунке 9:

Рисунок 9 – Схема работы с базой данныхН

Заключение

В заключение хотелось бы сказать: в современном мире актуальность создания сайтов самых разных видов безусловно очевидна: с появлением глобальной сети каждый человек получил интерактивный инструмент, позволяющий сообщить миру об услугах и товарах компании, привлечь единомышленников и покупателей; а также позволяющий получать информацию именно с того тематического сайта, который тебе интересен. На литературном сайте World_of_Verses пользователи могут прочитать стихотворения авторов, увидеть таблицу, в которой собраны все стихотворения с годом их написания, а также просмотреть информацию о биографии авторов, стихотворения которых есть на данном ресурсе. Помимо этого есть тематические подборки книг, которые могут помочь пользователям определиться, что же прочитать. Таким образом, в данной курсовой работе были рассмотрены актуальные вопросы разработки и создания современного Web-сайта. Задача была реализована при помощи шаблона MVC, языка разметки html и языка описания внешнего вида CSS, а также языка сценариев JavaScript. Также данный проект был занесён на онлайн-репозиторий GitHub.

Список использованных источников

Приложения

1. [Приложение] **Приложение** 5ef9dd0375194_Приложение А.docx