Оглавление

[Введение 2](#_Toc322720347)

[Постановка задачи 3](#_Toc322720348)

[1. Исследовательская часть 4](#_Toc322720349)

[1.1. Анализ аналогичных программных продуктов 4](#_Toc322720350)

[1.1.1. Плагин для WordPress – Simpleforum 4](#_Toc322720351)

[1.1.2. Simple Machine Forum (SMF) 5](#_Toc322720352)

[1.1.3. phpBB 7](#_Toc322720353)

[1.1.4 Выводы 8](#_Toc322720354)

[1.2. Анализ технологий 9](#_Toc322720355)

[Система контроля версий 9](#_Toc322720356)

[1.3. Анализ программных средств разработки 21](#_Toc322720357)

[Выбор инструментальных средств 21](#_Toc322720358)

[Выбор способа хранения данных 22](#_Toc322720359)

[Сервер Apache 24](#_Toc322720360)

[Netbeans IDE 25](#_Toc322720361)

[Средство отладки Opera Dragonfly 26](#_Toc322720362)

[Клиент GitHub - TortoiseGit 30](#_Toc322720363)

[Sql сервер MySQL 31](#_Toc322720364)

[Шифрование MD5 32](#_Toc322720365)

[1.4. Архитектура приложения 34](#_Toc322720366)

[Конструкторская часть 35](#_Toc322720367)

[Проектирование программных модулей 35](#_Toc322720368)

[Модуль регистрации 35](#_Toc322720369)

[Редактирование данных пользователя 40](#_Toc322720370)

# Введение

На сегодняшний день

# Постановка задачи

# 1. Исследовательская часть

## 1.1. Анализ аналогичных программных продуктов

### 1.1.1. Плагин для WordPress – Simpleforum

WordPress — система управления содержимым сайта (CMS) с открытым исходным кодом, распространяемая под GNU GPL. Написана на PHP, в качестве базы данных использует MySQL. Сфера применения — от блогов до достаточно сложных новостных ресурсов и даже интернет-магазинов. Встроенная система «тем» и «плагинов» вместе с удачной архитектурой позволяет конструировать практически любые проекты.

В качестве рассматриваемого форума выступает плагин Simpleforum.   


Рис. 1. Список категорий Simpleforum

### 1.1.2. Simple Machine Forum (SMF)

Simple Machines Forum (SMF) — бесплатный интернет-форум, написанный на PHP с использованием базы данных MySQL.

У SMF есть базовый репозиторий для бесплатного хранения и отслеживания модификаций через сайт Simple Machines. Множество модификаций, также известных как «моды», создаются и распространяются на бесплатной основе, включая видеоигры, дополнения для профилей, одобрение сообщений модератором, гороскоп, и многие другие, находящиеся в разработке.

Менеджер пакетов, включённый в SMF — одно из его главных достоинств. Он позволяет администратору устанавливать модификации и обновления для SMF без необходимости изменять исходный код скриптов, обычно при помощи всего нескольких кликов мышью.



Рис. 2. Главный вид SMF



Рис. 3. Список тем



Рис. 4. Список групп пользователей в админпанели

Достоинства данного программного продукта:

* Богатый функционал
* Множество настроек в админпанели

Недостатки:

* Перегруженность лишней информацией
* Отсутствие русского языка

### 1.1.3. phpBB

phpBB (PHP Bulletin Board) — популярный бесплатный веб-форум со свободным исходным кодом, разработанный на скриптовом языке PHP, поддерживающий различные СУБД, включая MySQL, PostgreSQL, MS SQL Server, MS Access, SQLite, а также Oracle (при наличии необходимых изменений).

Кроме поддержки различных СУБД достоинствами phpBB являются:

* Несложная в использовании система шаблонов.
* Многоязычный интерфейс: языковые файлы переведены на более чем 50 языков и доступны для свободного скачивания с официального сайта phpBB.
* Большое сообщество пользователей, готовых прийти на помощь.
* Большое количество доступных и обновляемых модификаций.

Рис. 5. Список тем форума phBB

Недостатки: изобилие лишней анимации и очень яркие цвета

### 1.1.4 Выводы

## 1.2. Анализ технологий

### Система контроля версий

Ситуация, в которой электронный документ за время своего существования претерпевает ряд изменений, достаточно типична. При этом часто бывает важно иметь не только последнюю версию, но и несколько предыдущих. В простейшем случае можно просто хранить несколько вариантов документа, нумеруя их соответствующим образом. Такой способ неэффективен (приходится хранить несколько практически идентичных копий), требует повышенного внимания и дисциплины и часто ведёт к ошибкам, поэтому были разработаны средства для автоматизации этой работы.

Традиционные системы управления версиями используют централизованную модель, когда имеется единое хранилище документов, управляемое специальным сервером, который и выполняет большую часть функций по управлению версиями. Пользователь, работающий с документами, должен сначала получить нужную ему версию документа из хранилища; обычно создаётся локальная копия документа, т. н. «рабочая копия». Может быть получена последняя версия или любая из предыдущих, которая может быть выбрана по номеру версии или дате создания, иногда и по другим признакам. После того, как в документ внесены нужные изменения, новая версия помещается в хранилище. В отличие от простого сохранения файла, предыдущая версия не стирается, а тоже остаётся в хранилище и может быть оттуда получена в любое время. Сервер может использовать т. н. дельта-компрессию — такой способ хранения документов, при котором сохраняются только изменения между последовательными версиями, что позволяет уменьшить объём хранимых данных. Поскольку обычно наиболее востребованной является последняя версия файла, система может при сохранении новой версии сохранять её целиком, заменяя в хранилище последнюю ранее сохранённую версию на разницу между этой и последней версией. Некоторые системы (например, ClearCase) поддерживают сохранение версий обоих видов: большинство версий сохраняется в виде дельт, но периодически (по специальной команде администратора) выполняется сохранение версий всех файлов в полном виде; такой подход обеспечивает максимально полное восстановление истории в случае повреждения репозитория.

Иногда создание новой версии выполняется незаметно для пользователя (прозрачно), либо прикладной программой, имеющей встроенную поддержку такой функции, либо за счёт использования специальной файловой системы. В этом случае пользователь просто работает с файлом, как обычно, и при сохранении файла автоматически создаётся новая версия.

Часто бывает, что над одним проектом одновременно работают несколько человек. Если два человека изменяют один и тот же файл, то один из них может случайно отменить изменения, сделанные другим. Системы управления версиями отслеживают такие конфликты и предлагают средства их решения. Большинство систем может автоматически объединить (слить) изменения, сделанные разными разработчиками. Однако такое автоматическое объединение изменений, обычно, возможно только для текстовых файлов и при условии, что изменялись разные (непересекающиеся) части этого файла. Такое ограничение связано с тем, что большинство систем управления версиями ориентированы на поддержку процесса разработки программного обеспечения, а исходные коды программ хранятся в текстовых файлах. Если автоматическое объединение выполнить не удалось, система может предложить решить проблему вручную.

Часто выполнить слияние невозможно ни в автоматическом, ни в ручном режиме, например, если формат файла неизвестен или слишком сложен. Некоторые системы управления версиями дают возможность заблокировать файл в хранилище. Блокировка не позволяет другим пользователям получить рабочую копию или препятствует изменению рабочей копии файла (например, средствами файловой системы) и обеспечивает, таким образом, исключительный доступ только тому пользователю, который работает с документом.

Многие системы управления версиями предоставляют ряд других возможностей:

* Позволяют создавать разные варианты одного документа, т. н. ветки, с общей историей изменений до точки ветвления и с разными — после неё.
* Дают возможность узнать, кто и когда добавил или изменил конкретный набор строк в файле.
* Ведут журнал изменений, в который пользователи могут записывать пояснения о том, что и почему они изменили в данной версии.

Контролируют права доступа пользователей, разрешая или запрещая чтение или изменение данных, в зависимости от того, кто запрашивает это действие.

Каждая система управления версиями имеет свои специфические особенности в наборе команд, порядке работы пользователей и администрировании. Тем не менее, общий порядок работы для большинства VCS совершенно стереотипен. Здесь предполагается, что проект, каким бы он ни был, уже существует и на сервере размещён его репозиторий, к которому разработчик получает доступ.

* Начало работы с проектом

Первым действием, которое должен выполнить разработчик, является извлечение рабочей копии проекта или той его части, с которой предстоит работать. Это действие выполняется с помощью стандартной команды извлечения версии (checkout или clone) либо специальной команды, фактически выполняющей то же самое действие. Разработчик задаёт версию, которая должна быть скопирована, по умолчанию обычно копируется последняя (или выбранная администратором в качестве основной) версия.

По команде извлечения устанавливается соединение с сервером и проект (или его часть — один из каталогов с подкаталогами) в виде дерева каталогов и файлов копируется на компьютер разработчика. Обычной практикой является дублирование рабочей копии: помимо основного каталога с проектом на локальный диск (либо в отдельный, специально выбранный каталог, либо в системные подкаталоги основного дерева проекта) дополнительно записывается ещё одна его копия. Работая с проектом, разработчик изменяет только файлы основной рабочей копии. Вторая локальная копия хранится в качестве эталона, позволяя в любой момент без обращения к серверу определить, какие изменения внесены в конкретный файл или проект в целом и от какой версии была «отпочкована» рабочая копия; как правило, любая попытка ручного изменения этой копии приводит к ошибкам в работе программного обеспечения VCS.

* Ежедневный цикл работы

При некоторых вариациях, определяемых особенностями системы и деталями принятого бизнес-процесса, обычный цикл работы разработчика в течение рабочего дня выглядит следующим образом.

* Обновление рабочей копии

По мере внесения изменений в проект рабочая копия на компьютере разработчика стареет, расхождение её с основной версией проекта увеличивается. Это повышает риск возникновения конфликтных изменений (см. ниже). Поэтому удобно поддерживать рабочую копию в состоянии, максимально близком к текущей основной версией, для чего разработчик выполняет операцию обновления рабочей копии (update) насколько возможно часто (реальная частота обновлений определяется частотой внесения изменений, зависящей от активности разработки и числа разработчиков, а также временем, затрачиваемым на каждое обновление — если оно велико, разработчик вынужден ограничивать частоту обновлений, чтобы не терять время).

* Модификация проекта

Разработчик модифицирует проект, изменяя входящие в него файлы в рабочей копии в соответствии с проектным заданием. Эта работа производится локально и не требует обращений к серверу VCS.

* Фиксация изменений

Завершив очередной этап работы над заданием, разработчик фиксирует (commit) свои изменения, передавая их на сервер (либо в основную ветвь, если работа над заданием полностью завершена, либо в отдельную ветвь разработки данного задания). VCS может требовать от разработчика перед фиксацией обязательно выполнить обновление рабочей копии. При наличии в системе поддержки отложенных изменений (shelving) изменения могут быть переданы на сервер без фиксации. Если утверждённая политика работы в VCS это позволяет, то фиксация изменений может проводиться не ежедневно, а только по завершении работы над заданием; в этом случае до завершения работы все связанные с заданием изменения сохраняются только в локальной рабочей копии разработчика.

* Ветвления

Делать мелкие исправления в проекте можно путём непосредственной правки рабочей копии и последующей фиксацией изменений прямо в главной ветви (стволе) на сервере. Однако при выполнении сколько-нибудь значительных по объёму работ такой порядок становится неудобным: отсутствие фиксации промежуточных изменений на сервере не позволяет работать над чем-либо в групповом режиме, кроме того, повышается риск потери изменений при локальных авариях и теряется возможность анализа и возврата к предыдущим вариантам кода в пределах данной работы. Поэтому для таких изменений обычной практикой является создание ветвей (branch), то есть «отпочковывания» от ствола в какой-то версии нового варианта проекта или его части, разработка в котором ведётся параллельно с изменениями в основной версии. Ветвь создаётся специальной командой. Рабочая копия ветви может быть создана заново обычным образом (командой извлечения рабочей копии, с указанием адреса или идентификатора ветви), либо путём переключения имеющейся рабочей копии на заданную ветвь.

Базовый рабочий цикл при использовании ветвей остаётся точно таким же, как и в общем случае: разработчик периодически обновляет рабочую копию (если с ветвью работает более одного человека) и фиксирует в ней свою ежедневную работу. Иногда ветвь разработки так и остаётся самостоятельной (когда изменения порождают новый вариант проекта, который далее развивается отдельно от основного), но чаще всего, когда работа, для которой создана ветвь, выполнена, ветвь реинтегрируется в ствол (основную ветвь). Это может делаться командой слияния (обычно merge), либо путём создания патча (patch), содержащего внесённые в ходе разработки ветви изменения и применения этого патча к текущей основной версии проекта.

* Слияние версий

Три вида операций, выполняемых в системе управления версиями, могут приводить к необходимости объединения изменений. Это:

1) Обновление рабочей копии (изменения, сделанные в основной версии, сливаются с локальными).

2) Фиксация изменений (локальные изменения сливаются с изменениями, уже зафиксированными в основной версии).

3) Слияние ветвей (изменения, сделанные в одной ветви разработки, сливаются с изменениями, сделанными в другой).

Во всех случаях ситуация принципиально одинакова и имеет следующие характерные черты:

1) Ранее была сделана копия дерева файлов и каталогов репозитория или его части.

2)Впоследствии и в оригинальное дерево и в копию были независимо внесены некоторые изменения.

3) Требуется объединить изменения в оригинале и копии таким образом, чтобы не нарушить логическую связность проекта и не потерять данные.

Совершенно очевидно, что при невыполнении условия (2) (то есть если изменения были внесены только в оригинал или только в копию) объединение элементарно — достаточно скопировать изменённую часть туда, где изменений не было. В противном случае слияние изменений превращается в нетривиальную задачу, во многих случаях требующую вмешательства разработчика. В целом механизм автоматического слияния изменений работает, основываясь на следующих принципах:

Изменения могут состоять в модификации содержимого файла, создании нового файла или каталога, удалении или переименовании ранее существовавшего файла или каталога в проекте.

Если два изменения относятся к разным и не связанным между собой файлам и/или каталогам, они всегда могут быть объединены автоматически. Их объединение состоит в том, что изменения, сделанные в каждой версии проекта, копируются в объединяемую версию.

Создание, удаление и переименование файлов в каталогах проекта могут быть объединены автоматически, если только они не конфликтуют между собой. В этом случае изменения, сделанные в каждой версии проекта, копируются в объединяемую версию. Конфликтующими обычно являются:

1) Удаление и изменение одного и того же файла или каталога.

2) Удаление и переименование одного и того же файла или каталога (в случае, если система поддерживает операцию переименования).

3) Создание в разных версиях файла с одним и тем же именем и разным содержимым.

Изменения в пределах одного текстового файла, сделанные в разных версиях, могут быть объединены, если они находятся в разных местах этого файла и не пересекаются. В этом случае в объединённую версию вносятся все сделанные изменения.

Изменения в пределах одного файла, если он не является текстовым, всегда являются конфликтующими и не могут быть объединены автоматически.

Во всех случаях базовой версией для слияния является версия, в которой было произведено разделение сливаемых версий. Если это операция фиксации изменений, то базовой версией будет версия последнего обновления перед фиксацией, если обновление — то версия предыдущего обновления, если слияние ветвей — то версия, в которой была создана соответствующая ветвь. Соответственно, сопоставляемыми наборами изменений будут наборы изменений, сделанных от базовой до текущей версии во всех объединяемых вариантах.

Абсолютное большинство современных систем управления версиями ориентировано, в первую очередь, на проекты разработки программного обеспечения, в которых основным видом содержимого файла является текст. Соответственно, механизмы автоматического слияния изменений ориентируются на обработку текстовых файлов, то есть файлов, содержащих текст, состоящий из строк буквенно-цифровых символов, пробелов и табуляций, разделённых символами перевода строки.

При определении допустимости слияния изменений в пределах одного и того же текстового файла работает типовой механизм построчного сравнения текстов (примером его реализации является системная утилита GNU diff), который сравнивает объединяемые версии с базовой и строит список изменений, то есть добавленных, удалённых и заменённых наборов строк. Минимальной единицей данных для этого алгоритма является строка, даже самое малое отличие делает строки различными. С учётом того, что символы-разделители, в большинстве случаев, не несут смысловой нагрузки, механизм слияния может игнорировать эти символы при сравнении строк.

Те найденные наборы изменённых строк, которые не пересекаются между собой, считаются совместимыми и их слияние делается автоматически. Если в сливаемых файлах находятся изменения, затрагивающие одну и ту же строку файла, это приводит к конфликту. Такие файлы могут быть объединены только вручную. Любые файлы, кроме текстовых, с точки зрения VCS являются бинарными и не допускают автоматического слияния.

* Конфликты и их разрешение

Ситуация, когда при слиянии нескольких версий сделанные в них изменения пересекаются между собой, называют конфликтом. При конфликте изменений система управления версиями не может автоматически создать объединённый проект и вынуждена обращаться к разработчику. Как уже говорилось выше, конфликты могут возникать на этапах фиксации изменений, обновления или слияния ветвей. Во всех случаях при обнаружении конфликта соответствующая операция прекращается до его разрешения. Некоторые системы (например, Subversion) даже не пытаются разрешать конфликты на этапе фиксации или слияния ветви со стволом; при возникновении таких конфликтов операция полностью отменяется и разработчику предлагается сначала выполнить обновление рабочей копии (очевидно, что при этом возникнут те же самые конфликты), разрешить конфликты и только после этого выполнять объединение своих изменений с базовой ветвью.

Для разрешения конфликта система, в общем случае, предлагает разработчику три варианта конфликтующих файлов: базовый, локальный и серверный. Конфликтующие изменения либо показываются разработчику в специальном программном модуле объединения изменений (в этом случае там демонстрируются сливаемые варианты и динамически изменяющийся в зависимости от команд пользователя объединённый вариант файла), либо просто помечаются специальной разметкой прямо в тексте объединённого файла (тогда разработчик должен сам сформировать желаемый текст в спорных местах и сохранить его).

Конфликты в файловой системе разрешаются проще: там может конфликтовать только удаление файла с одной из прочих операций, а порядок файлов в каталоге не имеет значения, так что разработчику остаётся лишь выбрать, какую операцию нужно сохранить в сливаемой версии.

* Блокировки

Механизм блокировки позволяет одному из разработчиков захватить в монопольное использование файл или группу файлов для внесения в них изменений. На то время, пока файл заблокирован, он остаётся доступным всем остальным разработчикам только на чтение, и любая попытка внести в него изменения отвергается сервером. Технически блокировка может быть организована по-разному. Типичным для современных систем является следующий механизм.

Файлы, для работы с которыми требуется блокировка, помечаются специальным флагом «блокируемый». Такая пометка может ставиться автоматически при добавлении файла в проект, обычно для этого предварительно создаётся список масок имён файлов, которые при добавлении должны становиться блокируемыми.

Если файл помечен как блокируемый, то при извлечении рабочей копии с сервера он получает в локальной файловой системе атрибут «только для чтения», что препятствует его случайному редактированию.

Разработчик, желающий изменить блокируемый файл, вызывает специальную команду блокировки (lock) с указанием имени этого файла. В результате работы этой команды происходит следующее:

1) сервер проверяет, не заблокирован ли уже файл другим разработчиком; если это так, то команда блокировки завершается с ошибкой «файл заблокирован другим пользователем» и разработчик, вызывавший её, должен ожидать, пока другой пользователь не снимет свою блокировку;

2) файл на сервере помечается как «заблокированный», с сохранением идентификатора заблокировавшего его разработчика и времени блокировки;

3) если блокировка на сервере прошла удачно, на локальной файловой системе с файла рабочей копии снимается атрибут «только для чтения», что позволяет начать его редактировать.

4) Разработчик работает с заблокированным файлом. Если в процессе работы выясняется, что файл изменять не нужно, он может вызвать команду снятия блокировки (unlock, release lock). Все изменения файла будут отменены, локальный файл вернётся в состояние «только для чтения», с файла на сервере будет снят атрибут «заблокирован» и другие разработчики получат возможность изменять этот файл.

По завершении работы с блокируемым файлом разработчик фиксирует изменения. Обычно блокировка при этом снимается автоматически, хотя в некоторых системах блокировку требуется снимать вручную после фиксации, либо указывать в команде фиксации изменений соответствующий параметр. Так или иначе, при этом файл после изменений теряет флаг «заблокирован» и может быть изменён другими разработчиками.

Массовое использование блокировок, когда все или большинство файлов в проекте являются блокируемыми и для любых изменений необходимо заблокировать соответствующий набор файлов, называется ещё стратегией «блокированного извлечения». Ранние системы управления версиями поддерживали исключительно эту стратегию, предотвращая таким способом появление конфликтов на корню. В современных VCS предпочтительным является использование неблокирующих извлечений, блокировки же считаются скорее неизбежным злом, которое нужно по возможности ограничивать. Недостатки использования блокировок очевидны:

1) Блокировки просто мешают продуктивной работе, поскольку вынуждают ожидать освобождения блокированных файлов, хотя в большинстве случаев даже совместные изменения одних и тех же файлов, которые делаются в ходе разных по смыслу работ, не пересекаются и объединяются при слиянии автоматически.

2) Частота возникновения конфликтов и сложность их разрешения в большинстве случаев не настолько велики, чтобы создать серьёзные затруднения. Возникновение же серьёзного конфликта изменений чаще всего сигнализирует либо о существенном расхождении во мнениях разных разработчиков относительно дизайна одного и того же фрагмента, либо о неправильной организации работы (когда два или более разработчика делают одно и то же).

3) Блокировки создают административные проблемы. Типичный пример: разработчик может забыть снять блокировку с занятых им файлов, уходя в отпуск. Для разрешения подобных проблем приходится применять административные меры, в том числе включать в систему технические средства для сброса неверных блокировок, но и при их наличии на приведение системы в порядок расходуется время.

С другой стороны, в некоторых случаях использование блокировок вполне оправданно. Очевидным примером является организация работы с бинарными файлами, для которых нет инструментальных средств слияния изменений либо такое слияние принципиально невозможно (как, например, для файлов изображений). Если автоматическое слияние невозможно, то при обычном порядке работы любое параллельное изменение подобных файлов будет приводить к конфликту. В данном случае гораздо удобнее сделать такой файл блокируемым, чтобы гарантировать, что любые изменения в него будут вноситься только последовательно.

* Версии проекта, тэги

Система управления версиями обеспечивает хранение всех существовавших вариантов файлов и, как следствие, всех вариантов проекта в целом, имевших место с момента начала его разработки. Но само понятие «версии» в разных системах может трактоваться двумя различными способами.

Одни системы поддерживают версионность файлов. Это означает, что любой файл, появляющийся в проекте, получает собственный номер версии (обычно — номер 1, условной «нулевой» версией файла считается пустой файл с тем же именем). При каждой фиксации разработчиком изменений, затрагивающих файл, соответствующая часть фиксируемых изменений применяется к файлу и файл получает новый, обычно следующий по порядку, номер версии. Поскольку фиксации обычно затрагивают только часть файлов в репозитории, номера версий файлов, имеющиеся на один и тот же момент времени, со временем расходятся, и проект в целом (то есть весь набор файлов репозитория), фактически, никакого «номера версии» не имеет, поскольку состоит из множества файлов с различными номерами версий. Подобным образом работает, например, система управления версиями CVS.

Для других систем понятие «версия» относится не к отдельному файлу, а к репозиторию целиком. Вновь созданный пустой репозиторий имеет версию 1 или 0, любая фиксация изменений приводит к увеличению этого номера (то есть даже при изменении одного файла на один байт весь репозиторий считается изменённым и получает новый номер версии). Таким способом трактует номера версий, например, система Subversion. Номера версии отдельного файла здесь, фактически, не существует, условно можно считать таковым текущий номер версии репозитория (то есть считать, что при каждом изменении, внесённом в репозиторий, все его файлы меняют номер версии, даже те, которые не менялись). Иногда, говоря о «версии файла» в таких системах, имеют в виду ту версию репозитория, в которой файл был последний раз (до интересующего нас момента) изменён.

Для практических целей обычно имеет значение не отдельный файл, а весь проект целиком. В системах, поддерживающих версионность отдельных файлов, для идентификации определённой версии проекта можно использовать дату и время — тогда версия проекта будет состоять из тех версий входящих в него файлов, которые имелись в репозитории на указанный момент времени. Если поддерживается версионность репозитория в целом, номером версии проекта может выступать номер версии репозитория. Однако оба варианта не слишком удобны, так как ни дата, ни номер версии репозитория обычно не несут информации о значимых изменениях в проекте, о том, насколько долго и интенсивно над ним работали. Для более удобной пометки версий проекта (или его частей) системы управления версиями поддерживают понятие тэгов.

Тэг (tag) — это символическая метка, которая может быть связана с определённой версией файла и/или каталога в репозитории. С помощью соответствующей команды всем или части файлов проекта, отвечающим определённым условиям (например, входящим в головную версию главной ветви проекта на определённый момент времени) может быть присвоена заданная метка. Таким образом можно идентифицировать версию проекта (версия «XX.XXX.XXX» — это набор версий файлов репозитория, имеющих тэг «XX.XXX.XXX»), зафиксировав таким образом его состояние на некоторый желаемый момент. Как правило, система тэгов достаточно гибкая и позволяет пометить одним тэгом и не одновременные версии файлов и каталогов. Это позволяет собрать «версию проекта» любым произвольным образом. С точки зрения пользователя системы пометка тэгами может выглядеть по-разному. В некоторых системах она изображается именно как пометка (тэг можно создать, применить к определённым версиям файлов и каталогов, снять). В других системах (например, Subversion) тэг представляет собой просто отдельный каталог на файловом дереве репозитория, куда из ствола и ветвей проекта с помощью команды копирования делаются копии нужных версий файлов. Так что визуально тэг — это просто вынесенная в отдельный каталог копия определённых версий файлов репозитория. По соглашению в дерево каталогов, соответствующее тэгу, запрещена фиксация изменений (то есть версия проекта, представляемая тэгом, является неизменной).

## 1.3. Анализ программных средств разработки

### Выбор инструментальных средств

### Выбор способа хранения данных

### Сервер Apache

### Netbeans IDE

### Средство отладки Opera Dragonfly

Developer Tools (до версии 10.50 Opera Dragonfly) — инструмент для веб-разработчиков. Работает на технологии JavaScript. Код Opera Dragonfly выпущен под свободной лицензией BSD.

Возможности:

* Просмотр DOM;
* Просмотр и редактирование CSS;
* Отладчик JavaScript;
* Просмотр HTTP запросов;
* Отлаживать страницы для мобильных устройств;

Удалённо подключаться к любым компьютерам и устройствам с установленным браузером Opera, поддерживающим данный инструмент, и осуществлять отладку веб-страниц;

Рис. 6. Вид Opera Dragonfly при инспектировании элемента.



Рис. 7. Opera Dragonfly. Область свойств CSS.

### Клиент GitHub - TortoiseGit

### Sql сервер MySQL

### Шифрование MD5

MD5 позволяет получать относительно надёжный идентификатор для блока данных. Такое свойство алгоритма широко применяется в разных областях. Оно позволяет искать дублирующиеся файлы на компьютере, сравнивая MD5 файлов, а не их содержимое. Как пример, dupliFinder — графическая программа под Windows и Linux. Такой же поиск может работать и в интернете.

С помощью MD5 проверяют целостность скачанных файлов — так, некоторые программы идут вместе со значением хеша. Например, диски для инсталляции.

MD5 используется для хеширования паролей. В системе UNIX каждый пользователь имеет свой пароль и его знает только пользователь. Для защиты паролей используется хеширование. Предполагалось, что получить настоящий пароль можно только полным перебором. При появлении UNIX единственным способом хеширования был DES (Data Encryption Standard), но им могли пользоваться только жители США, потому что исходные коды DES нельзя было вывозить из страны. Во FreeBSD решили эту проблему. Пользователи США могли использовать библиотеку DES, а остальные пользователи имеют метод, разрешённый для экспорта. Поэтому в FreeBSD стали использовать MD5 по умолчанию.[11]. Некоторые Linux-системы также используют MD5 для хранения паролей.

Многие системы используют базу данных для хранения паролей и существует несколько способов для хранения паролей.

* Пароли хранятся как есть. При взломе такой базы все пароли станут известны.
* Хранятся только хеши паролей (с помощью MD5, SHA). Найти пароли можно только полным перебором. Но при условии использования несложного, популярного или просто несчастливого пароля (который встречался ранее и занесён в таблицу) такая задача решается за доли секунды. Пароль из таблицы был найден всего за 0,036059 сек.
* Хранятся хеши паролей и несколько случайных символов. К каждому паролю добавляется несколько случайных символов (их ещё называют «salt» или «соль») и результат ещё раз хешируется. Например, md5(md5(pass)+word). Найти пароль с помощью таблиц таким методом не получится.

## 1.4. Архитектура приложения

# Конструкторская часть

## Проектирование программных модулей

### Модуль регистрации

Клиентская часть – это то, что видит пользователь. Клиентская часть регистрационной части – это веб-форма, которая содержит в себе поля ввода, и кнопку «Регистрация»

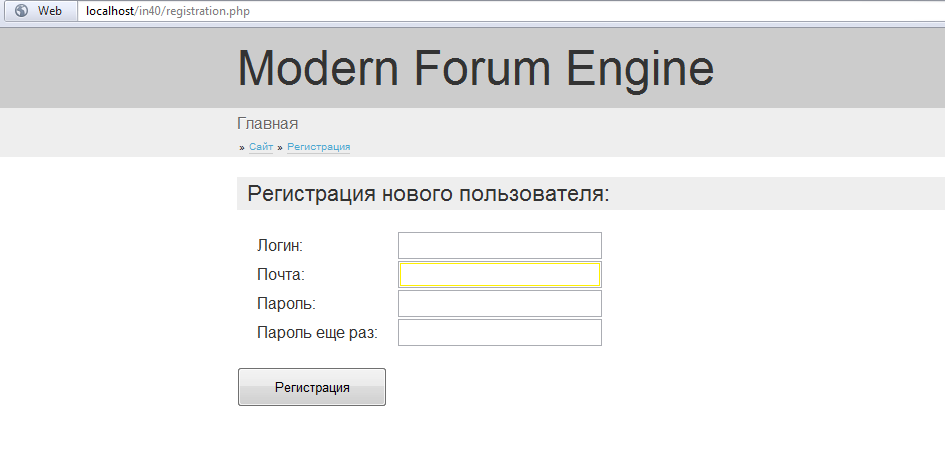


Рис. 8. Страница регистрации нового пользователя.

Все поля должны быть корректны. Алгоритм проверки корректности полей изображен на рисунке ниже.



Рис. 9. Алгоритм проверки корректности полей.

Логин должен быть уникальным, содержать латинские буквы, а так же кириллицу. Данная проверка осуществляется функциями Javascript. Если логин прошел эту проверку, он отправляется ajax запросом на сервер. Сервер извещает клиент о статусе логина – уникален он или нет. После этого происходит проверка почты функциями Javascript. Так же сравниваются пароли на совпадение. Если все проверки пройдены успешно, пользователь может их отправить на сервер, нажав кнопку «Регистрация».

В случае некорректности каких-либо данных выводятся сообщения, которые оповещают пользователя об этом. Примеры этих сообщений изображены на рисунке ниже.

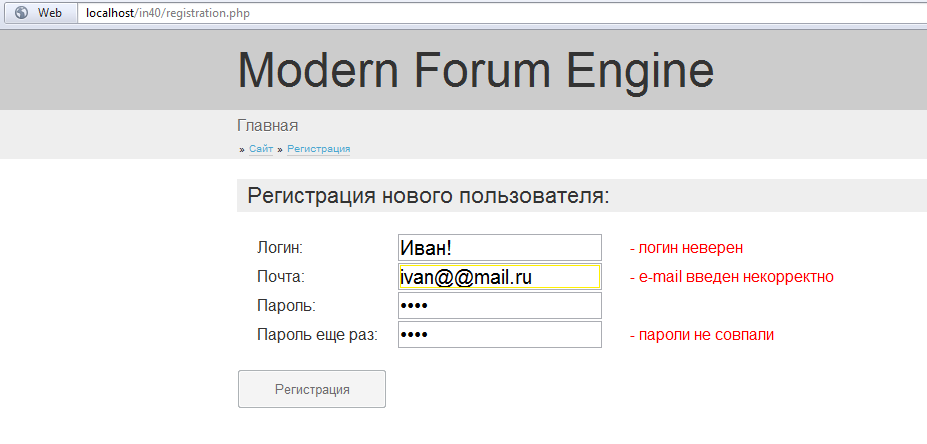


Рис. 10. Пример некорректного заполнения формы

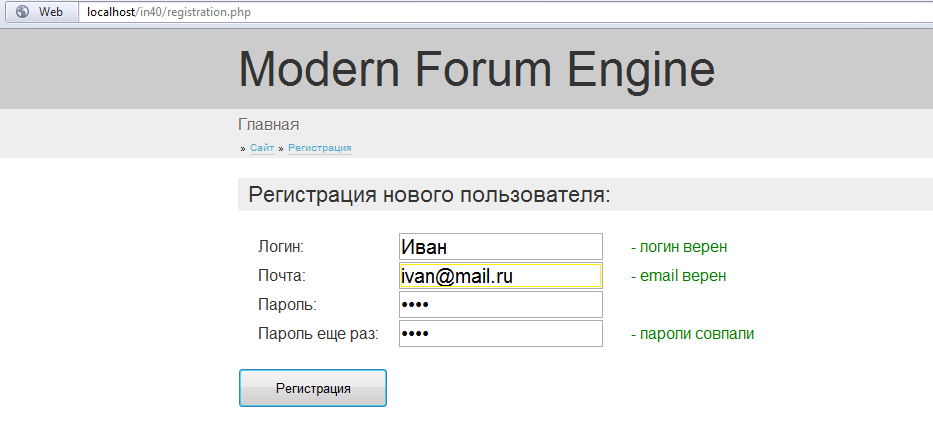


Рис. 11. Пример корректного заполнения формы

Перед отправкой всех данных, клиентская часть шифрует пароль, чтобы он был передан на сервер в зашифрованном виде. Это делается при помощи функций Javascript.

При создании регистрационной части были созданы следующие файлы:

* Registration.php – компоновщик страницы регистрации;
* Registration.js – скрипт, обеспечивающий интерактивность формы, проверку корректности и обмен данными между серверами(ajax).
* Registration.tpl – шаблон формы регистрации

Данные со стороны клиента поступают на сервер и записываются в базу данных. Но прежде чем это будет сделано, данные проходят еще один этап проверки. Это делается для большей безопасности, во избежание атак, цель которых – занести код в базу данных, чем самым получить доступ к ней.



Рис. 12. Серверная проверка корректности данных.

После проверки данные заносятся в базу данных.



Рис. 13. Поля таблицы пользователей в базе данных

Поля имеют следующие предназначения:

* Id – уникальный идентификатор пользователя в системе
* Login – логин пользователя
* User\_group\_id – идентификатор группы пользователей, к которой принадлежит данный пользователь
* Name – имя
* Father\_name – отчество
* Family\_name – фамилия
* Sex – пол
* Country – страна
* City – город
* Avatar – ссылка на сервис gravatar, Где хранится фотография пользователя
* Mobile – мобильный телефон
* Icq – сервис icq
* Vk – сервис вконтакте
* Facebook – сервис facebook
* About – информация о пользователе
* Registration\_date – дата регистрации
* Email – почта
* Password – хеш пароля

Если пользователю необходимо заполнить все поля, он может заполнить их после регистрации.

### Модуль пользователя

Дополнить или изменить личные данные можно в профиле пользователя.

Заполненные поля отправляются на сервер, где проходят проверку на корректность, во избежание внедрения SQL-кода. После этого данные заносятся в базу данных.

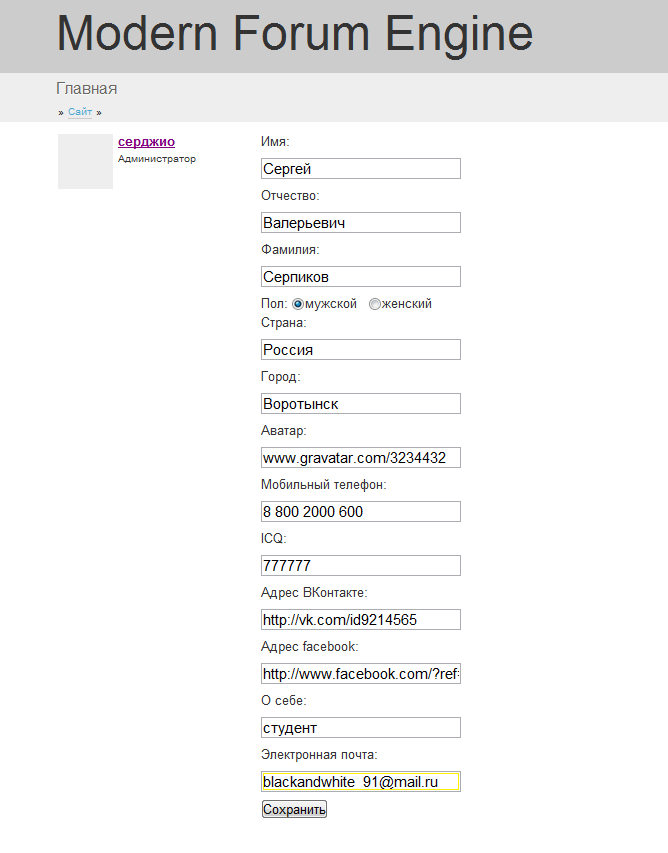


Рис. 14. Редактирование данных пользователя

### 

### Модуль навигации

Модуль навигации помогает пользователю перемещаться по разделам сайта.



Рис. 15. Пример навигационного меню

Пункты навигационного меню отображают путь до текущей страницы внутри сайта.

Был созданы файлы:

* bread\_crumbs.php – главный скрипт, собирающий модуль
* nav-item.tpl – шаблон пункта модуля навигации

Чтобы на странице был модуль навигации, необходимо в компоновщике страницы подключить данный модуль и вызвать инициализацию данного модуля:

include\_once 'modules/bread\_crumbs.php';

...

$page\_areas['nav'] = construct\_bread\_crumbs();

Главный скрипт bread\_crumbs.php определяет текущее положение страницы, просматриваемой пользователем, в иерархии сайта.

### Модуль логина

Модуль логина выводит поля ввода логина и пароля, а так же ссылку на форму регистрации, чтобы незарегистрированный пользователь мог зарегистрироваться. Если пользователь авторизовался на сайте, то модуль логина выводит текущий логин пользователя и ссылку для сброса авторизации.



Рис. 16. Модуль логина до авторизации



Рис. 17. Модуль логина после авторизации

Модуль логина состоит из файлов

* log\_reg.php – компоновщик
* login-form.tpl – шаблон для ввода пароля и логина
* login-hello.tpl – шаблон для выхода и индикации текущего пользователя

Для того, чтобы включить данный модуль необходимо в компоновщике страницы подключить данный модуль и вызвать инициализацию данного модуля:

include\_once 'modules/log\_reg.php';

...

$page\_areas['login'] = construct\_log\_reg();

## Создание тем

Пользователь может создать тему в категории. Если ему доступно данное право, то на странице будет отображена ссылка «создать тему».



Рис. . Ссылка на страницу создания темы.

Перед построением страницы, компоновщик ищет право у данного пользователя на создание тем. Если оно есть в базе данных, то ссылка будет добавлена в конечное построение страницы.

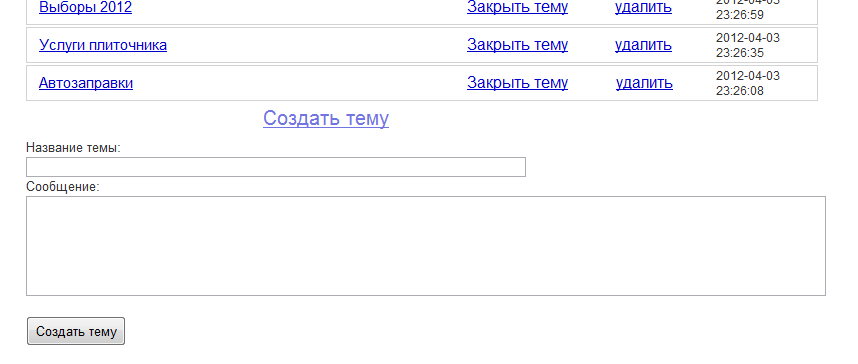


Рис. . Создание темы.

Чтобы создать тему, необходимо вписать название темы и текст сообщения. При нажатии на кнопку «отправить». Если есть незаполненное поле – кнопка будет неактивной, и, как следствие, создание темы будет невозможным.

## Создание ответа на тему

Пользователь имеет возможность ответа на тему. Для этого необходимо зайти в тему, нажав на ссылку, и внизу будет форма ответа на тему.

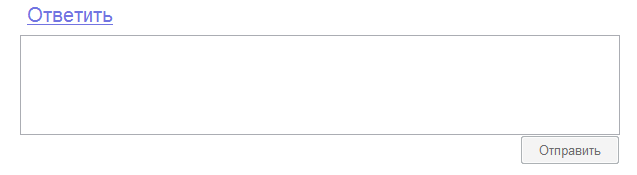


Рис. . Форма ответа на тему.

Если поле ввода сообщения не пустое, то кнопка «отправить» становиться активной, и пользователь имеет возможность отправить сообщение. Его сообщение публикуется в соответствующей таблице базы данных, а затем страница обновляется.

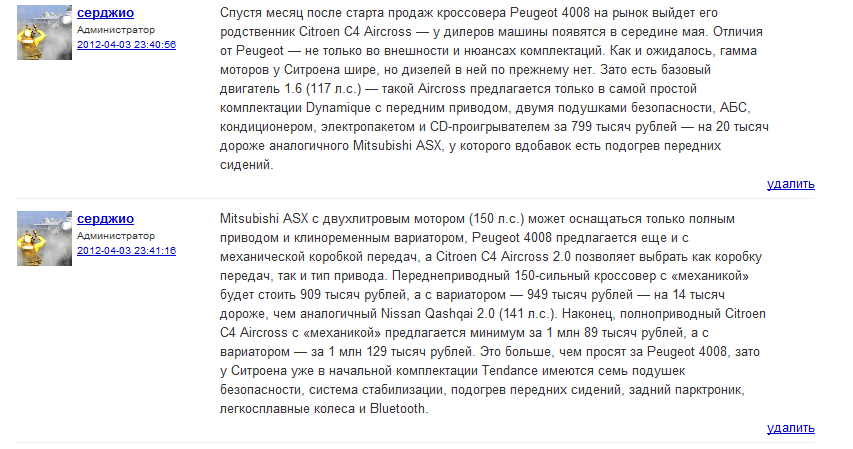


Рис. . Отправленные сообщения.

## Механизм разделения прав

В разрабатываемой системе необходимо реализовать гибкую систему прав. Чтобы грамотно это сделать, необходимо рассматривать всю систему как хранилище контента, где к каждой единице доступ регулируется системой прав. Кроме контента, система содержит действия пользователей, которые тоже должны регулироваться системой прав.

Рис. . Место системы прав в разрабатываемой системе

Чтобы упростить использование системы прав, все пользователи были разбиты по группам. Количество групп – произвольное число.



Рис. . Группирование пользователей

База данных содержит таблицу «general\_permissions» со списком прав с описаниями.



Рис. . Список прав с описаниями

Так же существует таблица групп пользователей, где каждая группа имеет свой уникальный идентификатор.



Рис. . Таблица групп пользователей.

Связующим звеном между правами и группами является таблица «user\_groups\_permissions»



Рис. . Соответствия прав группам пользователей

Именно в этой таблице и определяется набор прав у групп пользователей. При построении страницы компоновщиком, определяются права, и на основе их наличия подбирается соответствующий контент и выдаются управляющие ссылки. Если неуполномоченный пользователь вдруг получит управляющую ссылку (ссылку с GET-запросом, например, на удаление темы) при отсутствии прав на выполнении данной команды – команда будет проигнорирована компоновщиком.

Преимущество такого подхода в том, что можно определить право один раз и назначить его конкретной группе, каждый раз, при необходимости, проверяя его наличие.

## Проектирование базы данных. Логическое проектирование

CMS располагает контентом, который необходимо где-то хранить. Так же в ней содержится информация о пользователях, настройках и т.д. Поэтому для CMS необходима собственная база данных.

На этапе логического проектирования определяются сущности и их взаимосвязи.

Весь контент можно объединить по следующим категориям:

* пользователи
* текстовая информация форума
* права пользователей
* настройки сайта

Рассмотрим группу «пользователи». Каждый участник имеет личную информацию о себе и наделен некоторыми правами. Так как система будет содержать много групп пользователей с одинаковыми правами, так же понадобится сущность «группа пользователей». Пользователь состоит в группе, а группа наделена правами.

Текстовая информация представляет собой структурированные по категориям топики, которые содержат в себе текстовые сообщения. Таким образом, образуются сущности «категория», «тема», «сообщения». Сообщение связано с темой, которой она принадлежит, тема связана с категорией. При этом, необходимо обеспечить такой тип связи, при котором удаление родительской сущности вызывает удаление дочерних сущностей. Так же, к категории может применятся определенное действие пользователя, следовательно, нам необходимо иметь сущность «право категории».

Право в системе – это определенное действие (удаление контента, размещение сообщения). Таким образом, появляется сущность «право». Право может быть групповым, либо право категории. При удалении права, оно удаляется в связанных сущностях. Связанные сущности – «право группы пользователей» и «право категории». Если удаляется группа пользователей – с ней удаляются все связанные с ней права.