

**Министерство науки и высшего образования Российской Федерации**  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
**АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**  
**(ФГБОУ ВО «АмГУ»)**

Факультет математики и информатики  
Кафедра математического анализа и моделирования  
Направление подготовки – 01.03.02 Прикладная математика и информатика

ДОПУСТИТЬ К ЗАЩИТЕ  
И.о. зав. кафедрой  
\_\_\_\_\_ Н.Н. Максимова  
« \_\_\_\_\_ » \_\_\_\_\_ 2022 г.

**БАКАЛАВРСКАЯ РАБОТА**

на тему: Разработка менеджера паролей с использованием методов криптографического шифрования

Исполнитель  
студент группы 852об

\_\_\_\_\_  
(подпись, дата)

Д.М. Ховбощенко

Руководитель  
доцент, канд. физ.-мат. наук

\_\_\_\_\_  
(подпись, дата)

В.В. Сельвинский

Нормоконтроль  
старший преподаватель,  
канд. физ.-мат. наук

\_\_\_\_\_  
(подпись, дата)

Л.И. Мороз

Благовещенск 2022

Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
**АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**  
(ФГБОУ ВО «АмГУ»)

Факультет математики и информатики  
Кафедра математического анализа и моделирования

УТВЕРЖДАЮ  
И.о. зав. кафедрой  
\_\_\_\_\_ Н.Н. Максимова  
« \_\_\_\_\_ » \_\_\_\_\_ 2022 г.

**З А Д А Н И Е**

К бакалаврской работе студента Ховбощенко Даниила Михайловича

1. Тема бакалаврской работы: Разработка менеджера паролей с использованием методов криптографического шифрования (утверждена приказом от 05.04.2022 № 679-уч)
  2. Срок сдачи студентом законченной работы: 23.06.2022 г.
  3. Исходные данные к бакалаврской работе: отчет по преддипломной практике, научные статьи, учебно-методические работы, среда разработки – Ruby.
  4. Содержание бакалаврской работы (перечень подлежащих разработке вопросов): изучение методов шифрования и языков программирования; разработка клиентской части; реализация менеджера паролей с помощью Ruby.
  5. Перечень материалов приложения: листинг разработанной программы
  6. Консультанты по бакалаврской работе: нормоконтроль – Мороз Л.И., старший преподаватель, канд. физ.-мат. наук.
  7. Дата выдачи задания: 06.05.2022 г.
- Руководитель бакалаврской работы: Сельвинский Владимир Владимирович, доцент, канд. физ.-мат. наук
- Задание принял к исполнению (06.05.2022): \_\_\_\_\_ Ховбощенко Д.М.

## РЕФЕРАТ

Бакалаврская работа содержит 46 с., 13 рисунков, 3 таблицы, 3 приложения, 12 источников.

МЕНЕДЖЕР ПАРОЛЕЙ, КРИПТОГРАФИЧЕСКОЕ ШИФРОВАНИЕ, RUBY, ЗАЩИТА ДАННЫХ, БАЗА ДАННЫХ, КЛИЕНТ, ГЕНЕРАТОР ПАРОЛЕЙ

Дипломная работа посвящена разработке менеджера паролей с использованием методов криптографического шифрования.

В данной работе рассмотрены методы шифрования, методы программирования, разработана программа с интерфейсом.

Изучены методы криптографического шифрования, рассмотрены среды разработки программного обеспечения.

Написан и разработан менеджер паролей. Применён алгоритм AES-128 для шифрования. Использована система хранения баз данных PostgreSQL. Для разработки был выбран язык Ruby. В качестве хэш-функции выступал Bcrypt.

Представлено сравнение аналогов данного программного продукта, выведение плюсов и минусов каждого из них, проведён анализ.

## СОДЕРЖАНИЕ

Введение	6
1 Основные идеи разработки менеджера паролей	8
1.1 Выбор критериев	8
1.2 Рекомендации при создании мастер-пароля	11
1.3 Выбор инструментов	11
1.3.1 PostgreSQL	11
1.3.2 Ruby on rails	13
1.3.3 Общение между веб-сервером и приложением	17
1.3.4 Безопасность передаваемых данных	19
1.3.4.1 HTTPS	19
1.3.4.2 Bcrypt	21
1.3.4.3 AES 128	21
2 Архитектура	25
2.1 Схема архитектуры	25
2.2 Описание Архитектуры	26
2.2.1 Регистрация	26
2.2.2 Доступ к учетной записи	26
2.2.3 Защита данных	27
3 Описание программной реализации	28
3.1 Uml диаграмма	28
3.2 Описание классов	29
3.3 Зависимость времени шифрования от длины передаваемых данных.	
Время, затраченное на вычисление хэш-функции	30
3.4 База данных	33
3.5 Интерфейс	35
3.6 Используемые средства для разработки менеджера паролей	37
3.7 Установка необходимых компонентов	37
Заключение	39

Библиографический список	40
Приложение А Скрипт на языке ruby, вычисляет затраченное на шифрование время с учетом длины данных и значения параметра cost хеш-функции bcrypt.	41
Приложение Б Скрипт запуска nginx и nginx.conf.	42
Приложение В Конфигурация Unicorn.	45

## ВВЕДЕНИЕ

В современном обществе как никогда актуален вопрос хранения персональных данных, таких как логины, пароли, цифровые идентификаторы, необходимые для работы с огромным количеством ресурсов и приложений: форумы, сервисы электронной почты, социальные сети, электронные кошельки, компьютерные игры. Мотивацией для разработки данного менеджера паролей послужило желание создать сервис, позволяющий безопасно и удобно хранить свои идентификационные данные необходимые для доступа к разному роду ресурсов. Наиболее надежным способом защиты своих данных является создание сложного пароля для каждой учетной записи. Конечно, стараться запомнить большое количество паролей очень тяжело, а то и вовсе невозможно. Поэтому приходится прибегать к помощи менеджера паролей. Хранение паролей в текстовом файле или на бумажном листе – ненадежно, человек получивший доступ к ним без труда сможет воспользоваться открытыми данными. Можно воспользоваться одним, пусть даже очень сложным паролем для всех ресурсов, но это лишь увеличит риск присвоения ваших данных, потому что, узнав ваш пароль, злоумышленник получает доступ сразу ко всем ресурсам.

Цель: Разработка менеджера паролей с использованием методов криптографического шифрования.

Задачи:

1. Разработать концептуальную схему работы приложения.
2. Выбрать подходящий метод шифрования.
3. Разработать структуру приложения.
4. Разработать интерфейс.
5. Разработать конечный менеджер паролей.

В первой главе дипломной работы были рассмотрены критерии с помощью существующих аналогов, рекомендации к созданию мастер-пароля, а также выбраны средства разработки.

Во второй главе бакалаврской работы была описана архитектура разрабатываемого приложения.

В третьей главе работы был рассмотрен процесс разработки менеджера паролей, проведены некоторые тесты по проверке отклика сервера, описание средств и предоставление конечного программного продукта.

# 1 ОСНОВНЫЕ ИДЕИ РАЗРАБОТКИ МЕНЕДЖЕРА ПАРОЛЕЙ

## 1.1 Выбор критериев

При создании менеджера паролей учитывались следующие критерии:

1. Безопасность передаваемой информации.
2. Безопасность хранимых данных.
3. Интуитивный интерфейс.

Для выявления критериев, который современный пользователь хотел бы видеть в подобном сервисе, прошло ознакомление со схожими по назначению программными средствами. В частности, такие как онлайн менеджер паролей FortNotes, основная единица информации в котором *note*, это показано на рисунке 1.

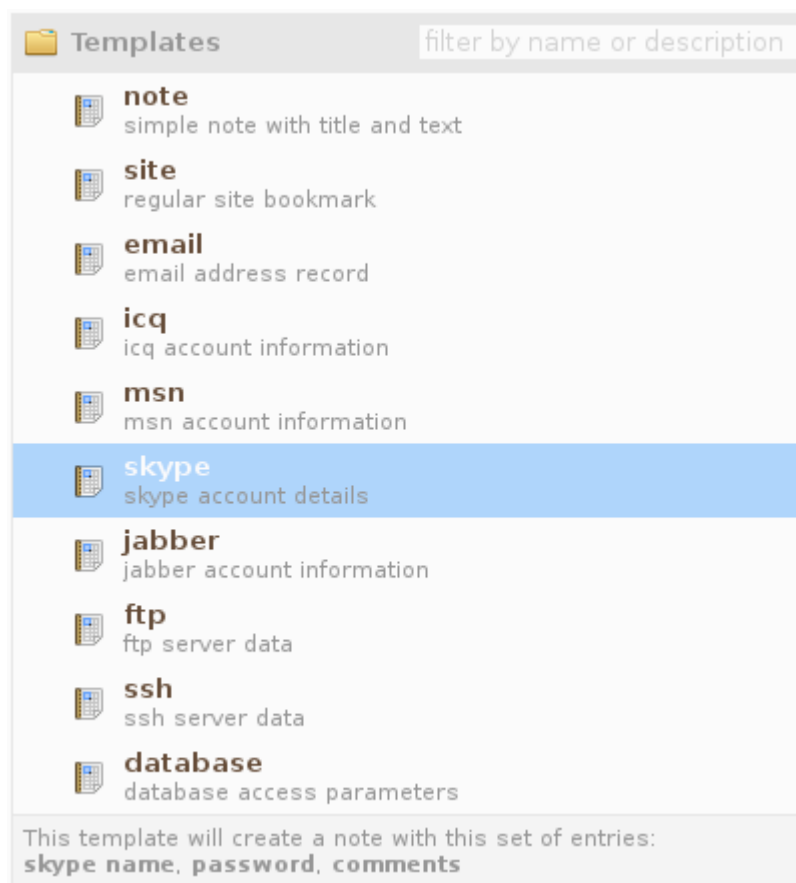


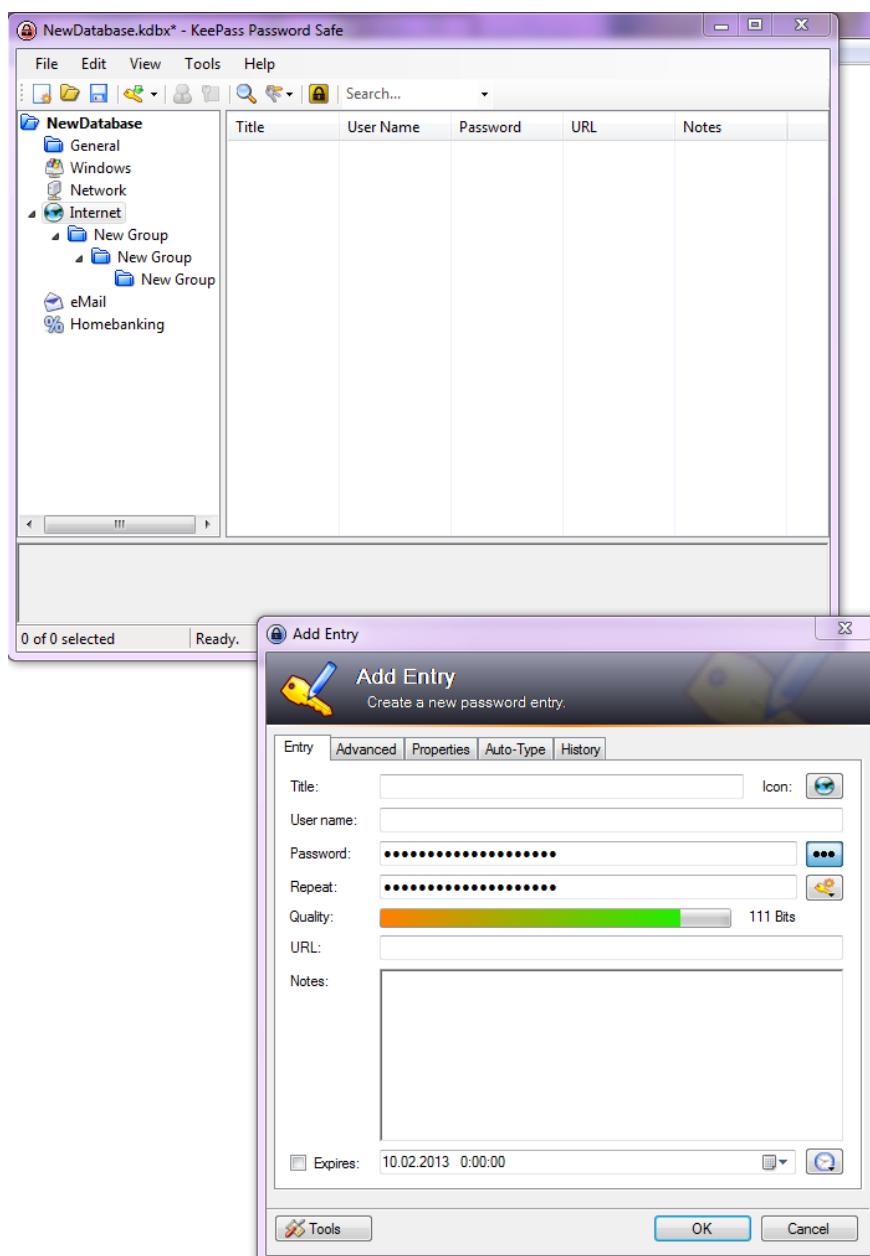
Рисунок 1 – Выбор заметки в менеджере паролей FortNotes

Из достоинств FortNotes можно выделить удобный доступ к учетной записи и гибкие возможности редактирования данных, но при этом высокий



уровень безопасности накладывает определенные неудобства. Так, нет возможности восстановить мастер-пароль, по которому происходит доступ к учетной записи и шифрование. К учетной записи нет привязки электронного адреса, что не дает возможности обратиться к владельцу напрямую. В обзорах возникали недовольные отзывы такой мерой предосторожности. По этому одной из важных функций реализуемого менеджера паролей стало внедрение функции восстановления мастер-пароля и привязка email адреса.

Менеджер паролей KeePass представлен в виде десктопного приложения с обширными возможностями редактирования данных, с возможностью создания множества каталогов, показано на рисунке 2



## Рисунок 2 – Интерфейс программы KeePass

Для шифрования данных используется единственный мастер-пароль. При этом интерфейс программы не слишком приятный, некоторых пользователей оттолкнет большое количество функций, программа работает только со своей локальной базой паролей, что исключает возможность без подготовки просмотреть свои данные на другом персональном компьютере.

Как эталон следует рассмотреть мультиверсионную программу 1Password. Как и большинство подобных продуктов, программа использует один пароль для доступа к данным.

Кроме паролей 1Password способна сохранять данные о лицензиях на программы, секретные заметки, данные кредитных карт и паспортов. Это осуществляется следующим образом.

Для каждой страницы сохраняется адрес (что исключает возможность использования данных фишинговыми сайтами), логин и пароль, а также скриншот. Кроме того, можно присвоить теги и заметки и прикрепить файл. Доступ и последующее использование сохраненных в 1Password паролей может осуществляться двумя способами. Первый, более подходящий для просмотра различных пунктов базы данных и их редактирования. Второй же способ доступа к записям базы данных паролей возможен при использовании расширения для браузера. Достойный генератор безопасных паролей. Кроме того параметры для генерации задает сам пользователь.

Рассмотрев достоинства и недостатки этих продуктов, было решено, создать менеджер паролей в виде клиент-серверного приложения, с централизованной базой данных, интуитивно понятным интерфейсом, не нагроможденным лишним функционалом, с методами защиты данных, не уступающих аналогам (большинство популярных программ использует AES). Для работы в менеджере паролей будет использована система из двухпарольного доступа: один пароль используется для доступа к учетной записи, второй для шифрования данных.

## 1.2 Рекомендации при создании мастер-пароля

Так как данный менеджер паролей использует выбранный мастер-пароль, чтобы сформировать ключ, он должен быть достаточно сложным, чтобы устоять при атаках злоумышленников, например полного перебора.

Существует несколько основных правил составления надежного пароля:

1. Пароль не должен содержать меньше шести символов.
2. Должен содержать заглавные и прописные буквы.
3. Должен содержать символы или элементы пунктуации.
4. Ни в коем случае не совпадать с именем пользователя.
5. Не должен содержать дату дня рождения пользователя или важных ему дат.
6. Пароль должен быть легко запоминаем, иначе все усилия окажутся напрасны.

Мастер-пароль так же может быть перехвачен злоумышленником при использовании кейлогера – программным или аппаратным средством, регистрирующим каждое нажатие клавиши на клавиатуре компьютера. Такая угроза может быть снижена путем использования виртуальной клавиатуры.

## 1.3 Выбор инструментов

### 1.3.1 PostgreSQL

Для хранения данных была выбрана система управления базами данных PostgreSQL. PostgreSQL поддерживается на всех современных Unix системах, включая самые распространенные, такие как Linux, FreeBSD, NetBSD, OpenBSD, а так же MacOS и MS Windows NT.

Из основных возможностей можно выделить:

1. Надежность.
  - обеспечивается полным соответствием принципов ACID – атомарность, непротиворечивость, изолированность, сохранность данных;
  - многоверсионность (Multiversion Concurrency Control, MVCC); используется для поддержания согласованности данных в конкурентных условиях.

- наличие Write Ahead Logging (WAL). Общепринятый механизм протоколирования всех транзакций, что позволяет восстановить систему после возможных сбоев;

- point in time Recovery (PITR) – восстановление базы (из WAL);

- целостность данных на уровне схемы – внешние ключи (foreign keys), ограничения (constraints);

- открытость кода, что означает абсолютную доступность для любого.

## 2. Производительность.

- поддержка индексов. Стандартных индексов, частичных индексов и функциональных индексов;

- планировщик запросов;

- система блокировки. Поддерживается на уровне таблиц и записей. На нижнем уровне, блокировка для общих ресурсов оптимизирована под конкретную ОС и архитектуру;

- масштабируемость.

## 3. Расширяемость.

## 4. Безопасность данных.

- PostgreSQL нельзя запустить под привилегированным пользователем;

- SSL, SSH шифрование трафика между клиентом и сервером;

- сложная система аутентификации на уровне хоста. Система аутентификации поддерживает пароли, Kerberos, IDENT и прочие системы, которые могут подключаться, используя механизм подключаемых аутентификационных модулей;

- детализированная система прав доступа ко всем объектам базы данных.

PostgreSQL предоставляет командный интерфейс для работы с системным каталогом, с помощью которого можно не только получать информацию об объектах системы, но и создавать новые. Например, создавать базы данных с помощью CREATE DATABASE, новый домен – CREATE DOMAIN, оператор – CREATE OPERATOR, тип данных – CREATE TYPE.

Для создания нового типа данных и индексных методов доступа достаточно:

- написать функции ввода/вывода и зарегистрировать их в системном каталоге с помощью CREATE FUNCTION;
- определить тип в системном каталоге с помощью CREATE TYPE;
- создать операторы для этого типа данных с помощью CREATE OPERATOR;
- написать функции сравнения и зарегистрировать их в системном каталоге с помощью CREATE FUNCTION;
- создать оператор по умолчанию, который будет использоваться для создания индекса по primary key - CREATE OPERATOR CLASS;
- описанный сценарий использует существующих вид индекса. Для создания новых индексов надо использовать GiST.

Одной из примечательных особенностей PostgreSQL является обобщенное поисковое дерево или GiST (домашняя страница проекта), которое дает возможность специалистам в конкретной области знаний создавать специализированные типы данных и обеспечивает индексный доступ к ним, не будучи экспертами в области баз данных. Аналогом GiST является технология DataBlade.

Дистрибутив PostgreSQL в поддиректории contrib/ содержит большое количество (около 80) так называемых контриб-модулей, реализующих разнообразную дополнительную функциональность, такую как полнотекстовый поиск, работа с xml, функции математической статистики, поиск с ошибками, криптографические модули и т.д. Также есть утилиты, облегчающие миграцию с mysql, oracle для административных работ.

### 1.3.2 Ruby on rails

Ruby-on-rails – свободно-используемый фреймворк, написанный на ruby, поддерживает использование архитектуры MVC (ModelViewController). По умолчанию идет с собственным ORM (Object-relation mapping).

Преимущества MVC:

1. Отделяет бизнес-логику от пользовательского интерфейса.
2. Легко хранить неповторяющийся код DRY.
3. Легко обслуживать приложение.

Схему модели можно посмотреть на рисунке 3.

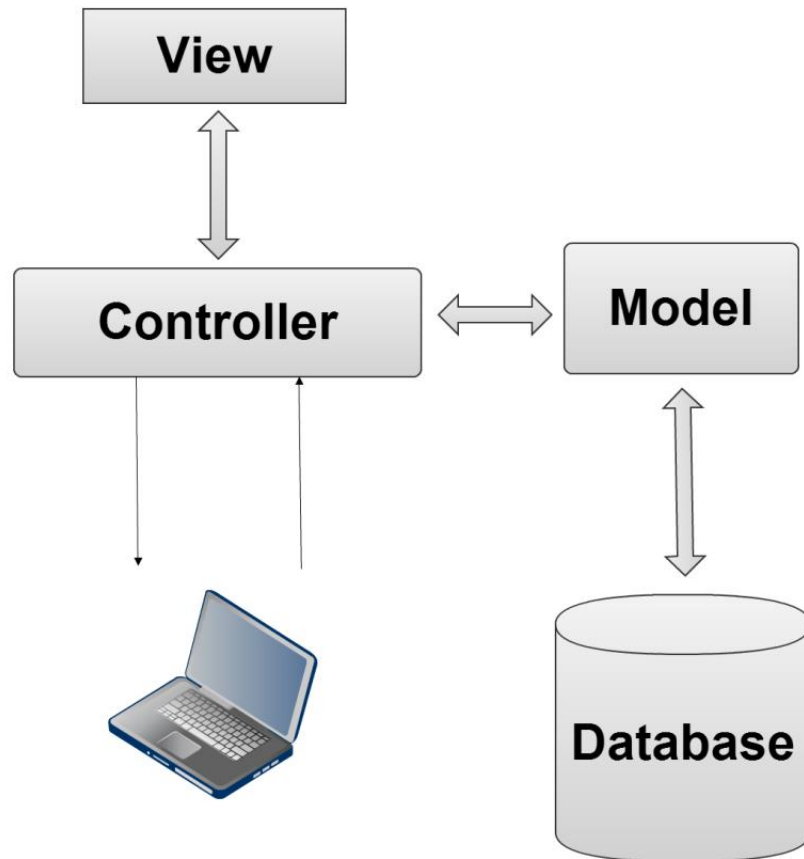


Рисунок 3 – Модель MVC

**Модель (Model).** Модель представляет собой информацию (данные) приложения и правила для обработки этих данных. Модель используется для управления правилами взаимодействия с таблицей базы данных [10].

**Представление (View).** Представления являются HTML файлами с встроенным кодом Ruby, который выполняет задачи, связанные исключительно с представлением данных. Их задача – предоставление данных веб-браузеру, который используется для обращения к серверу приложения.

**Контроллеры (Controllers).** Контроллер взаимодействует между моделью и представлением и клиентом, т.е. обрабатывает входящие запросы от веб-

браузера, запрашивает данные через модель и передачу этих данных, предварительно сформированных отображением.

Ruby on rails построен на следующих компонентах:

1. Action Pack. Отдельный gem (библиотека), содержащий Action Controller, Action View, Action Dispatch. Action Controller – компонент, который управляет контроллерами в приложении. Фреймворк обрабатывает входящие запросы к приложению, извлекает параметры и направляет их в назначенный «Action». Action Controller отвечает за такие сервисы как управление сессиями, управление перенаправлениями и рендеринг шаблонов. Action View – управление представлениями в приложении. На выходе по умолчанию создается html. Action Dispatch – управление маршрутизацией веб-запросов.

2. Active Mailer. Фреймворк для встроенных email служб. Обрабатывает входящую электронную почту, или рассылает электронные письма любой сложности, основанные на гибких шаблонах.

3. Active Model. Представляет из себя определенный интерфейс между службами гемма Action Pack и геммами Object Relationship Mapping, такими как Active Record. Active Model позволяет Rails использовать другие фреймворки ORM вместо Active Record, если вдруг приложению потребуются изменения.

4. Active Record. Основа для моделей в приложении. Он предоставляет независимость от базы данных, базовый CRUD-функционал, расширенные возможности поиск и способность устанавливать связи между моделями и модели с другим сервисом.

5. Railties. Код ядра ruby on rails, который создает новые приложения и соединяет разные фреймворки и плагины вместе.

В корне находятся папки, задающие структуру приложения. Большая часть работы выполняется в папке app/. Краткое описание целей папок и файлов представлены в таблице 1 [9].

Таблица 1 – Основные папки и файлы приложения

app/	Содержит контроллеры, модели и представления приложения.
config/	Конфигурации правил, маршрутов, базы данных приложения.
config.ru	Конфигурация Rack для серверов, основанных на Rack, используемых для запуска приложения.
db/	Содержит текущую схему базы данных, а также миграции базы данных.
doc/	Углубленная информация по приложению (документация)
Gemfile Gemfile.lock	Эти файлы позволяют определить, какие нужны зависимости от гемов для приложения на Rails.
lib/	Внешние модули для приложения.
log/	Файлы логов приложения.
public/	Содержит статичные файлы и скомпилированные ресурсы
Rakefile	Этот файл содержит набор команд, которые могут быть запущены в командной строке. Определения команд производятся во всех компонентах Rails.
script/	Содержит скрипт rails, который запускает приложение, и может содержать другие скрипты, используемые для развертывания или запуска приложения.
test/	Юнит-тесты, фикстуры и прочий аппарат тестирования.
tmp/	Временные файлы
vendor/	Место для кода внешних разработчиков. В типичном приложении на Rails, включает RubyGems.

Определение базы данных происходит в конфигурационном файле config/database.yml.

По умолчанию, файл содержит разделы для трех различных сред, в которых может быть запущен Rails:

- а) среда development используется на вашем рабочем/локальном компьютере для того, чтобы вы могли взаимодействовать с приложением;



б) среда `test` используется при запуске автоматических тестов;

в) среда `production` используется, для деплоя приложения.

Не нужно обновлять конфигурации баз данных вручную. Если взглянуть на опции генератора приложения, то можно увидеть, что одна из опций называется `database`. Эта опция позволяет выбрать адаптер из списка наиболее часто используемых СУРБД. Вы даже можете запускать генератор неоднократно: `cd ... && rails new blog – database = postgresql`. После того, как подтверждена перезапись `config/database.yml`, приложение станет использовать `postgresql`.

Конфигурирование базы данных PostgreSQL. При выборе PostgreSQL, `config/database.yml` будет модифицирован для использования базы данных PostgreSQL:

```
development:  
  adapter: postgresql  
  encoding: unicode  
  database: blog_development  
  pool: 5  
  username: blog  
  password:
```

### 1.3.3 Общение между веб-сервером и приложением

Ruby on Rails использует интерфейс RACK, что позволяет использовать менее распространённые механизмы (FCGI, CGI, SCGI). Ruby on Rails может работать с `nginx` или любым другим веб-сервером, поддерживающим FastCGI. FastCGI - открытый унифицированный стандарт, расширяющий интерфейс CGI, позволяющий создавать высокопроизводительные web-приложения без использования специфичных API web-сервера[12].

Интерфейс FastCGI является расширением (улучшением) CGI. FastCGI проектировался для поддержки «долго живущих» приложений - служб (сервисов). Это основное отличие от традиционной реализации CGI 1.1, где для

обработки каждого запроса создается процесс, однократно используется, после чего завершается. Стартовые условия у FastCGI-процесса более жесткие, чем у CGI, который при запуске имеет доступ к переменным окружения и может использовать стандартные потоки ввода-вывода - STDIN, STDOUT, STDERR.

Связь же между web-сервером и FastCGI-процессом осуществляется через один сокет, который процесс должен слушать на предмет входящих подключений от web-сервера. После приема соединения от web-сервера FastCGI-процесс обменивается данными с использованием простого протокола, решающего две задачи: организация двунаправленного обмена в рамках одного соединения (для эмуляции STDIN, STDOUT, STDERR) и организация нескольких независимых FastCGI-сессий в рамках одного соединения. FastCGI приложение выполняет одну из трех определенных протоколом ролей. Основная - Responder, в которой приложение получает данные HTTP-запроса и формирует HTTP-ответ (аналогично CGI 1.1). Вторая роль – Authorizer – приложение получает данные HTTP-запроса и генерирует решение авторизован или не авторизован. Третья - Filter, в которой приложение также получает данные HTTP-запроса и дополнительно поток данных, хранящихся на сервере, после чего выполняет фильтрацию (преобразование) данных и формирует HTTP-ответ. Может использоваться для изменения «на лету» статички - HTML-документов, изображений, а также для их кеширования.

Web-сервер или сервер приложений может передавать запускаемому приложению аргументы командной строки. По умолчанию передается единственный аргумент - имя. Запускаемым приложением может быть также интерпретируемый скрипт (начинающийся с #!), тогда интерпретатору передаются все аргументы, указанные в начале скрипта.

При запуске приложения web-сервер оставляет единственный файловый дескриптор - FCGI\_LISTENSOCK\_FILENO, который ссылается на сокет, созданный сервером. Приложение слушает этот дескриптор на предмет входящих соединений от сервера.

FCGI\_LISTENSOCK\_FILENO равен STDIN\_FILENO (имеет тот же номер в таблице дескрипторов - 0), остальные стандартные дескрипторы - STDOUT\_FILENO и STDERR\_FILENO - закрываются перед запуском приложения.

Web-сервер сам выбирает необходимый тип сокета: Unix-сокеты для локального взаимодействия (файловый сокет, AF\_UNIX), либо TCP/IP-сокеты для удаленного (AF\_INET). Оба сокеты имеют одинаковый интерфейс, поэтому детали транспорта скрыты от разработчика FastCGI-приложения.

Сервер может передавать дополнительные параметры приложению через переменные окружения (переменные среды). Текущая спецификация определяет только одну такую переменную - FCGI\_WEB\_SERVER\_ADDRS (содержит список доверенных хостов, от которых могут приходить FastCGI-запросы), однако сервер может предоставлять возможность задания любого набора переменных.

Также web-сервер может предоставлять возможности по установке любых других начальных параметров, таких как: корневая директория, пользователь и группа, рабочая директория процесса, приоритет, ограничения на память, число открытых дескрипторов.

#### 1.3.4 Безопасность передаваемых данных

##### 1.3.4.1 HTTPS

Для безопасной передачи данных используется протокол https, данные передаваемые по протоколу https защищены криптографическим протоколом SSL.

Браузер и сервер, устанавливая SSL соединение, используют процесс, называемый «криптография открытого ключа». Две части данных используются для создания SSL соединения: открытый ключ и закрытый ключ. Все, что шифруется с помощью открытого ключа пользователя, может быть расшифровано только при помощи закрытого ключа и наоборот.

Поскольку шифрование и дешифрование с помощью открытого и закрытого ключа использует большое количество вычислительной мощности,

оно используется только при первом создании SSL соединения для получения так называемого симметрического ключа. Симметрический ключ используют далее для шифрования данных веб-страницы. Упрощенную схему процесса создания SSL соединения можно увидеть ниже на рисунке 4



Рисунок 4 – Упрощенная схема процесса создания SSL соединения

До того, как SSL становится доступен требуется предоставить сертификат. Сертификат содержит открытый ключ, который идентифицирует сервер. Сертификат так же включает личные данные владельца. Сертификат содержит в себе так же электронную цифровую подпись. Сертификат создается

и проверяется Центрами сертификации (CA) – платные сертификаты. Можно так же использовать самоподписной сертификат, генерируемый непосредственно в браузере.

#### 1.3.4.2 Bcrypt

Bcrypt – адаптивная хэш-функция. По сути, bcrypt – это блочный шифр Blowfish, используемый в режиме ECB, с более сложным алгоритмом подготовки ключей (особенно, что касается S-блоков). Это позволяет алгоритму быть стойким к возможным будущим атакам и существенно более адаптивным. Реализация алгоритма использует 128-битную соль и усовершенствованный алгоритм, известный, как *eksblowfish* или *expensive key schedule blowfish*. Заголовок функции bcrypt выглядит так: `bcrypt (cost, salt, pwd)`. Здесь *cost* – контроллер подготовки ключей (задает ресурсоемкость фазы подготовки ключей), *salt* – 128-битное значение, а *pwd* – текстовый (до 56 байтов) ключ, используемый для шифрования по алгоритму Blowfish.

Алгоритму необходима соль (или же дополнительная последовательность символов) - она увеличит стойкость алгоритма. Алгоритм является адаптивным, т.е. время подготовки ключей может быть всего порядка одной миллисекунды или таким большим, как того потребует. Для сервера почти безразлично, что проверка пароля занимает 3 секунды вместо 0.3, но для атакующего такое увеличение сложности – абсолютный хаос. Это также позволяет напрямую бороться с экспоненциальным наращиванием вычислительных мощностей (известным, как закон Мура), поскольку со временем можно постепенно изменять значение переменной *cost*, чтобы увеличить сложность паролей.

Bcrypt – одна из наиболее популярных функций получения ключей (следующая по популярности за PBKDF2), и почти для каждого языка программирования доступна ее надежная реализация. В сравнении с PBKDF2, алгоритм расширения ключа в bcrypt практически не исследовался криптографами.

#### 1.3.4.3 AES 128

Структура алгоритма показана на рисунке 5.

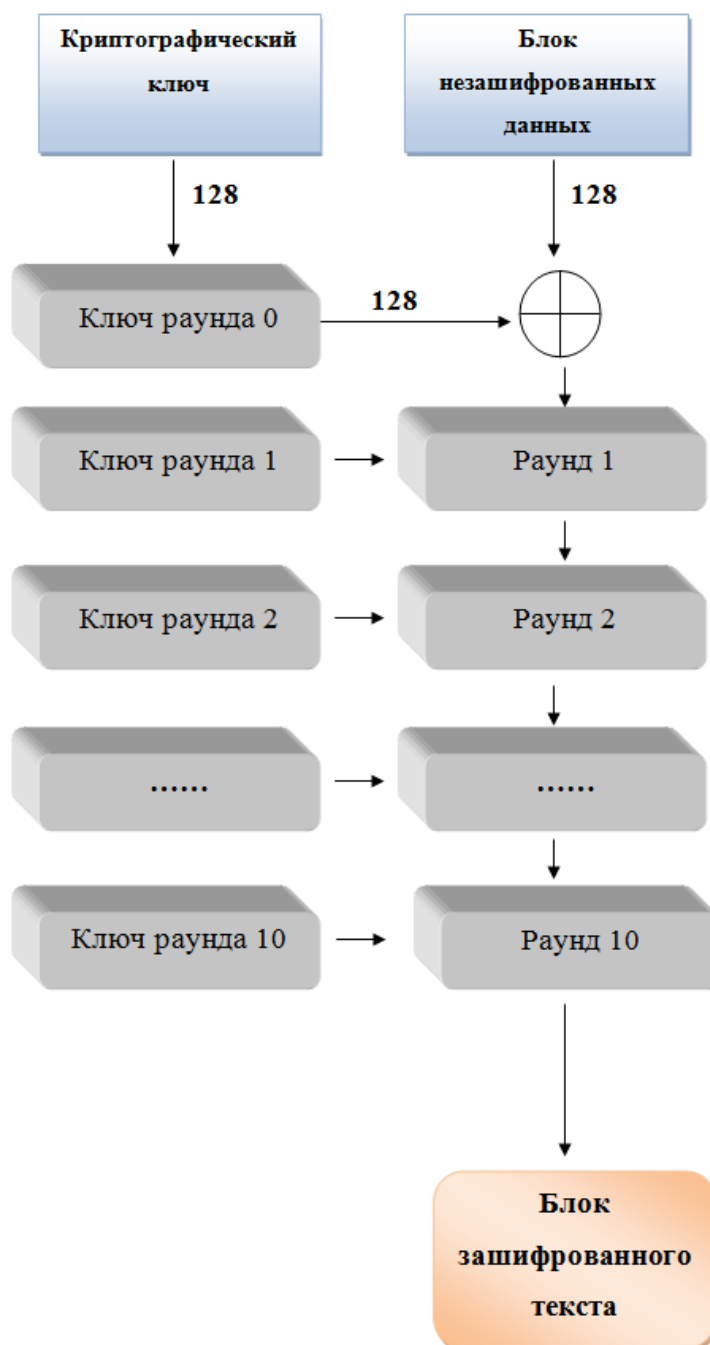


Рисунок 5 – Структура алгоритма AES

Улучшенный стандарт шифрования (Advanced Encryption Standard - AES) пришёл на смену стандарту Data Encryption Standard (DES).

Алгоритм AES состоит из часто повторяющихся раундов шифрования. Сначала на основе 128-битового ключа получают одиннадцать так называемых ключей раундов, каждый из которых имеет размер 128 бит. Каждый раунд включает в себя преобразование с использованием соответствующего криптографического ключа

После начального раунда, во время которого осуществляется логическая операция исключаящего или первого ключа раунда и незашифрованного текста (операция «Addroundkey»), следуют девять одинаково структурированных раундов.

Каждый раунд состоит из следующих операций:

- а) замещение байтов;
- б) сдвиг строк;
- в) перемешивание столбцов;
- г) добавление ключа раунда.

Десятый раунд подобен раундам с первого по девятый, но без операции перемешивания столбцов. И ключ, и исходные данные (называемые также «состоянием») структурированы в виде матрицы байтов размером 4x4.

Операция «Subbytes» - это нелинейное замещение. Это основная причина надёжности шифрования алгоритма AES. Существуют различные способы интерпретации операции замещения байтов. В этом отчёте достаточно представить этап замещения байтов как поиск по таблице. С помощью этой таблицы поиска 16 байт состояния (исходных данных) замещаются соответствующими значениями, найденными в таблице.

Операция «Shiftrows» - операция сдвига строк обрабатывает различные строки. Эта операция просто осуществляет циклическое смещение на различную величину смещения. Вторая строка исходных данных в виде байтовой матрицы 4x4 (состояния) смещается в матрице на один байт влево, третья строка смещается на два байта влево, а четвёртая строка смещается на три байта влево. Первая строка остаётся без изменений.

Операция «Mixcolumns» - операция перемешивания столбцов. В отличие от операции сдвига строк, которая работает со строками в матрице состояния 4x4, операция перемешивания столбцов обрабатывает столбцы. В принципе, необходимо выполнить только умножение матрицы. Чтобы сделать эту операцию обратимой, не используются обычные сложение и умножение. В AES используются операции в поле Галуа.

Операция добавления ключа раунда проста. Осуществляется операция исключающего ИЛИ с соответствующими байтами исходных данных и полученным ключом.

Операция «Keyexpansion» означает процесс, в результате которого из 128 бит изначального ключа формируется одиннадцать ключей раундов длиной 128 бит.

Чтобы получить ключ раунда  $(n+1)$  из ключа раунда  $(n)$ , выполняются следующие действия:

1. Получают новый первый столбец следующего ключа раунда. Сначала все байты старого четвёртого столбца необходимо заместить с помощью операции «Subbytes». Эти четыре байта сдвигаются вертикально на один байт, а затем выполняется логическая операция исключающего ИЛИ этих байтов со старым первым столбцом. В результате этих действий получается новый первый столбец;

2. Столбцы со 2 по 4 нового ключа раунда рассчитываются так:

- а) [новый второй столбец] = [новый первый столбец] XOR [старый второй столбец]

- б) [новый третий столбец] = [новый второй столбец] XOR [старый третий столбец]

- в) [новый четвёртый столбец] = [новый третий столбец] XOR [старый четвёртый столбец]



## 2 АРХИТЕКТУРА

### 2.1 Схема Архитектуры

Архитектура приложения состоит из пяти основных составляющих, они показаны на рисунке 6

1. Клиента (браузер).
2. Веб-сервера nginx.
3. Веб-сервера unicorn.
4. Веб-приложения.
5. База данных.

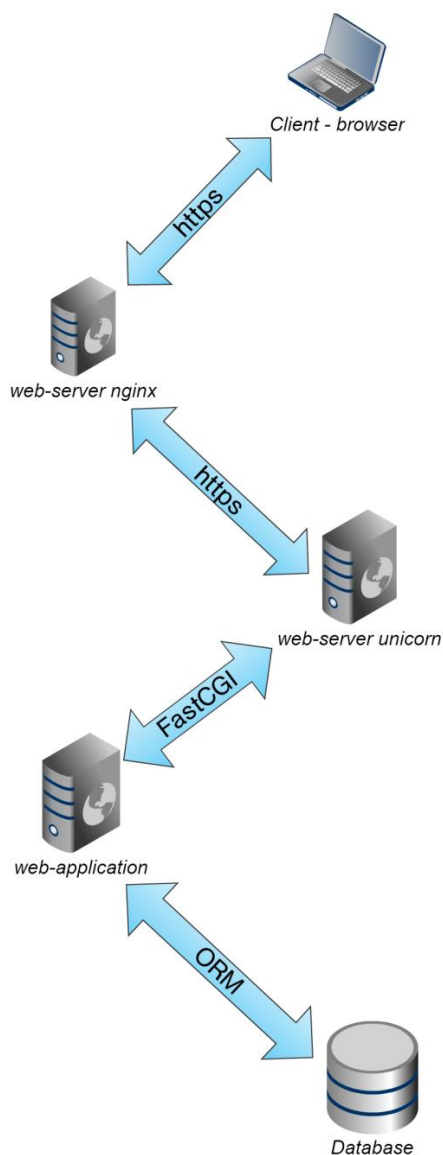


Рис. 6. Схема архитектуры приложения менеджер паролей

## 2.2 Описание архитектуры

### 2.2.1 Регистрация

Для того чтобы новому пользователю пройти процедуру регистрации, необходимо, перейдя на url адрес ресурса Менеджер паролей, ввести регистрационные данные, такие как логин, пароль, в роли логина выступает электронный адрес. Передача данных между клиентской базой, далее браузером, и Веб-сервером, далее сервер, происходит по защищенному каналу SSL. Сервер, получив запрос о регистрации нового пользователя, обращается к базе данных с проверкой существования такой учетной записи, если такая запись уже существует, сервер формирует сообщение о том, что данная учетная запись уже используется, необходимо ввести новые данные. Если такой учетной записи нет - создается новая учетная запись с присвоением идентификатора, в поле пароль хранится хэш-функция пароля, вычисляемая адаптивной криптографической хеш-функцией bcrypt. Сервером формируется запрос на подтверждение регистрации, по указанному электронному адресу отправляется письмо, если в течение установленного сервером времени нет подтверждения о регистрации (по умолчанию 24 часа), учетная запись удаляется. Совершив переход по сгенерированной ссылке, пользователь подтверждает регистрацию, в поле activated происходит смена значения на true.

### 2.2.2 Доступ к учетной записи

Для того чтобы зарегистрированный пользователь мог получить доступ до своей учетной записи, ему необходимо ввести идентификационные данные, а именно пароль доступа. Поиск учетной записи происходит по электронному адресу. Если такой записи нет, пользователь извещается, что такого логина не существует, либо данные были введены неверно. Если логин совпадает, сервер, предварительно вычислив хэш-функцию пароля, сравнивает с хэш-функцией пароля хранимой в базе данных.

При неверном вводе пароля более трех раз, доступ к учетной записи закрывается на установленный временной интервал (по умолчанию 1 час). В

случае успеха, клиенту возвращается ответ с токеном. В дальнейшем для общения с сервером используется только этот токен, вместо логина и пароля. Токен существует до следующей авторизации.

### 2.2.3 Защита данных

Получив доступ к своей учетной записи, пользователь может приступать к просмотру и редактированию данных. Для того чтобы добавить новую заметку, состоящую из полей логин, пароль и ресурс, необходимо ввести мастер-пароль, который будет использоваться для шифрования. При создании новой записи вызывается js-функция для шифрования данных. Записи, которые добавляет авторизованный пользователь, поступают на сервер в зашифрованном виде по каналу передачи данных https. Для шифрования используется симметричный алгоритм блочного шифрования AES с размером блока 128 бит, в качестве ключа шифрования используются первые 128 бит хэш-функции мастер-пароля.

Для просмотра записей, которые предоставляются сервером в зашифрованном виде, данные должны быть расшифрованы. Необходимо ввести мастер пароль в нужное поле. Вызванная js-функция расшифровывает данные, пользователь может приступать к просмотру своих заметок.

### 3 ОПИСАНИЕ ПРОГРАММНОЙ РЕАЛИЗАЦИИ

#### 3.1 Uml диаграмма

Ниже представлена uml диаграмма приложения, построенного на основе модели MVC (Model-View-Controller), показано на рисунке 7

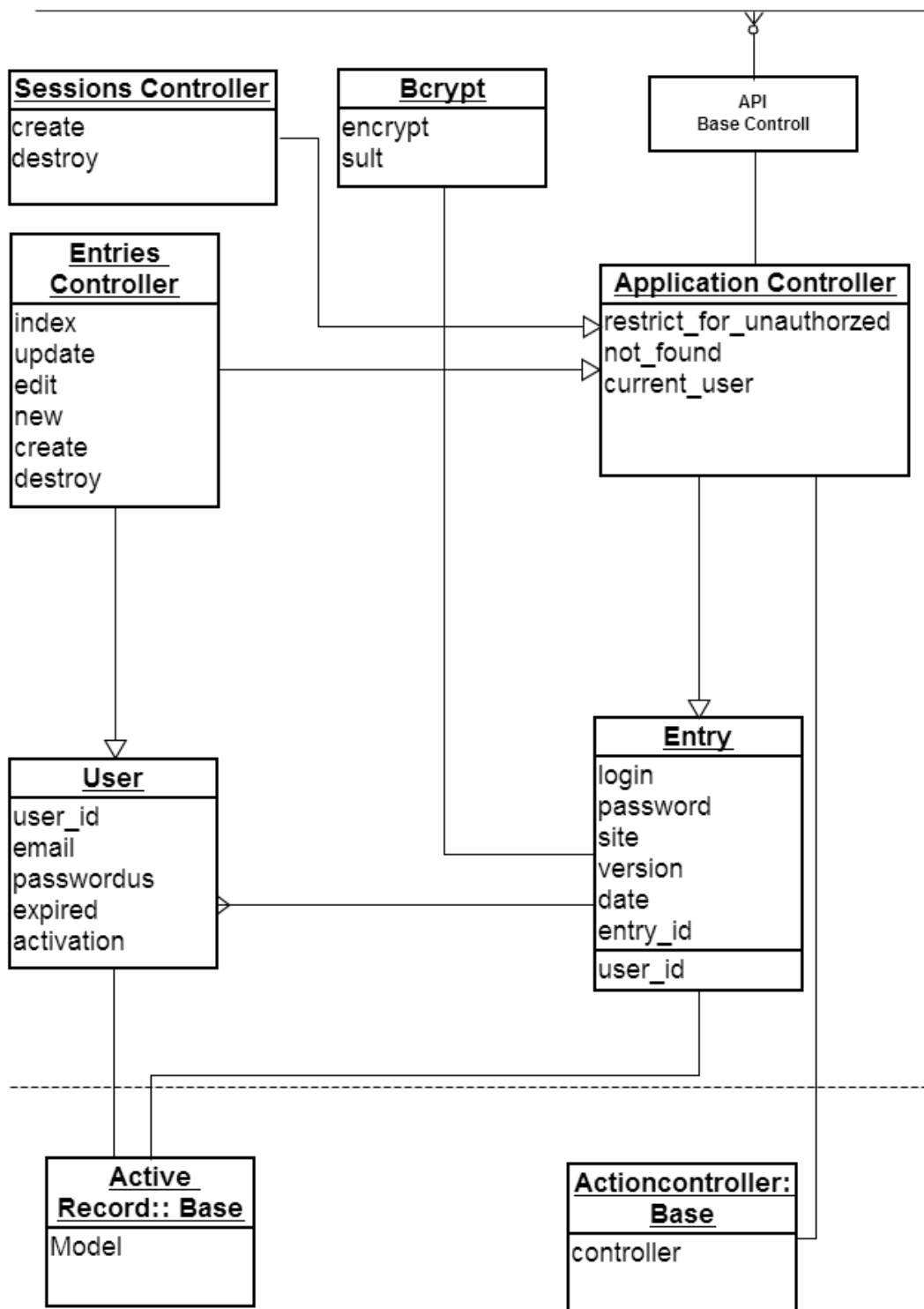


Рисунок 7 – Uml диаграмма

### 3.2 Описание классов

Sessions controller:

1. Create – создание сессии.
2. Destroy – завершение сессии.

Entries controller:

1. Index – список всех заметок.
2. Update – обновление.
3. Edit – изменение заметки.
4. New – страница новой заметки.
5. Create – создание заметки.
6. Destroy – удаление заметки.

Application controller:

1. Restrict\_for\_unauthorized – если не авторизирован, редирект на страницу входа.
2. Not found – страница не найдена.
3. Current\_user – текущий пользователь.

Общение с базой данных.

Entry:

1. Login – логин заметки.
2. Password – пароль заметки, зашифрованный мастер паролем.
3. Site – ресурс.
4. User – принадлежность к пользователю.
5. Version – версия изменений.

User:

1. Email – логин пользователя.
2. Password – хэш функция пароля пользователя.
3. Activation – активирована ли учетная запись.
4. Expired – срок истечения.

### 3.3 Зависимость времени шифрования от длины передаваемых данных. Время, затраченное на вычисление хэш-функции

Для того чтобы проанализировать время затраченное на шифрование передаваемых данных, был написан скрипт на языке Ruby (смотрите приложение А). Итоговые результаты зависят от:

1. Длины передаваемых данных (параметр length).
2. Параметра cost в хэш-функции bcrypt.
3. Количества повторных вычислений (по умолчанию, count = 50).

В качестве результатов представлены длины с параметром length = 20, 30, 40, 50, 60, 70, 80, 90, 100. Время, затраченное на 50 вычислений, измеряется в секундах, рассмотрено в таблице 2

Таблица 2 – Вычисление времени, затраченного на шифрование (в секундах)

length	Count (bcrypt)	Real time
20	15	15.128543
	10	3.784697
	5	0.068778
30	15	15.093400
	10	3.781647
	5	0.068839
40	15	15.097829
	10	3.781545
	5	0.068968
50	15	15.124013
	10	3.799697
	5	0.078889
60	15	15.114301
	10	3.821614
	5	0.088831

## Продолжение таблицы 2

(в секундах)

70	15	15.093829
	10	3.691447
	5	0.070064
80	15	15.140229
	10	3.894135
	5	0.076938
90	15	15.300612
	10	3.781545
	5	0.089739
100	15	15.0978291
	10	3.781545
	5	0.068968

Из результатов тестирования можно сделать вывод, что количество передаваемых символов (проверено число символов, не превышающих значение равное 100) не влияет в значительной мере на скорость шифрования. Разницы в скорости найти не удалось.

Чтобы проверить влияние параметра `cost` на время выполнения хэш-функции, были произведены замеры, которые приведены в таблице 3.

Таблица 3 – Время, затраченное на вычисление хэш-функции (в секундах)

Cost(bcrypt)	Real time (count = 50)
5	0.127938
6	0.251132
7	0.487970
8	0.968343
9	1.924664

10	3.841469
11	7.385415
12	14.517403
13	28.373055
14	53.851162
15	103.932742

По полученным результатам можно сделать вывод, что при увеличении параметра cost на единицу время вычисления хэш-функции возрастает, показано на рисунке 7. На каждом шаге время в среднем увеличивается в 1,951.

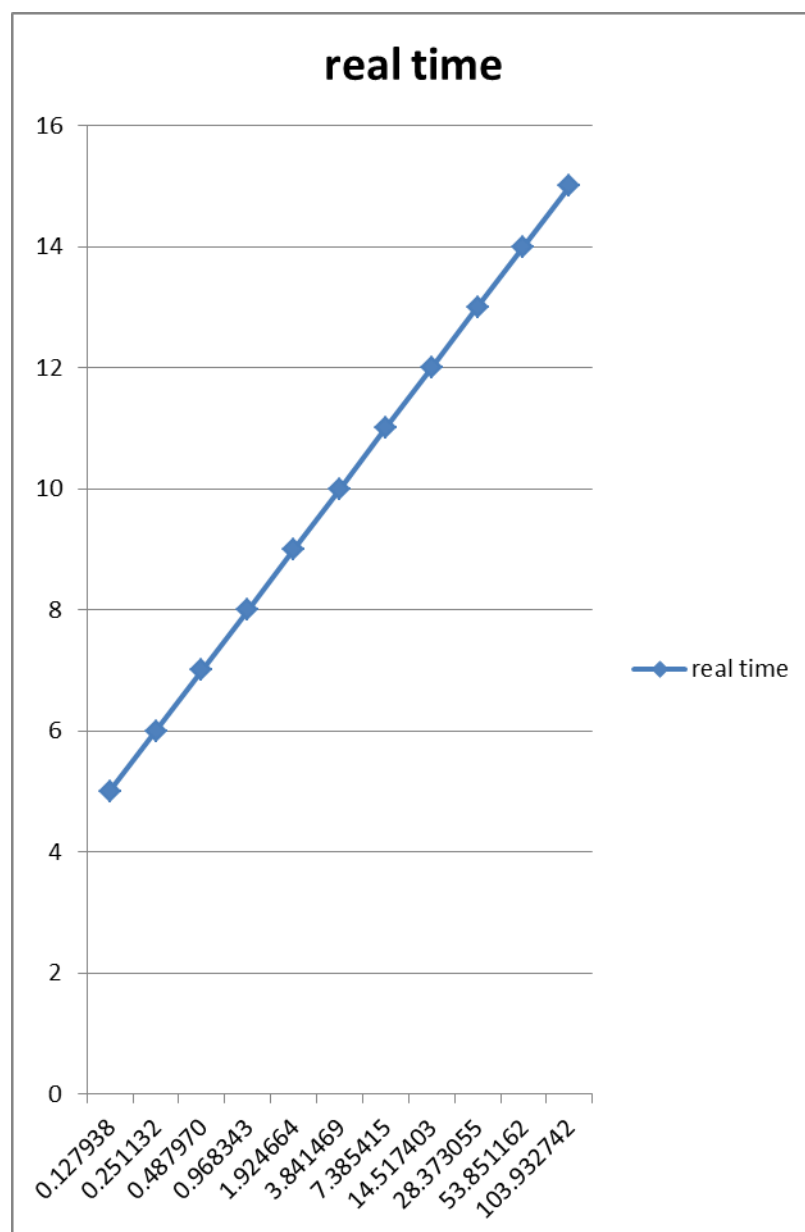




Рисунок 7 – График зависимости значения cost от времени вычисления

### 3.4 База данных

Архитектура базы данных состоит из двух таблиц user и entry, показано на рисунке 8.

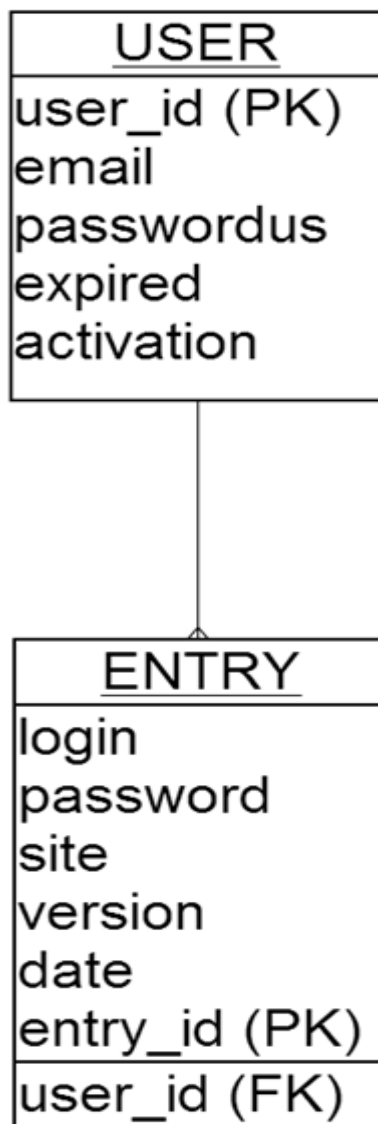


Рисунок 8 – Архитектура базы данных

В таблице USER хранится информация о самой учетной записи:

1. email – логин.
2. passwordus – пароль пользователя.
3. expired – временной штамп.
4. activation – активирована ли учетная запись.

5. user\_id – идентификатор пользователя.

Таблица ENTRY состоит из полей:

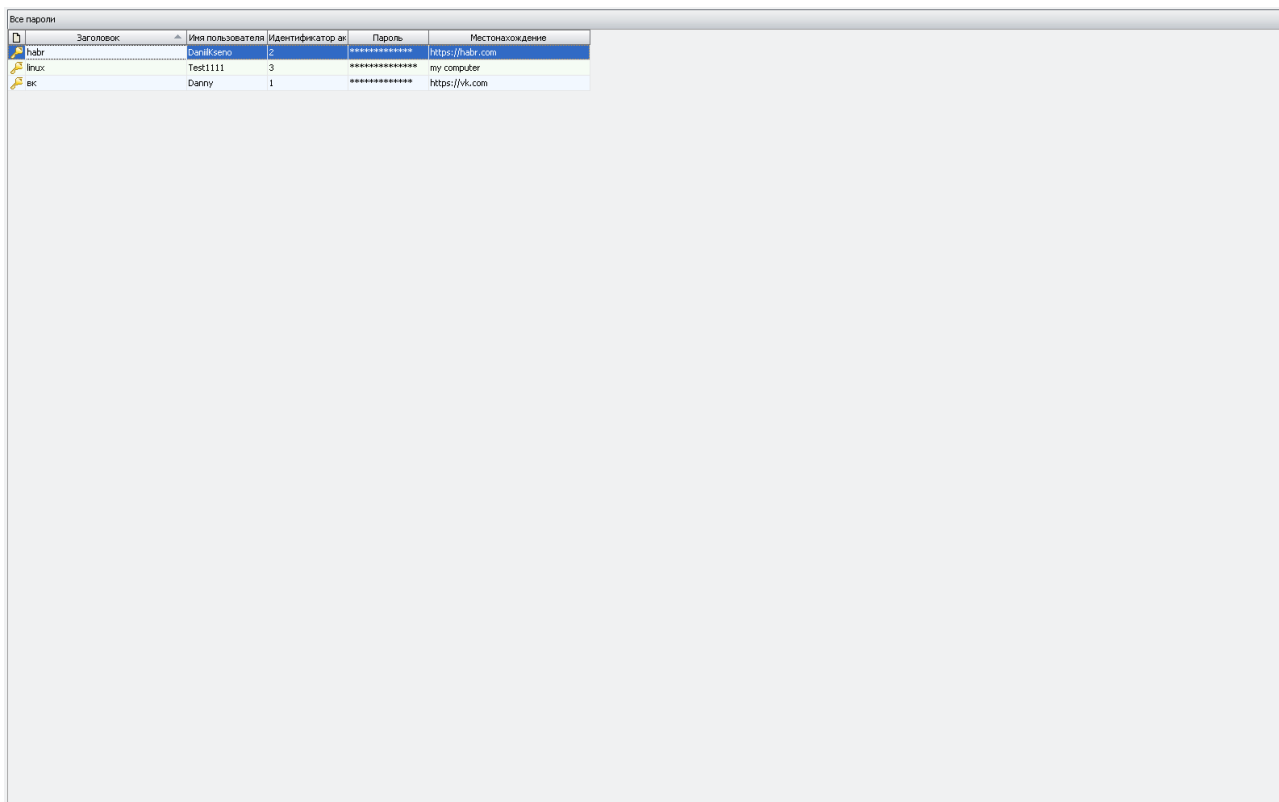
1. login – логин ресурса.
2. password – пароль от ресурса.
3. site – название ресурса.
4. version – версия изменения данных.
5. date – время добавления заметки.
6. entry\_id – идентификатор заметки.

Типы данных можно увидеть в приведенном SQL коде.

```
CREATE DATABASE manager_paroley OWNER ivan ENCODING 'UTF8';
CREATE TABLE user (
  user          integer PRIMARY KEY,
  email         varchar(80),
  password      varchar(80),
  expired       timestamp,
  activation    boolean,
);
CREATE TABLE entry (
  entry         integer PRIMARY KEY,
  login         varchar(80),
  password      varchar(80),
  site          varchar(80),
  version       integer,
  date          date,
  user_id       integer references user(user_id)
```

### 3.5 Интерфейс

На странице пользователя реализована возможность поиска записей, редактирования записанных паролей и учётных записей пользователя, а также добавления новых, как показано на рисунке 9.



The screenshot shows a window titled "Все пароли" (All passwords) with a table of saved credentials. The table has five columns: "Заголовок" (Header), "Имя пользователя" (Username), "Идентификатор ак" (Account ID), "Пароль" (Password), and "Местонахождение" (Location). Three entries are visible:

Заголовок	Имя пользователя	Идентификатор ак	Пароль	Местонахождение
habr	DaniilKseno	2	*****	https://habr.com
лпхх	Test1111	3	*****	my computer
vk	Danny	1	*****	https://vk.com

Рисунок 9 – Страница пользователя

Для восстановления пароля необходимо активировать функцию `Forgot password`. По запросу пользователя на его электронный адрес средствами фреймворка `Active Mailer` отправляется письмо для восстановления пароля.

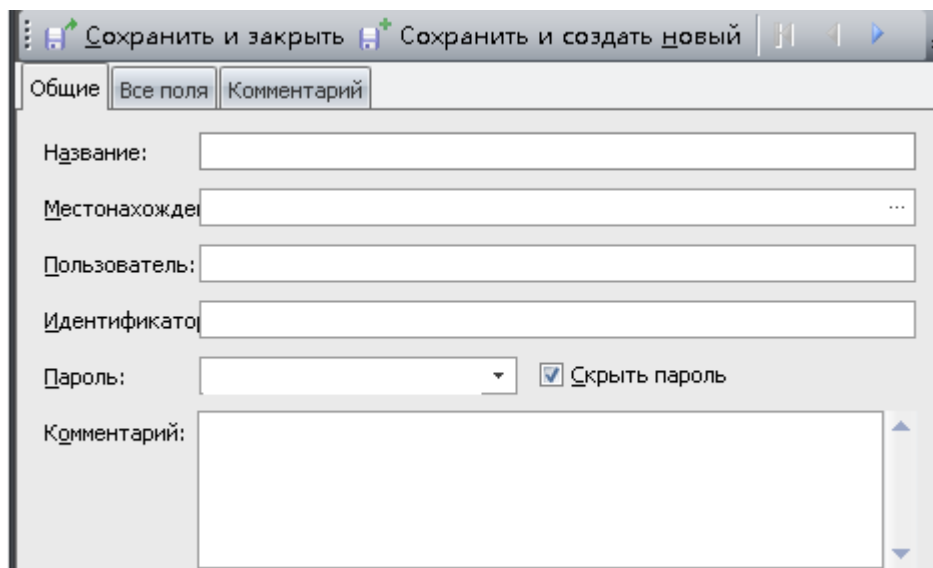


Рисунок 10 – Окно создания новой записи

Здесь необходимо ввести название ресурса, пароль от которого создается, далее указать местоположение (то есть сайт) на котором зарегистрирован пользователь. Затем ввести имя пользователя сайта, после этого указать идентификатор (не обязательно, так как он устанавливается автоматически), ну и непосредственно перейти к созданию самого пароля для учётной записи.

В приложении реализована возможность самостоятельной записи пароля а также его непосредственной генерации встроенным генератором паролей

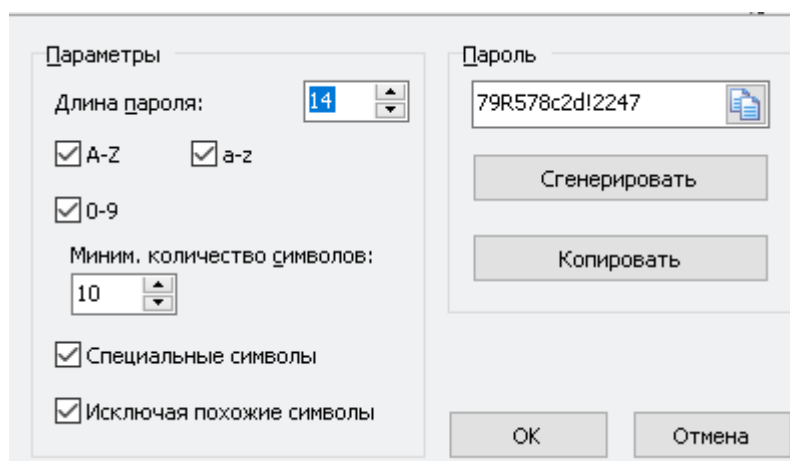


Рисунок 11 – Окно генерации пароля

Как видно на рисунке, имеется возможность включить в пароль специальные символы, такие как знаки препинания, восклицательный и вопросительный знак. Можно выбрать заглавные и прописные буквы и цифры,

а так же выставить необходимую длину пароля. Так же сделана возможность исключить похожие символы в генерируемом пароле, дабы сделать его более безопасным.

### **3.6 Используемые средства для разработки менеджера паролей**

Для разработки использовался динамический язык программирования ruby 1.9.

Сервер написан с использованием фреймворка ruby on rails 3.2. Данный язык считается одним из лучших инструментов для быстрой разработки сложных ресурсов, а также на многих платформах присутствует его интерпретатор. Фреймворк ruby on rails работает со многими серверами СУБД, а так же его можно разворачивать на веб серверах nginx и unicorn.

Была выбрана адаптивная криптографическая хеш-функция bcrypt. Так как функция является адаптивной ее можно замедлить, для усложнения атаки перебором. На сегодняшний день считается одной из менее изученных.

В качестве хранилища данных используется реляционная база данных. Решение выбора не ограничивается одной базой данных, по умолчанию используется PostgreSQL.

### **3.7 Установка необходимых компонентов**

Необходимые настройки и компоненты:

1. Проверяем актуальность пакетов.

```
sudo apt-get update
```

2. Установка RVM (Ruby Version Manager) необходима для использования нескольких версий ruby.

```
sudo apt-get install git-core curl #Это для того, чтобы заработала установка Rvm. curl -L https://get.rvm.io | bash -s stable --ruby  
type rvm | head -1
```

3. Ставим все необходимые пакеты.

```
Rvm sudo apt-get install build-essential openssl libreadline6 libreadline6-dev  
curl git-core zlib1g zlib1g-dev libssl-dev libyaml-dev libsqlite3-dev sqlite3 libxml2-  
dev libxslt-dev autoconf libc6-dev ncurses-dev automake libtool bison subversion
```

4. Устанавливает ruby через RVM.  
rvm install 1.9.3
5. Для использования гемов потребуется установить RubyGems.  
rvm rubygems current
6. Установка ruby on rails.  
gem install rails
7. Для установки nginx [8].  
sudo apt-get install nginx
8. Создадим пользователя nginx.  
sudo useradd nginx
9. Файл хостов, добавленных в черный список;  
touch /etc/nginx/blockips.conf
10. Для запуска nginx и nginx.conf используем init скрипт (Приложение Б).
11. Устанавливаем PostgreSQL.  
sudo apt-get install postgresql postgresql-client postgresql-server-dev\#! /bin/sh
12. Для установки unicorn используем Gem 'unicorn', конфигурация (Приложение В).

## ЗАКЛЮЧЕНИЕ

В работе рассмотрено решение актуальной проблемы – хранения множества учетных записей.

Работа над дипломным проектом позволила ознакомиться со средствами и методиками разработки серверных приложений, не включенных в курс обязательных дисциплин. Были произведены замеры скорости выполнения хеш-функции и шифрования данных, на основе результатов сделаны соответствующие выводы. Было изучено множество методов шифрования данных, рассмотрено большое количество сред разработки.

В процессе выполнения дипломной работы были выполнены все поставленные задачи:

1. Разработана концептуальная схема работы менеджера паролей.
2. Правильно подобран нужный алгоритм шифрования паролей.
3. Разработана структура приложения на основе модели MVC под фреймворк ruby on rails.
4. Менеджер паролей оснащен интуитивным и понятным интерфейсом.
5. Конечный продукт был успешно протестирован и используется создателем по назначению.

Цель: разработка менеджера паролей с использованием методов криптографического шифрования была достигнута.

## БИБЛИОГРАФИЧЕСКИЙ СПИСОК

- 1 Бабаш, А. В. История криптографии. Часть I / А.В. Бабаш, Г.П. Шанкин. – М.: Гелиос АРВ, 2016. – 240 с.
- 2 Кузьмин, Т. В. Криптографические методы защиты информации: моногр. / Т.В. Кузьмин. – Москва: Огни, 2013. – 192 с.
- 3 Хоффман, Л. Дж. Современные методы защиты информации / Л.Дж. Хоффман. – Москва: 2014. – 264 с.
- 4 Литвинская, О. С. Основы теории передачи информации. Учебное пособие / О.С. Литвинская, Н.И. Чернышев. – Москва: 2015. – 168 с.
- 5 AES128 – Implementation for Encryption and Decryption URL: <http://www.ti.com/lit/an/slaa397a/slaa397a.pdf>.
- 6 Язык программирования Ruby. Юкиhiro Мацумото, Дэвид Флэнаган. СПб.: Питер, 2011. 494 с.
- 7 Ruby on Rails Tutorial by Michael Hartl. URL: <http://russian.railstutorial.org/chapters/beginning>.
- 8 Ruby on Rails по-русски. URL: <http://rusrails.ru/rusrails.all.pdf>.
- 9 Оби Фернандес. Путь Rails. Подробное руководство по созданию приложений в среде Ruby on Rails. СПб.: Символ-Плюс, 2008. 768 с.
- 10 Tom Copeland, Anthony Burns. Deploying Rails: Automate, Deploy, Scale, Maintain, and Sleep at Night., 2012. 240 с.
- 11 PostgreSQL documentation. URL: <http://www.postgresql.org/files/documentation/pdf/9.1/postgresql-9.1-US.pdf>.
- 12 Nginx documentation URL: <http://nginx.org/ru/docs/> (дата обращения: 18.04.2022).



## ПРИЛОЖЕНИЕ А

Скрипт на языке ruby, вычисляет затраченное на шифрование время с учетом длины данных и значения параметра cost хеш-функции bcrypt.

```
require 'digest/sha1'
require 'bcrypt'

require 'aes'

require 'benchmark'

MIN = 20 # Мин длина
MAX = 100 # макс длина
COST_MIN = 5
COST_MAX = 15
COUNT = 50 # Число итераций на каждое вычисление
password = 'password'*10
(MIN..MAX).each do |i|
  pass = password[0..i]
  puts
  puts "Password length = #{i}"
  Benchmark.bm(20) do |x|
    (COST_MIN..COST_MAX).each do |cost|
      x.report("BCrypt (cost = #{cost}:") { COUNT.times {
BCrypt::Password.create(pass, :cost => cost) } }
      puts BCrypt::Password.create(pass, :cost => cost)
    end
    x.report("AES:") { 50.times { aes = AES.new(pass); aes.encrypt(pass); } }
  end
end
```

ПРИЛОЖЕНИЕ Б  
Скрипт запуска nginx и nginx.conf.

```
EXEC_PATH="/usr/local/nginx/sbin/nginx"
case "$1" in
    start)
        echo "Starting NginX"
        start-stop-daemon --start --exec $EXEC_PATH
        ;;
    stop)
        echo "Stopping NginX"
        start-stop-daemon --stop --exec $EXEC_PATH
        ;;
    restart)
        echo "Stopping NginX"
        start-stop-daemon --stop --exec $EXEC_PATH
        sleep 1
        echo "Starting NginX"
        start-stop-daemon --start --exec $EXEC_PATH
        ;;
    *)
        echo "Usage: {start|stop|restart}"
        exit 1
        ;;
esac
exit 0

worker_processes 1; # указываем один процесс.
```

## Продолжение ПРИЛОЖЕНИЯ Б Скрипт запуска nginx и nginx.conf.

user ivan ivan; # Пользователь с правами которого запускается worker - он же пользователь, от которого осуществляется деплой.

pid /tmp/nginx.pid; # Задаем местоположение файла с идентификатором текущего мастер-процесса Nginx.

error\_log /tmp/nginx.error.log;

events {

    worker\_connections 1024; # Стандартный показатель количества одновременно открытых соединений рабочего процесса.

    accept\_mutex off;

}

https {

    #стандартные директивы:

    include mime.types;

    default\_type application/octet-stream;

    access\_log /tmp/nginx.access.log combined;

    sendfile on;

    tcp\_nopush on;

    tcp\_nodelay off;

    gzip on;

    upstream myapp\_server {

        server unix:/srv/myapp/shared/unicorn.sock fail\_timeout=0; # Местоположение сокета должно совпадать с настройками файла config/unicorn.rb от корня

    }

Продолжение ПРИЛОЖЕНИЯ Б  
Скрипт запуска nginx и nginx.conf.

```
server {  
    listen 443 default deferred; # прослушивание порта  
    client_max_body_size 1024; # Максимальный размер тела запроса.  
    keepalive_timeout 5;  
    root /srv/myapp/current/public; # Эта строка всегда должна указывать в  
    директорию public Rails приложения.  
    try_files $uri/index.html $uri.html $uri @myapp; # Имя переменной не важно -  
    главное, чтобы в блоке location ниже было аналогичное  
  
    location @myapp {  
        proxy_pass http://myapp_server; # Часть после http:// должна полностью  
        соответствовать имени в блоке upstream выше.  
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;  
        proxy_set_header Host $http_host;  
        proxy_redirect off;  
    }  
  
    error_page 500 502 503 504 /500.html;  
    location = /500.html {  
        root /srv/myapp/current/public;  
    }  
}  
}
```

## ПРИЛОЖЕНИЕ В

### Конфигурация Unicorn

```
deploy_to = "/srv/myapp"
rails_root = "#{deploy_to}/current"
pid_file = "#{deploy_to}/shared/pids/unicorn.pid"
socket_file = "#{deploy_to}/shared/unicorn.sock"
log_file = "#{rails_root}/log/unicorn.log"
err_log = "#{rails_root}/log/unicorn_error.log"
old_pid = pid_file + '.oldbin'

timeout 30
worker_processes 2 #
listen socket_file, :backlog => 1024
pid pid_file
stderr_path err_log
stdout_path log_file

preload_app true # Мастер процесс загружает приложение, перед тем, как
создавать рабочие процессы.

GC.copy_on_write_friendly = true if GC.respond_to?(:copy_on_write_friendly=)

before_exec do |server|
  ENV["BUNDLE_GEMFILE"] = "#{rails_root}/Gemfile"
end

before_fork do |server, worker|
  # Перед тем, как создать первый рабочий процесс, мастер отсоединяется от
базы.
```

Продолжение ПРИЛОЖЕНИЯ В  
Конфигурация Unicorn

```
defined?(ActiveRecord::Base) and
ActiveRecord::Base.connection.disconnect!

if File.exists?(old_pid) && server.pid != old_pid
  begin
    Process.kill("QUIT", File.read(old_pid).to_i)
  rescue Errno::ENOENT, Errno::ESRCH
    # someone else did our job for us
  end
end
end

after_fork do |server, worker|
  defined?(ActiveRecord::Base) and
  ActiveRecord::Base.establish_connection
End
```