МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**
Кафедра математической кибернетики и компьютерных наук

# РАЗРАБОТКА WEB-ПРИЛОЖЕНИЯ ДЛЯ БЕЗОПАСНОГО ХРАНЕНИЯ ПАРОЛЕЙ

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

Студентки 4 курса 451 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Кожиной Ольги Олеговны

Научный руководитель
доцент, к. ф.-м. н. _____ Г. Г. Наркайтис

Заведующий кафедрой
к. ф.-м. н. _____ С. В. Миронов

Саратов 2019

**СОДЕРЖАНИЕ**

# ВВЕДЕНИЕ

Пароли являются важным элементом компьютерной безопасности, помогающим сохранить конфиденциальные данные пользователей в секрете, а также обеспечить информационную защиту серверов, устройств, баз данных, файлов и так далее.

Современные парольные политики становятся все строже: пароли должны быть уникальными для каждого сайта и для каждой учетной записи, а также состоять из цифр, букв разного регистра и специальных символов, чтобы обе-зопасить его владельца от взлома. Следуя этим правилам, пользователь накопит большой набор сложных и длинных паролей, удержать в памяти которые не представляется возможным.

Web-менеджер паролей — это способ решить проблему хранения чувствительной информации. Кроссбраузерность и кроссплатформенность делают его удобным и доступным помощником в повседневной жизни. Остается лишь запомнить один главный пароль для входа в программу, а также периодически его менять.

Целью настоящей работы является разработка web-приложения на языке Java для безопасного хранения паролей с использованием технологий Spring Framework и Hashicorp Vault, позволяющее пользователю аутентифицироваться с помощью социальной сети Facebook и сервиса для разработки IT-проектов Github. Для достижения поставленной цели необходимо решить следующие задачи:

1. исследовать технологии, необходимые для разработки web-приложения;
2. рассмотреть механизмы, обеспечивающие безопасность Vault Hashicorp, а также настроить этот инструмент;
3. проанализировать взаимодействие с использованием протокола авторизации OAuth 2.0;
4. спроектировать и создать приложение.

**Структура и объем работы.** Бакалаврская работа состоит из раздела «Определения, обозначения и сокращения», введения, двух разделов основной части, заключения, списка использованных источников и трех приложений. Общий объем работы — 53 страницы, из них 43 страницы — основное содержание, включая 18 рисунков, цифровой носитель в качестве приложения, в списке использованных источников содержится 21 наименование.

# 1 Краткое содержание работы

## 1.1 Первый раздел работы

Раздел посвящен рассмотрению теоретических основ использованных в работе технологий: дан обзор клиент-серверной архитектуры, рассмотрен протокол авторизации OAuth 2.0, описаны механизмы безопасности Vault Hashicorp, а также проанализировано использование фреймворка Spring.

**Клиент-серверная архитектура.** В этом подразделе описано взаимодействие в сети Интернет, для которого используется клиент-серверная архитектура — тип сетевой архитектуры, в рамках которой вычислительные мощности и сетевая нагрузка распределены между серверами — некими программными компонентами, выполняющими сервисные функции — и клиентами, запрашивающими «услуги» сервера. И клиент, и сервер являются программным обеспечением, причем чаще всего они находятся на различных рабочих станциях и взаимодействуют друг с другом удаленно посредством сетевых протоколов. Так как к серверу, как правило, обращается множество клиентов, он размещается на отдельной особым образом настроенной станции с высоким уровнем производительности.

Приложение, реализующее описанную выше клиент-серверную архитектуру называют веб-приложением [1].

**Обзор OAuth 2.0.** В этом подразделе описан открытый стандарт авторизации OAuth, позволяющий клиентам получать доступ к защищенным ресурсам сервера от имени владельца ресурса. Владельцем ресурса может быть другой клиент или конечный пользователь. OAuth также помогает конечным пользователям разрешать третьим лицам доступ к своим ресурсам на сервере без необходимости передавать собственные регистрационные данные, такие как имя пользователя и пароль. Полная система авторизации OAuth 2.0 описана в RFC 6749 [2].

OAuth определяет четыре роли:

- Владелец ресурса — пользователь, который авторизует приложение для доступа к своему аккаунту. Доступ приложения к пользовательскому аккаунту ограничен «областью видимости» (scope) предоставленных прав авторизации (например, доступ на чтение или запись);
- Клиент — приложение, которое хочет осуществить доступ к аккаунту пользователя. Перед осуществлением доступа приложение должно быть

авторизовано пользователем, а авторизация должна быть одобрена со стороны API;

— Сервер ресурсов — непосредственно хранит защищённые данные аккаунтов пользователей.
— Авторизационный сервер — проверяет подлинность информации, предоставленной пользователем, а также создаёт авторизационные токены для приложения, с помощью которых приложение будет осуществлять доступ к пользовательским данным.

Теперь, имея представление о ролях, используемых в OAuth, рассмотрим абстрактное описание протокола.

1. Приложение запрашивает у пользователя авторизацию на доступ к серверу ресурсов.
2. Если пользователь авторизует запрос, приложение получает разрешение на авторизацию (authorization grant).
3. Приложение запрашивает авторизационный токен у сервера авторизации путём предоставления информации о самом себе и разрешении на авторизацию от пользователя.
4. Если подлинность приложения подтверждена и разрешение на авторизацию действительно, сервер авторизации создаёт токен доступа для приложения. Процесс авторизации завершён.
5. Приложение запрашивает ресурс у сервера ресурсов, предоставляя при этом токен доступа для аутентификации.
6. Если токен действителен, сервер ресурсов предоставляет запрашиваемый ресурс приложению.

Фактический порядок шагов описанного процесса может отличаться в зависимости от используемого типа разрешения на авторизацию, но в целом процесс будет выглядеть описанным образом [3].

**Обзор Vault Hashicorp.** В этом подразделе описан Vault Hashicorp — это инструмент для безопасного доступа к секретам, где секреты — это все, к чему необходимо жестко контролировать доступ, например, ключи API, пароли или сертификаты. Vault предоставляет унифицированный интерфейс доступа к любому секрета, обеспечивая строгий контроль над ним и ведение подробного журнала событий. Vault также реализует шифрование как сервис с централизованным управлением ключами, чтобы упростить шифрование данных для

передачи и хранения в облачных хранилищах и центрах обработки данных.

Анализ внешних угроз. В архитектуре Vault существует три системы: клиент, который общается с Vault посредством API, Vault или более точно сервер, который предоставляет API и отвечает на запрос, а также хранилище, которое сервер использует для чтения и записи данных.

Между клиентом и сервером Vault нет взаимного доверия. Клиенты используют TLS для установления безопасного канала связи. Хранилище данных также не является доверенным, поэтому Vault использует барьер безопасности для всех запросов к хранилищу. Защитный барьер автоматически шифрует все данные, покидающие Vault, с использованием 256-битного шифра Advanced Encryption Standard (AES).

Анализ внутренних угроз. В системе Vault критической проблемой безопасности является злоумышленник, пытающийся получить доступ к секретным материалам, к которым у них нет прав доступа. Это становится внутренней угрозой, если у злоумышленника есть некоторый уровень доступа к Vault и он может пройти проверку подлинности.

Vault поддерживает использование «принципа четырёх глаз» (правило, согласно которому для принятия решения требуется одобрение не одного человека, а нескольких людей, которые входят в руководство) для распечатывания хранилища с использованием схемы разделения секрета Шамира. Vault запускается в запечатанном состоянии. Это означает, что ключ шифрования, необходимый для чтения и записи из хранилища, еще не известен. Процесс распечатывания требует предоставления главного ключа, из которого было получить ключ шифрования. Риск распространения главного ключа заключается в том, что один злоумышленник, имеющий доступ к нему, может расшифровать все хранилище. Вместо этого схема Шамира позволяет разделить главный ключ на несколько долей или частей. Количество общих частей и необходимый порог настраиваются, но по умолчанию Vault генерирует 5 частей, любые 3 из которых должны быть предоставлены для восстановления главного ключа [4].

**Обзор Spring.** В данном подразделе дано описание фреймворку Spring как облегченной платформы для построения Java-приложений. Ядро Spring Framework основано на принципе внедрения зависимостей (Dependency lnjection — DI), когда создание и управление зависимостями между компонентами становятся внешними задачами [5]. Паттерн внедрения зависимостей также

отображен в этой работе. Для того, чтобы заменить одну реализацию работы с vault-сервером на другую, достаточно реализовать интерфейсы из модуля SecretService. Таким образом, DI позволяет писать слабосвязный код.

В этой работе используется несколько модулей технологии Spring Framework, а именно:

- Spring Boot — позволяет создавать полноценные приложения производственного класса с минимумом усилий;
- Spring MVC — главной целью MVC является разделение объектов, бизнес-логики и внешнего вида приложения. Все эти компоненты слабо связаны между собой и при желании можно изменить, например, внешний вид приложения, не внося существенные изменения в остальные два компонента;
- Spring Data — миссией Spring Data является предоставление единой модели программирования с использованием Spring для доступа к данным, сохраняя при этом специальные черты базового хранилища;
- Spring Security — это JEE фреймворк, предоставляющий механизмы построения систем аутентификации и авторизации, а также другие возможности обеспечения безопасности для корпоративных приложений, созданных с помощью Spring Framework [6]. Для взаимодействия по OAuth в работе используется Spring Security OAuth.

Также стоит отметить библиотеки spring-vault-core и spring-mustache, которые являются обертками и необходимы для минимизации количества кода и большего удобства для работы с Vault или Mustache.

## 1.2 Второй раздел работы

Раздел посвящен разработке собственного web-приложения на практике. Описываемое приложение представляет собой web-менеджер с несколькими видами аутентификации и страницей управления секретами. Стоит отметить, что web-приложение, в отличие от десктопных и мобильных аналогов, не имеет жесткой привязки к устройству, поэтому его можно открывать на различных устройствах, что обеспечивает кроссбраузерность и кроссплатформенность.

На рисунке 1 представлены модули приложения, которые далее будут подробно описаны.

**Настройка Vault.** В данном подразделе описывается настройка инструмента Vault. Установить Vault довольно просто: необходимо скачать предва-

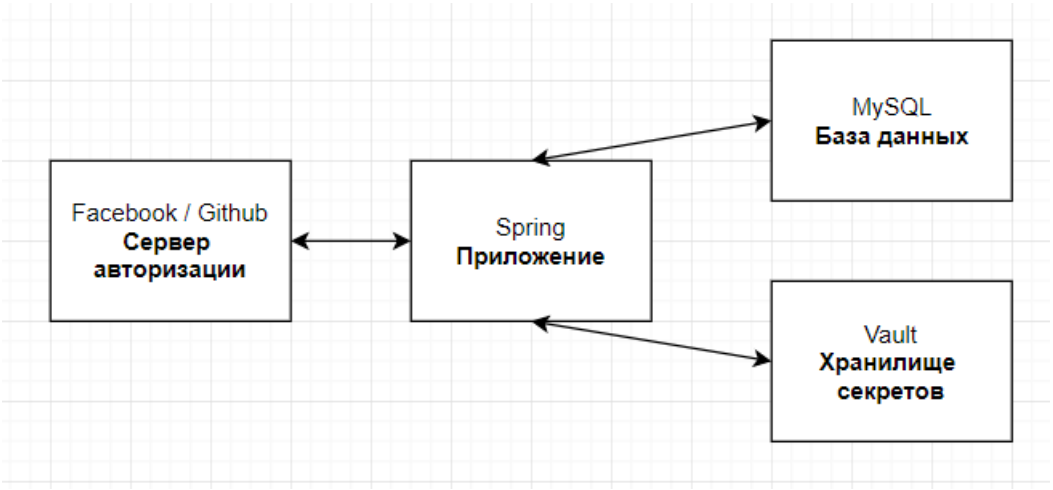


Рисунок 1 – Структура модулей в приложении

рительно скомпилированный бинарный файл, подходящий вашей системе, разархивировать и прописать путь к файлу в переменной *PATH*.

Чтобы проверить, что Vault был правильно установлен, необходимо запустить команду *vault -h*. Если на экране появится интерфейс команды помощи, то Vault был установлен успешно.

Для того, чтобы запустить Vault сервер на локальной машине, необходимо выполнить следующие шаги:

1. Необходимо настроить защищенное соединение между приложением и Vault, то есть передавать данные по протоколу HTTP с настроенным TLS для того, чтобы информация, которой обмениваются клиент и сервер, была доступна только этому клиенту и этому серверу, а не третьим лицам (например, провайдеру или администратору Wi-Fi-сети). Для этого нужно сгенерировать самоподписанный сертификат с использованием расширения SAN и добавить сертификат в JVM keystore [7];
2. Запустить Vault сервер с параметрами, среди которых будет присутствовать сгенерированный сертификат и публичных ключ;
3. Запустить команды: для инициализации Vault-сервера, для распечатывания Vault-сервера, для включения механизма секретов kv для хранения произвольных секретов по заданному пути.

**Настройка базы данных и работа с ней.** В данном подразделе описывается работа с базой данных. В качестве базы данных применялась реляционная база данных MySQL, а как средство по работе с ней — фреймворк Hibernate. Разработанная структура базы данных минималистична и содержит только ту

информацию, которая не является чувствительной, то есть, раскрытие, модификация или сокрытие которой не может привести к ощутимому убытку или (денежному) ущербу [8], поэтому пароль к учетной записи и другие секреты пользователя хранятся в Vault (рис. 2).

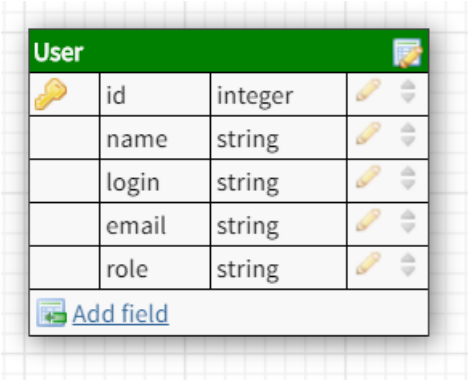


Рисунок 2 – Таблица «Пользователи»

Средства Spring Data и Hibernate позволяют работать с записями в таблице как с обычными Java-объектами; для этого классы-сущности помечаются специальными аннотациями. Так, например, в работе используется аннотация @GeneratedValue, которая идет в паре с аннотацией @Id и указывает на то, что при создании нового объекта первичный ключ будет автоматически сгенерирован. Стандартные функции работы с данными CRUD выполнены согласно описанию в теоретической части дипломной работы.

**Разработка механизма аутентификации.** В данном подразделе описывается несколько способов аутентификации, предложенных в работе: аутентификация с регистрацией в самом приложении, а также аутентификация по OAuth 2.0 с помощью Facebook и Github.

Чтобы пользоваться аутентификацией через приложение, необходимо зарегистрироваться. Зарегистрировавшись с помощью формы регистрации, пользователь может войти в приложение, указав логин и пароль. Пользователь также может войти в приложение, нажав на иконки социальных сетей в нижней части формы. В этом случае пользователь будет отправлен на страницу социальной сети, где ему необходимо будет ввести свой логин и пароль для входа в социальную сеть, а также подтвердить разрешение на взаимодействие с приложением.

Реализация взаимодействия с использованием OAuth 2.0. Приложение использует Client Credentials Grant для взаимодействия по OAuth2. Для это-

го приложение нужно зарегистрировать в необходимых сервисах. Клиентский механизм OAuth 2.0 отвечает за доступ к защищенным OAuth 2.0 ресурсам других серверов. Конфигурация включает в себя описание соответствующих защищенных ресурсов, к которым пользователи могут иметь доступ, поэтому в файле application.yml необходимо прописать clientId и clientSecret, а также некоторые другие параметры, полученные на шаге регистрации сервиса. Кроме того, следует добавить собственный фильтр в цепочку фильтров сервлетов Spring Security.

**Разработка клиентской части приложения.** В данном подразделе описан интерфейс главной страницы разработанного приложения, а также механизм защиты клиентской части.

Рассмотрим функционал главной страницы приложения (рис. 3), который является формой управления секретами пользователя. Здесь можно добавить секрет, вписав название и значение секрета в соответствующие поля и нажав на кнопку в виде плюса, увидеть список секретов. Просмотреть секрет можно, нажав на кнопку в виде глаза на строке с секретом; удалить секрет можно, нажав на кнопку в виде корзины на строке с секретом. В верхнем правом углу располагается кнопка выхода из системы.

Для наполнения главной страницы использовались CSS-стили, которые расширяют возможности HTML-верстки посредством управления местоположением элементов на странице, описания их цвета, размера и свойств [9]. Помимо собственных стилевых файлов также широко применялся Bootstrap — свободный набор инструментов для создания сайтов и веб-приложений. Включает в себя HTML- и CSS-шаблоны оформления для типографики, веб-форм, кнопок, меток, блоков навигации и прочих компонентов веб-интерфейса, включая JavaScript-расширения.

Обеспечение безопасности клиентской части. Подделка межсайтовых запросов (CSRF) — это атака, которая вынуждает конечного пользователя выполнять нежелательные действия в веб-приложении, в котором они в настоящий момент проходят проверку подлинности. CSRF-атаки специально направлены на запросы об изменении состояния, а не на кражу данных [10]. Для защиты от CSRF-атак нам необходимо убедиться, что в запросе есть что-то, что вредоносный сайт не может предоставить, например, случайно сгенерированный токен в качестве параметра.

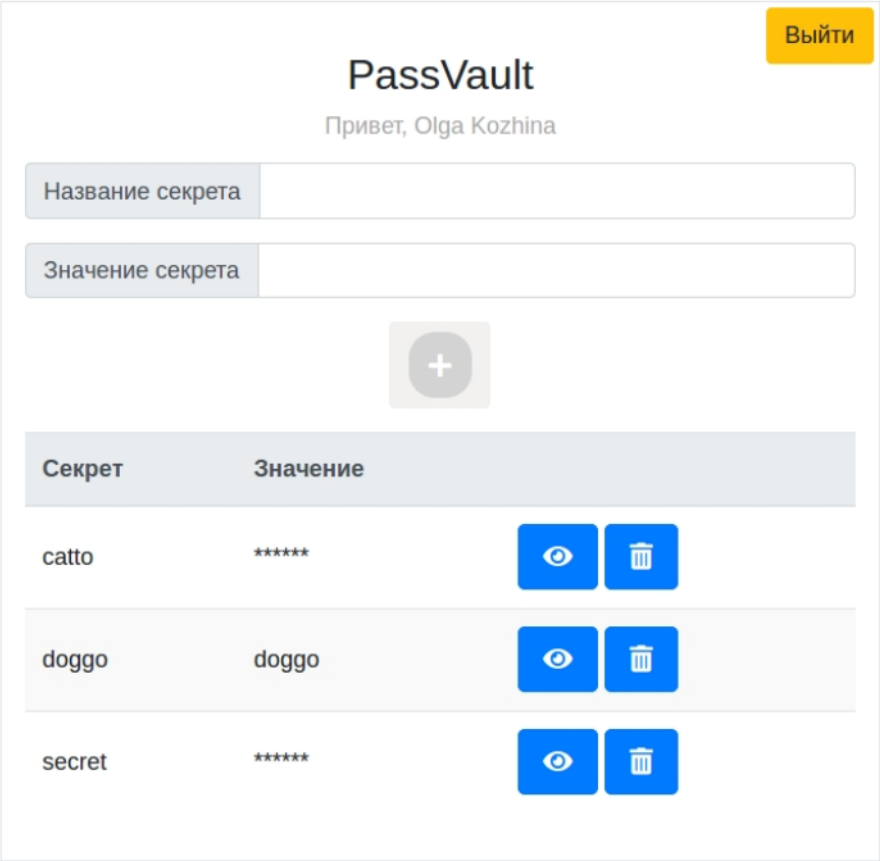


Рисунок 3 – Главная страница приложения

По умолчанию в Spring Security включена защита от CSRF, поэтому необходимы только изменения в клиентской части. На каждой странице, на которой происходит вызов POST-методов, присутствует строка:

```
<input type="hidden" name="_csrf" value="{{_csrf.token}}"/>
```

**Развертывание приложения в контейнере сервлетов.** В данном подразделе описаны технологии, при помощи которых происходит развертывание приложения в контейнере. Для этого использовался Maven — инструмент для автоматической сборки проект, который создает конечный файл проекта на основе его спецификации в конфигурационном файле, самостоятельно скачивая необходимые библиотеки и подключая их к проекту. Полученный файл впоследствии развертывается на встроенном в Spring Boot контейнере сервлетов Tomcat. Таким образом, в клиент-серверной модели в роли клиента выступает браузер, а сервер заменяет программное обеспечение на той же машине — Apache Tomcat. В связке с Maven он позволяет быстро увидеть результаты работы в действии — прямо в окне браузера.

# ЗАКЛЮЧЕНИЕ

В ходе работы были выполнены поставленные цели: подробно рассмотрена архитектура типичного веб-приложения, сделан обзор используемых технологий, фреймворков и библиотек, произведен их анализ. Реализовано приложение с использованием вышеупомянутых технологий, организована работа с базой данных и хранилищем секретов, спроектированы и разработаны классы на сервере и на клиенте, а также организовано взаимодействие с использованием протокола OAuth 2.0.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ


1 Структура веб-приложений [Электронный ресурс]. — URL: https://www.intuit.ru/studies/courses/1139/250/lecture/6422 (Дата обращения 10.05.2019). Загл. с экр. Яз. рус.

2 Спецификация протокола OAuth 2.0 [Электронный ресурс]. — URL: https://tools.ietf.org/html/rfc6749 (Дата обращения 20.05.2019). Загл. с экр. Яз. англ.

3 Блог IT-компании DigitalOcean [Электронный ресурс]. — URL: https://www.digitalocean.com/community/tutorials/oauth-2-ru (Дата обращения 20.05.2019). Загл. с экр. Яз. рус.

4 Vault Documentation [Электронный ресурс]. — URL: https://www.vaultproject.io/docs/what-is-vault/index.html (Дата обращения 22.05.2019). Загл. с экр. Яз. англ.

5 *Шеффер, К.* Spring 4 для профессионалов / К. Шеффер, К. Хо, Р. Харроп. — Киев: Вильямс, 2015.

6 Spring Documentation [Электронный ресурс]. — URL: https://docs.spring.io/ (Дата обращения 15.05.2019). Загл. с экр. Яз. англ.

7 Хранилище ключей и сертификатов [Электронный ресурс]. — URL: http://java-online.ru/keystore-keytool.xhtml (Дата обращения 18.05.2019). Загл. с экр. Яз. рус.

8 Чувствительная информация [Электронный ресурс]. — URL: https://www.finam.ru/dictionary/wordf03208/ (Дата обращения 21.05.2019). Загл. с экр. Яз. рус.

9 *Хеник, Б.* HTML и CSS. Путь к совершенству / Б. Хеник. — Санкт-Петербург: Питер, 2011.

10 OWASP Foundation [Электронный ресурс]. — URL: https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF) (Дата обращения 23.05.2019). Загл. с экр. Яз. англ.