Министерство образования и науки РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Рыбинский государственный авиационный технический университет

имени П.А. Соловьева»

Факультет радиоэлектроники и информатики

Кафедра Математическое и программное обеспечение ЭВС

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

**Дипломная работа**

Разработка системы централизованной авторизации на основе LDAP и современных протоколов аутентификации

на соискание квалификации – Бакалавр программной инженерии

по профилю – Разработка программно-информационных систем

по направлению – 09.03.04 Программная инженерия

Соискатель, студент группы ИПБ-15 Козин О.И.

*(Код) (Подпись, дата) (Фамилия И.О.)*

Руководитель, доцент Волков М.Л.

*(Уч. степень, звание) (Подпись, дата) (Фамилия И.О.)*

Консультант

по экономике к.т.н., доцент Клементьева Н.А.

*(Уч. степень, звание) (Подпись, дата) (Фамилия И.О.)*

Нормоконтролер ст.преп. Задорина Н.А.

*(Уч. степень, звание) (Подпись, дата) (Фамилия И.О.)*

К защите допустить

Зав. кафедрой к.ф.-м.н., профессор Шаров В.Г.

*(Уч. степень, звание) (Подпись, дата) (Фамилия И.О.)*

Рыбинск 2019

Содержание

[Введение 5](#_Toc11361352)

[1 Анализ предметной области 7](#_Toc11361353)

[1.1 Описание предметной области 7](#_Toc11361354)

[1.2 Формулировка проблемы и актуальность 22](#_Toc11361355)

[1.3 Обзор аналогов 22](#_Toc11361356)

[2 Программная документация 24](#_Toc11361357)

[2.1 Техническое задание на программное обеспечение 24](#_Toc11361358)

[2.1.1 Общие сведения 24](#_Toc11361359)

[2.1.1.1 Наименование программы 24](#_Toc11361360)

[2.1.1.2 Плановые сроки начала и окончания работ 24](#_Toc11361361)

[2.1.1.3 Порядок оформления и предъявления заказчику результатов работ 24](#_Toc11361362)

[2.1.1.4 Перечень нормативно-технических документов, методических материалов, использованных при разработке ТЗ 24](#_Toc11361363)

[2.1.1.5 Определения, обозначения, сокращения 25](#_Toc11361364)

[2.1.2 Назначение и цели создания системы 25](#_Toc11361365)

[2.1.2.1 Назначение системы 25](#_Toc11361366)

[2.1.2.2 Цели создания системы 26](#_Toc11361367)

[2.1.3 Характеристика объектов автоматизации 26](#_Toc11361368)

[2.1.3.1 Краткие сведения об объекте автоматизации 26](#_Toc11361369)

[2.1.3.2 Существующие программное обеспечение 27](#_Toc11361370)

[2.1.4 Требования к системе 27](#_Toc11361371)

[2.1.4.1 Требование к функциональным характеристикам 27](#_Toc11361372)

[2.1.4.1.1 Перечень подсистем 27](#_Toc11361373)

[2.1.4.1.2 Требования к способам и средствам связи для информационного обмена между компонентами системы 27](#_Toc11361374)

[2.1.4.1.3 Требования к режимам функционирования системы 27](#_Toc11361375)

[2.1.4.1.4 Перспективы развития, модернизации системы 28](#_Toc11361376)

[2.1.4.2 Требования к надежности 28](#_Toc11361377)

[2.1.4.3 Надежность программного кода 28](#_Toc11361378)

[2.1.4.5 Требования к интерфейсу 28](#_Toc11361379)

[2.1.4.6 Требования к защите информации от несанкционированного доступа 29](#_Toc11361380)

[2.1.4.7 Требования по сохранности информации при авариях 29](#_Toc11361381)

[2.1.5 Требования к функциям, выполняемым системой 29](#_Toc11361382)

[2.1.6 Нефункциональные требования 30](#_Toc11361383)

[2.1.7 Требования к видам обеспечения 30](#_Toc11361384)

[2.1.7.1 Требования к лингвистическому обеспечению системы 30](#_Toc11361385)

[2.1.7.2 Требования к программному обеспечению системы 30](#_Toc11361386)

[2.1.7.3 Требования к техническому обеспечению 31](#_Toc11361387)

[2.2 Пояснительная записка к программному обеспечению 31](#_Toc11361388)

[2.2.1 Назначение и область применения 31](#_Toc11361389)

[2.2.2 Технические решения 32](#_Toc11361390)

[2.2.2.1 Постановка задачи на разработку программы 32](#_Toc11361391)

[2.2.2.2 Описание структуры программной системы 32](#_Toc11361392)

[2.2.2.3 Описание технологии программирования 33](#_Toc11361393)

[2.2.2.5 Описание и обоснование выбора метода организации входных и выходных данных 34](#_Toc11361394)

[2.2.2.6 Описание алгоритмов и функционирования программы 34](#_Toc11361395)

[2.2.3 Ожидаемые технико-экономические показатели 36](#_Toc11361396)

[2.3 Описание программы 36](#_Toc11361397)

[2.3.1 Общие сведения 36](#_Toc11361398)

[2.3.1.1 Необходимое программное обеспечение 36](#_Toc11361399)

[2.3.1.2 Используемые языки программирования 37](#_Toc11361400)

[2.3.2 Функциональное назначение 37](#_Toc11361401)

[2.3.3 Описание логической структуры 37](#_Toc11361402)

[2.3.4 Входные и выходные данные 38](#_Toc11361403)

[2.4 Программа и методика испытаний 38](#_Toc11361404)

[2.4.1 Объект испытаний 38](#_Toc11361405)

[2.4.2 Цель испытаний 39](#_Toc11361406)

[2.4.3 Требования к программе 39](#_Toc11361407)

[2.4.4 Средства и порядок испытаний 41](#_Toc11361408)

[2.4.5 Методы испытаний 41](#_Toc11361409)

[3 Эксплуатационная документация на программный продукт 43](#_Toc11361410)

[3.1 Руководство администратора 43](#_Toc11361411)

[3.1.1 Назначение программного продукта 43](#_Toc11361412)

[3.1.2 Необходимые технические характеристики сервера 43](#_Toc11361413)

[3.1.3 Установка 43](#_Toc11361414)

[3.1.4 Интерфейс пользователя 43](#_Toc11361415)

[3.1.5 Интерфейс администратора 45](#_Toc11361416)

[4 Акт испытаний программного продукта 48](#_Toc11361417)

[5 Экономическое обоснование 49](#_Toc11361418)

[5.1 Краткая характеристика программного продукта 49](#_Toc11361419)

[5.2 Расчёт себестоимости разработки программной системы 50](#_Toc11361420)

[5.2.1 Материальные затраты 50](#_Toc11361421)

[5.2.2 Заработная плата с отчислением на социальные нужды 50](#_Toc11361422)

[5.2.3 Амортизационные отчисления и эксплуатационные расходы, связанные с использованием ЭВМ 51](#_Toc11361423)

[5.2.4 Накладнее расходы 53](#_Toc11361424)

[5.2.5 Полная себестоимость 53](#_Toc11361425)

[5.3 Оценка экономической эффективности разработки программной системы 54](#_Toc11361426)

[5.4 Вывод 55](#_Toc11361427)

[Заключение 56](#_Toc11361428)

[Список литературы 57](#_Toc11361429)

[Приложения 58](#_Toc11361430)

[Приложение А – примеры программного кода 58](#_Toc11361431)

Введение

В современном мире, с его бурным развитием информационных технологий, с каждым днём становится всё больше и больше сфер подверженных автоматизации с помощью программных продуктов.

Количество информации растёт, её ценность повышается. Вопрос защиты информации от несанкционированного доступа важен как никогда. Особенно это актуально в сфере коммерческого программного обеспечения, в которой необходимо обеспечение защищённости информации как от внешних угроз, так и поддержание ограничений на доступ к данным по причине разделения полномочий внутри круга доверенных лиц.

Информационное системы усложняются, за счёт увеличения количества предоставляемых сервисов. А разнообразие типов пользователей ещё сильнее усложняет организацию доступа к системам. Например, для корпоративного пользователя (работника компании) очень важно иметь единую точку управления его учётной записью. Чтобы оперативно реагировать на изменение статуса работника. Ведь, когда работник уходит в отпуск, необходимо в короткие сроки заморозить его учётную запись для всех доступных ему сервисов во избежание нарушения безопасности. А для частного пользователя важен вопрос хранения персональных данных, и управления ими. Кроме того, таким пользователям необходим более гибкий механизм идентификации, так как их данные могут довольно часто манятся, а доступ к сервисам должен оставаться возможным.

Поэтому для нужд разных пользователей существуют различные методы и протоколы аутентификации. И хотя применение каждого из них в своём случае оправдано, в целом такой подход ещё больше усложняет процессы доступа к сервисам. Возникает ситуация, при которой либо каждый сервис должен уметь работать с множеством протоколов аутентификации, либо пользователь должен использовать отдельную учётную запись для каждого сервиса. Оба этих варианта неудачны. Первый ведёт к значительному повышению затрат на разработку сервисов. А второй, из-за множества логинов и паролей, делает использование сервисов пользователем крайне неудобным, и снижает безопасность.

Решить эту проблему можно путём внедрения посредника между сервисами и различными методами аутентификации. Этот посредник должен работать с разными протоколами аутентификации, и иметь единый интерфейс для сервисов. При таком подходе разработка сервисов упрощается, так как нужно уметь работать только с посредником. А, для пользователей становится неважен способ аутентификации, если он поддерживается посредником.

Таким образом разработка подобной системы централизованной аутентификации позволит упростить работу разработчиков сервисов, и повысит удобство пользователей.

1 Анализ предметной области

1.1 Описание предметной области

Процессы аутентификации и авторизации позволяют реализовать разделения прав доступа и защиту конфиденциальной информации от внешних угроз. Это необходимо в большинстве программных продуктов, особенно коммерческих. Также реализация конкретной системы может значительно повысит удобство работы пользователей, что зачастую ведёт ещё большему увеличению безопасности.

Для успешного прохождения авторизации необходимо пройти несколько этапов:

1) Идентификация – заявление о том, кем является пользователь, обычно выражается в предоставлении своего уникального идентификатора-имени.

2) Аутентификация – проверка подлинности пользователя. То есть действительно ли пользователь тот, за кого себя выдаёт. Самый распространённый способ — это подтверждение с помощью секретного пароля. Кроме того, аутентификация может быть односторонней и двусторонней (взаимной). При односторонней, проверки требует только тот, кто запрашивает доступ. При двусторонней запрашиваемый ресурс тоже должен подтвердить себя.

3) Авторизация – определение соответствующих прав доступа к запрашиваемому ресурсу. Этот процесс позволяет использовать результаты работы аутентификации и выдать пользователю только то, что ему разрешается согласно его статусу.

Успешное последовательное прохождение этих этапов позволяет гарантировать защищённость информации, к которой осуществляется доступ.

Но, так как информация имеет ценность, то найдутся люди, желающие ей завладеть. При этом они могут использовать различные методы чтобы нарушить нормальное функционирование этапов авторизации или получить доступ к конфиденциальной информации напрямую. Поэтому появляется необходимость в полной защите данных: как при хранении, так и при передаче.

На практике процессы авторизации обычно происходят с использованием компьютерной сети, которая не может быть полностью ограждена от физического доступа злоумышленника. В таком случае приходится учитывать, что любая информация, передаваемая по сети может быть известна кому угодно. Следовательно, необходимо использовать методы, позволяющие преобразить данные так, что даже при их перехвате неавторизированным пользователем конфиденциальность не будет нарушена.

Такие методы предоставляет криптография. Криптография - это наука об обеспечении безопасности данных. Она занимается поисками решений четырех важных проблем безопасности - конфиденциальности, аутентификации, целостности и контроля участников взаимодействия. Основным способом решения этих проблем является шифрование.

Шифрование - это преобразование данных определённым способом-алгоритмом в нечитабельную форму, с возможностью возвращения исходного состояния, используя специальные ключи шифрования.

На практике приходится считаться со сложными структурами данных и с аппаратные ограничениями, поэтому отдельных алгоритмов шифрования недостаточно. Нужны комплексные решения описывающие все этапы передачи данных, способные надёжно и эффективно решать проблему защиты информации. Такими решениями являются криптосистемы.

Они включают в себя набор правил и алгоритмов, регламентирующих манипуляции с данными, а также способы создания и распространения ключей. Криптографические системы можно классифицировать по типу используемых алгоритмов шифрования:

1. На основе симметричных алгоритмов.
2. На основе асимметричных алгоритмов.

Симметричные криптосистемы, также называемые системами с секретным ключом – способ шифрования, в котором один ключ используется как для шифрования, так и для расшифровывания. Является первым подходом к сокрытию информации, изобретённым человеком.

Главной особенностью симметричной криптосистемы является то, что ключ алгоритма шифрования должен оставаться в секрете от посторонних. Поэтому должна обеспечиваться защита ключа на протяжении всего его существования.

Кроме возможности потери конфиденциальности ключа, также существует опасность применения криптоанализа к зашифрованным данным. Поэтому выбранный алгоритм шифрования должен быть достаточно стойким для недопущения расшифровки информации за приемлемое время.

Для симметричных алгоритмов уровень криптостойкости обычно определяется размером ключа шифрования, что эквивалентно полному перебору его значений.

Асимметричные криптосистемы в своей работе используют пару ключей: закрытый и открытый. Открытый ключ применяется для шифрования и распространяется между участниками общения в открытом виде. Расшифровка производится с помощью закрытого ключа, который должен оставаться в секрете.

Реализация асимметричных алгоритмов шифрования основана на применении односторонних функций, то есть таких функций, что, зная несложно найти значение F(x), но вычисление x по F(x) невозможно за приемлемое время. Теоретически, зная F(x), x можно подобрать, перебирая все возможные значения x и применять к ним функцию до тех пор, пока результат не совпадёт с F(x). Однако практически, при достаточной размерности множества X, выполнить такой перебор за приемлемое время невозможно.

Классический пример такой функции – это функция, основанные на принципе обратного модульного возведения в степень, также называемые задачей дискретного логарифмирования.

Но получается, что односторонняя функция не может использоваться в качестве функции шифрования. Так как хоть она и позволяет надёжно зашифровать данные, но расшифровать их некто не сможет. Для решения этой проблемы используются односторонние функции с секретом. Это особый вид функций, которые имеют некоторый секрет, позволяющий быстро вычислить обратное значение функции. Именно на этом принципе основаны большинство асимметричных криптографических систем.

В итоге оба подхода имеют свои преимущества и недостатки. Симметричные системы проще в реализации, за счёт более простых операций, а скорость шифрования в среднем на несколько порядков выше, чем у аналогичных по защищённости систем с открытым ключом. Единственным серьёзным недостатком является сложность обмена секретного ключа. Поэтому для применения симметричных систем на практике необходимо решить проблему надёжного распределения ключей между участниками процесса передачи данных.

При работе в публичных сетях, требуется быть уверенным в подлинности ключа. А именно в том, что полученный от пользователя ключ действительно сгенерирован им. Такой функционал может предоставить механизм электронной подписи.

Обычно, в большинстве реальных систем, электронная подпись основана на принципах асимметричного шифрования, только применяет их иначе. Так же генерируется пара ключей: открытый, закрытый. Но для шифрования (подписи) данных применяется закрытый, соответственно для расшифровки (проверки подписи) открытый. Такой подход позволяет любому владельцу открытого ключа проверять подпись, а возможность подписать, остаётся только за хранителем закрытого ключа. Таким образом проведя подписание каких-либо информации, мы генерируем электронную подпись и можем отправлять её вместе с исходными данными. Далее, зная отправителя и его открытый ключ, можно применить его к подписи и получить исходный файл.

Однако, получается, что каждый обладатель открытого ключа может получить из подписи оригинальные данные, а они должны быть секретны. Кроме того, подписанный файл по размеру будет сопоставим с оригинальным, что удвоит нагрузку на передающую среду. Поэтому в электронная подпись невозможна без хеш-функций.

Хэш-функция – осуществляет преобразование (хэширование) набора входных данных произвольной длинны, в выходную последовательность определённой длинны. При этом для каждого набора результат хэширования будет уникальным, а провести обратное преобразование невозможно.

Таким образом подписывая не сами данные, а их хэш, общедоступным окажется именно хэш, что безопасно. И размер подписи теперь возможно уменьшить, выбрав подходящую хэщ-функцию.

Одной только проверки подлинности ключей недостаточно для сохранения конфиденциальности информации. Кроме этого, очевидно, необходимо скрывать сами значения ключей. Это можно сделать если использовать асимметричное шифрование на ключи. Так как размеры ключей небольшие, то потеря ресурсов на зашифровку-расшифровку будет незначительной.

Используя цифровую подпись и асимметричное шифрование можно реализовать безопасное распределение ключей для симметричных криптосистем.

Но такая система связи остаётся уязвимой для злоумышленника, который представляется доверенным пользователем, но отдаёт свой открытый ключ асимметричного шифрования, таким образом подменяя его.

Для решения этой проблемы открытый ключ доверенного пользователя вместе с сопроводительной информацией: именем, сроком действия и прочим подписывается центром сертификации. Предполагается, что центр сертификации честный и не подпишет ключ злоумышленника. А также то, что центр сертификации распространяет свой открытый ключ максимально широко, и желающие пройти аутентификацию, ещё до инициализации процесса обмена ключами, будут иметь открытый ключ такого центра, и злоумышленник ничего не сможет с этим поделать.

Этот процесс называется сертификацией. А подписанные центром сертификации открытые ключи, включающие в себя также прочие данные о владельце, называются цифровыми сертификатами.

Таким образом современные протоколы аутентификации и авторизации должны включать в себя множество механизмов обеспечения безопасности.

Протоколы в общем можно классифицировать следующим образом:

1. Клиент-серверные
2. Основанные на генерации одноразовых паролей
3. С использованием доверенного сервера IdP

Самым простым методом клиент-серверной аутентификации является простая аутентификация по паролю. Она предполагает наличие у пользователя и у сервиса пары логин-пароль, с помощью которых и происходит подтверждение личности. Является наиболее не безопасным, так как перехват пароля ведёт к полной потере конфиденциальности. Поэтому все коммуникации должны происходить в защищённой среде. Например, используя HTTPS. Одним из протоколов реализующих такой метод аутентификации является HTTP authentication.

Клиент

Сервер

Запрос ресурса

Запрос на аутентификацию

Учётные данные

Отчёт о выполнении

Рисунок 1. Пример работы HTTP аутентификации со схемой Basic

1. Клиент запрашивает ресурс, используя GET запрос
2. Сервер отвечает о необходимости аутентификации кодом 401
3. Клиент снова запрашивает ресурс и передаёю свои данные в заголовке Authorization в открытом виде.
4. Сервер проводит проверку данных пользователя и в случае успеха возвращает код 200.

Более совершенным методом клиент-серверной аутентификации является аутентификация по сертификатам. Она предполагает наличие доверенного сервера certificate authority (CA). Этот сервер выступает гарантом подлинности сертификатов, которые он подписал и выдают их сервисам. Далее на основе своего сертификата сервис генерирует сертификаты для пользователей. Такие сертификаты содержат данные идентифицирующие конкретного пользователя. Если сервис может расшифровать данные из сертификата пользователя, то такой пользователь считается доверенным.

Клиент

Сервис

Сертификат

подписанный сервисом

Доступ

Рисунок 2. Аутентификация по сертификатам

Также существует дайджест-аутентификация, это клиент-серверный метод, обеспечивающий большую безопасность чем простая парольная схема. Для усложнения криптоанализа передаваемых данных, вместо передачи и хранения учётных данных в открытом виде используется хэш от этих данных. Кроме того, при инициализации соединения с сервисом генерируется случайное число известное обоим участникам диалога, оно добавляется к логину и паролю перед взятием хэша. Владея одинаковыми данными сервер может проверить хэш отправленный клиентом и выдать доступ.

Аутентификация основанная на генерации одноразовых паролей может быть использована как самостоятельный метод аутентификации, либо как дополнительный метод, реализуя концепцию двухфакторной аутентификации. Согласно ей, пользователь должен использовать два типа учётных данных. Например, обычный пароль к учётной записи и одноразовый пароль из смс-сообщения.

Одним из популярных методов аутентификации, основанных на принципе генерации одноразовых паролей, является использование алгоритма HOTP. Это алгоритм защищённой односторонней аутентификации, использующий для генерации хеш-функцию SHA-1. В качестве параметров для вычисления хэша используется общий для клиента и сервера секрет, и специальная метка - счётчик, увеличивающаяся на единицу при возникновении события. В качестве события обычно выступает генерация следующего пароля. Таким образом, зная общий секрет и значение счётчика сервер может произвести те же действия что и клиент и проверить результаты. Использование SHA-1 гарантирует невозможность подмены сообщения и обеспечивает сохранность общего секрета.

Из-за использования счётчика пароли HOTP могут оставаться действительными в течении неограниченного времени, а сам счётчик может терять синхронизацию с счётчиком на сервере, и требовать процедуры синхронизации. Эти недостатки были устранены в алгоритме TOTP.

TOTP также требует общий с сервером секрет, но вторым параметром для хэш-функции является не счётчик, а метка времени. Причём применяется интервал времени, обычно 30 секунд. Таким образом сгенерированный пароль является действительным только в указанный промежуток времени, что повышает безопасность.

Аутентификация с использованием доверенных IdP чаще всего используется для построения больших распределённых систем. В ней функционал по аутентификации делегируется сервисами отдельному приложению – серверу аутентификации или identity provider. Это позволяет упростить структуру сервисов. А также открывает возможность реализовать технологию единого входа Single Sign-On (SSO). Она позволяет однажды аутентифицирующемуся пользователю иметь доступ ко всем сервисам, находящимся в зоне ответственности IdP. По такому же принципу работает вход через учётные записи социальных сетей.

Клиент

IdP

Сервис

Учётные данные

Токен сервиса

Токен сервиса

Отчёт о выполнении

Рисунок 3. Общая схема аутентификации с IdP

В общем виде процесс аутентификации по токену выглядит следующим образом:

1. Клиент проводит аутентификацию с IdP. Способом может быть любым, но обычно это протоколы Kerberos, OpenID Connect и аналогичные.
2. Теперь Клиент может запрашивать у IdP токен для конкретного сервиса.
3. Используя токен, Клиент аутентифицируется в сервисе.

В некоторых ситуациях, например, когда клиентом является браузер невозможно выполнять сложные последовательности запросов. Для таких случаев разработана схема с пассивным клиентом.

Клиент

IdP

Сервис

Запрос учётных данных

Учётные данные

Запрос ресурса

Запрошенный ресурс

Токен

1

2

Перенаправление запроса

3

4

5

6

7

Рисунок 4. Схема аутентификации с пассивным клиентом, с перенаправлением запросов

Формат токенов очень важен при проведении аутентификации, поэтому стандартизирован. Основные используемые форматы токенов:

1. JSON Web Token (JWT)
2. Simple Web Token (SWT)

JWT – это открытии стандарт, который описывает формат сообщений в виде JSON файлов. Согласно нему токен состоит из заголовка, хранящего собственное описание, основной части, в которой хранится пользовательская информация, а также электронной подписи, удостоверяющей весь файл. В стандарте определены специальные имена, определяющие конкретные значения: iss, exp и т.д.

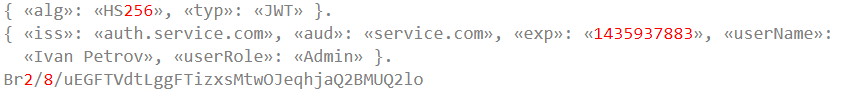


Рисунок 5. Пример не зашифрованного токена формата JWT

SWT – открытый стандарт, имеет простую структуру. Что позволяет просто использовать для передачи заголовки протоколов тапа HTTP. Имеет зарезервированные имена. Использует симметричное шифрование и цифровую подпись.



Рисунок 6. Пример не зашифрованного токена формата SWT

Из существующих протоколов аутентификации с токеном можно выделить наиболее популярные. Одним из таких является Kerberos.

Kerberos может предоставлять как сетевую аутентификацию, так и безопасную авторизацию без необходимости повторного ввода пароля или предоставления других удостоверяющих данных. В

Для обеспечения безопасности протоколом предусмотрен следующий формат проведения аутентификации:

1. Пользователь, желая аутентифицироваться, обращается к KDC и передаёт ему в открытом виде данные о себе и о желаемом ресурсе.
2. KDC, посылает в ответ разрешение на получение разрешения (TGT), которое интерпретируется клиентом как неразбираемый набор бит, а также ключ сессии. Этот ключ генерируется случайным образом и зашифровывается секретным ключом пользователя, хранящимся в базе данных KDC.
3. Теперь пользователь, применяя свой закрытый ключ может получить сессионный ключ. И зашифровав им TGT, отправляет обратно на KDC.
4. KDC применяет сохранённый сессионный ключ получает TGT и если он соответствует исходному, значит пользователь тот за кого себя выдаёт. Генерируется новое сообщение. В него записываются данные необходимые для аутентификации и авторизации (опционально) пользователя сервисом – Service Ticket (ST). А также генерируется новый сессионный ключ, но на этот раз для взаимодействия сервиса и пользователя. Этот ключ вместе с ST шифруется закрытым ключом это сервиса. А чтобы пользователь мог получить доступ к нему, то его копия шифруется сессионным ключом пользователя. Всё вместе отправляется пользователю.
5. Пользователь, расшифровав ключ для общения с сервисом запоминает его. И посылает ST сервису.
6. Сервис расшифровывает ST своим секретным ключом, оттуда же берёт и сессионный ключ. По полученным данным сервис может решать какой доступ предоставить.

OpenId Connect – это аутентификационная надстройка над протоколом авторизации OAuth 2.0. OIDC предполагает наличие отдельного доверенного сервера аутентификации, благодаря чему поддерживает технологию SSO, позволяет отделить функционал по аутентификации от конкретных приложений и гарантирует безопасность. Очень популярный протокол, поддерживается практически всеми крупными компаниями, например: Google, Facebook.

Для передач данных в процессе аутентификации используется стандарт токенов JWT или SWT. Процесс авторизации, в общем случае, выглядит следующим образом:

1. Пользователь запрашивает ресурс.
2. Ресурс перенаправляет пользователя на сервер аутентификации.
3. Пользователь вводит свои учётные данные на форме сервера аутентификации.
4. Сервер аутентификации проверяет данные пользователя по своей базе, и вы даёт токен доступа (id token).
5. Имея id token пользователь может получить доступ к ресурсу.

В результате аутентификации пользователь получает:

1. id\_token - JWT, содержащий информацию об аутентификации, а также дополнительную информацию о пользователе.
2. access\_token - токен, с помощью которого можно запросить дополнительную информацию о пользователе на стороне Idp.

OpenID Connect также регламентирует стандартный способ получения и представления профилей пользователей в качестве набора значений, называемых claims. Это позволяет сервису запрашивать только необходимые ему данные о пользователе.

SAML - язык разметки, основанный на языке XML, предназначен для передачи данных о конкретном пользователе в защищённом виде между участниками процесса аутентификации: IdP, производящим аутентификацию, и сервисом, предоставляющим ресурс. SAML 2.0 определяет синтаксис и семантику утверждений (assertion), относящихся к аутентификации, атрибутам и авторизационной информации. Определены следующие типы утверждений:

1. Authentication Assertion – определяет, что данный субъект прошел аутентификацию определенным способом в определенный момент времени.
2. Authorization Decision Assertion – определяет, на какие действия авторизован конкретный субъект.
3. Attribute Assertion – содержит информацию об атрибутах объекта: почта, пол, и т.д.

Последовательность при выполнении аутентификации в общем случае аналогична таковой в OpenID Connect.

Кроме того, в качестве IdP может использоваться LDAP каталог. LDAP – это протокол описывающий принципы доступа к иерархи данных. Позволяет отвязать реализацию каталога и способ доступа к нему. Благодаря древовидной структуре хорошо подходит для хранения данных о пользователях.

Иерархия каталога состоит из записей. Запись является экземпляром одного (или нескольких) объектного класса. Объектный класс описывает набор атрибутов, которые могут или должны быть у наследуемой записи. Атрибут – именует данные. Для однозначного определения записей при запросах связанных с поиском, требуется обеспечить уникальность записей на одном уровне иерархии. Соответственно конкретная запись определяется цепочкой атрибутов.

В результате классификацию методов аутентификации можно представить в схеме.

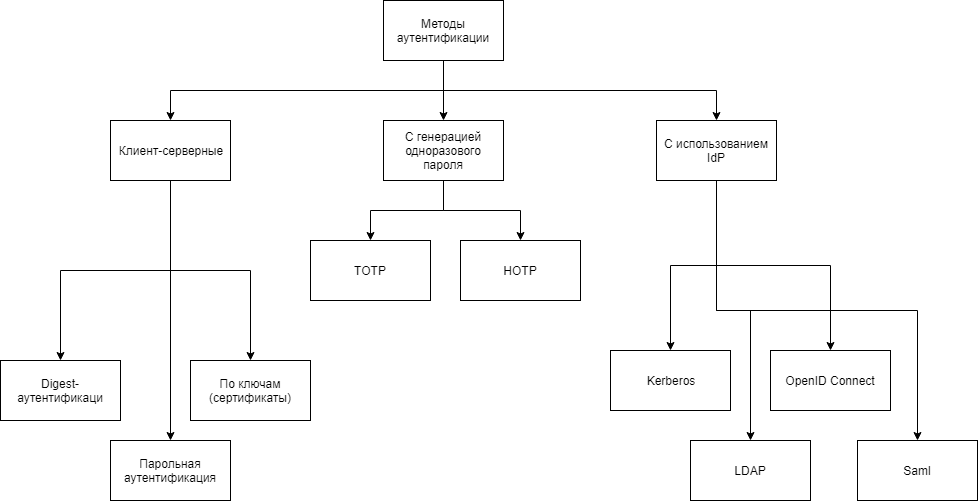


Рисунок 7 Схема классификации методов аутентификации.

1.2 Формулировка проблемы и актуальность

Проблематика заключается в потребности пользователей сервисов ООО «БиАйВи» в агрегации различных методов аутентификации в одну систему. С предоставлением им единого интерфейса аутентификации.

Актуальность состоит в потребности ООО «БиАйВи» в повышении гибкости системы аутентификации и предоставлении пользователям более удобного интерфейса взаимодействия с собственными сервисами, с помощью централизации различных способов аутентификации в одной системе.

1.3 Обзор аналогов

KeyCloak - это open-source сервер аутентификации и управления учетными записями, построенный на базе спецификаций OAuth 2.0, Open ID Connect, JSON Web Token и SAML 2.0. Разрабатывается JBoss.

Преимущества:

* Встроенный механизм управления учетными записями
* Поддержка большого количества платформ
* Поддержка каталогов LDAP
* Поддержка протоколов SAML 2 и OpenID Connect

Недостатки:

* Не поддерживает Kerberos
* Ограниченные возможности по расширению функционала

OpenidDict – это open-sour библиотека, позволяющая разворачивать сервер аутентификации. Поддерживает OAuth 2.0, Open ID Connect.

Преимущества:

* Легко расширяется и дополняется
* Поддержка механизма управления учётными записями

Недостатки:

* Поддерживает только OpenID Connect
* Поддерживает только платформу ASP.NET

2 Программная документация

2.1 Техническое задание на программное обеспечение

2.1.1 Общие сведения

2.1.1.1 Наименование программы

Полное наименование программы: Система централизованной авторизации на основе LDAP и современных протоколов аутентификации.

Краткое наименование: Системы централизованной авторизации.

2.1.1.2 Плановые сроки начала и окончания работ

Плановый срок начала работ по созданию системы централизованной авторизации – 01 ноября 2018 года. Плановый срок окончания работ по разработке системы централизованной авторизации – 15 июня 2019 года.

2.1.1.3 Порядок оформления и предъявления заказчику результатов работ

Система передается в виде функционирующего серверного приложения, установленного на оборудование заказчика в сроки, установленные договором. Совместно с системой производится сдача разработанного комплекта документации.

2.1.1.4 Перечень нормативно-технических документов, методических материалов, использованных при разработке ТЗ

При разработке автоматизированной системы и создании проектно-эксплуатационной документации Исполнитель должен руководствоваться требованиями следующих нормативных документов:

* ГОСТ 19.201-78. ТЕХНИЧЕСКОЕ ЗАДАНИЕ. ТРЕБОВАНИЯ К СОДЕРЖАНИЮ И ОФОРМЛЕНИЮ.
* ГОСТ 34.601-90. Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Стадии создания.
* ГОСТ 34.201-89. Информационная технология. Комплекс стандартов на автоматизированные системы. Виды, комплексность и обозначение документов при создании автоматизированных систем.

2.1.1.5 Определения, обозначения, сокращения

Таблица 1 - Расшифровка определений, обозначений, сокращений

|  |  |  |
| --- | --- | --- |
| № | Наименование | Расшифровка |
| 1 | KDC | Key Distribution Center – центр выдачи ключей |
| 2 | ТЗ | Техническое задание |
| 3 | TGT | Ticket granting ticket – билет на получение билета |
| 4 | IdP | Identity provider – поставщик идентификационных данных |
| 5 | SSO | Single Sign-On – технология единого входа |

2.1.2 Назначение и цели создания системы

2.1.2.1 Назначение системы

Система централизованной авторизации предназначена для агрегации различных протоколов аутентификации и предоставления конечному пользователю единого интерфейса по доступу к сервисам ООО «БиАйВи» удобным ему методом.

2.1.2.2 Цели создания системы

Основной целью создания системы является упрощение процесса аутентификации для пользователей сервисов компании ООО «БиАйВи».

Для реализации поставленных целей система должна решать следующие задачи:

* Предоставление общего интерфейса для пользователей, использующих разные протоколы аутентификации
* Поддерживать работу с LDAP аутентификацией
* Поддерживать работу с системами основанными на протоколе OpenId Connect
* Поддерживать работу с системами на основе протокола SAML
* Реализация технологии Single Sign-On
* Работать с большим количеством запросов

2.1.3 Характеристика объектов автоматизации

2.1.3.1 Краткие сведения об объекте автоматизации

* Система реализует технологию Single Sign-On, что позволяет пользователям, прошедшим процесс аутентификации не вводить свои учётные данные при запросе доступа к сервисам.
* Объектом автоматизации является процесс аутентификации и авторизации.
* Используемые протоколы аутентификации реализуют концепцию доверенной третьей стороны.
* Для доступа к сервису ООО «БиАйВи» пользователь предварительно должен пройти процесс регистрации в нём.
* Каждому методу аутентификации с IdP соответствует один сервер аутентификации.

2.1.3.2 Существующие программное обеспечение

На момент начала работы реализован механизм регистрации пользователей в инфраструктуре сервисов. Кроме того, существуют договорённости с партнёрами ООО «БиАйВи», имеющими свои сервера IdP, позволяющие получить к ним доступ.

Кроме того, реализована простая парольная аутентификация, позволяющая получить доступ к сервисам без SSO.

2.1.4 Требования к системе

2.1.4.1 Требование к функциональным характеристикам

2.1.4.1.1 Перечень подсистем

В состав системы централизованной авторизации должны входить следующие подсистемы:

* Модуль аутентификации Сбербанк Онлайн
* Модуль аутентификации Госуслуги
* Модуль аутентификации LDAP
* Модуль аутентификации SAML

2.1.4.1.2 Требования к способам и средствам связи для информационного обмена между компонентами системы

Компоненты системы независимы друг от друга и все располагаются на одном сервере.

2.1.4.1.3 Требования к режимам функционирования системы

Для системы централизованной авторизации определены следующие режимы:

* Нормальный режим – система функционирует штатно.
* Режим сбоя – система сигнализирует об ошибке и не принимает запросы.

2.1.4.1.4 Перспективы развития, модернизации системы

В дальнейшем планируется увеличение числа поддерживаемых протоколов и систем аутентификации. Также планируется усовершенствование механизма масштабирования.

2.1.4.2 Требования к надежности

Система должна выдерживать длительное функционирование и множественные запросы.

Система должна функционировать без сбоев с вероятностью 0,98 в течении 10000 часов.

Система должна сохранять работоспособность и обеспечивать восстановление своих функций в случае сбоя в системе электроснабжения.

2.1.4.3 Надежность программного кода

Надежность программного кода обеспечивается следованием каскадной модели при разработке программного обеспечения. А именно последовательным выполнением приведенных ниже этапов.

1. Формулировка требований к системе.
2. Проектирование системы.
3. Написание кода модулей системы.
4. Тестирование модулей.
5. Внедрение полученных модулей в систему централизованной авторизации.
6. Тестирование полученной системы
7. Поддержка

2.1.4.5 Требования к интерфейсу

Система должна предоставлять простой интерфейс администратора для управления настройками и просмотра журнала активности пользователей. Интерфейс должен предоставлять функционал по просмотру внутренней базы данных, настройке ролей пользователя, настройке характеристик пароля пользователя (для парольного метода входа), просмотра ролей и групп пользователя

Благодаря использованию систем аутентификации основанных на IdP, интерфейс пользователя реализован на стороне владельца IdP.

2.1.4.6 Требования к защите информации от несанкционированного доступа

Так как в системе предполагается использование конфиденциальных данных пользователей, то необходимо обеспечить их хранение согласно законам Российской Федерации. По этому закону хранение персональных данных требует специальной организации дата центров с применением аппаратного шифрования, с постоянным ведением аудита и периодическим тестированием на взлом. Такой уровень защиты данных достигается засечёт того, что система и база данных системы располагаются в среде Сбербанка, а именно в Сбербанк Страхование Жизни. За проведение тестирования на безопасность системы внутри среды Сбербанка отвечает компания ООО «Безопасная информационная зона» также известная как «Бизон».

2.1.4.7 Требования по сохранности информации при авариях

База данных системы должна поддерживать технологию резервного копирования. Данные должны восстанавливаться в полном объёме, с указанием времени в котором эти данные актуальны.

Процедура резервного копирования должна происходить не менее 2 раз в сутки.

2.1.5 Требования к функциям, выполняемым системой

Система централизованной авторизации должна обеспечивать функциональность:

1. Проведение аутентификации зарегистрированных пользователей используя:
   1. Систему Сбербанк Онлайн, на основе протокола аутентификации OpenID Connect.
   2. Систему Госуслуги, на основе протокола аутентификации OpenID Connect.
   3. Протокол аутентификации в LDAP каталоге
   4. Протокол аутентификации SAML
2. Поддержку технологии аутентификации SSO.

2.1.6 Нефункциональные требования

Система централизованной аутентификации должна обеспечивать нефункциональные требования:

1. Поддержка горизонтального масштабирования
2. Поддерживать нормальное функционировние под нагрузкой в 50 запросов в минуту минимум.

2.1.7 Требования к видам обеспечения

2.1.7.1 Требования к лингвистическому обеспечению системы

Для обеспечения корректной работы системы используемые ОС должны включать в себя минимум английский языковой пакет.

2.1.7.2 Требования к программному обеспечению системы

Для работы системы аутентификации сервер должен обладать развёрнутой виртуальной машиной на основе ОС Ubuntu Server 18.04 с установленным Docker CE.

В качестве балансировщика используется сервер с ОС Ubuntu Server 18.04 и с установленным NGINX.

2.1.7.3 Требования к техническому обеспечению

Для нормального функционирования системы сервер с ней должен обладать минимум:

* 4 Гб оперативной памяти
* 2 ядерный процессор
* Частота центрального процессора не менее 1500 МГц
* Жесткий диск 20 Гб

Для нормального функционирования балансировщика нагрузки сервер должен иметь минимум:

* 4 Гб оперативной памяти
* 1 ядерный процессор
* Частота центрального процессора не менее 1500 МГц
* Жесткий диск 20 Гб

2.2 Пояснительная записка к программному обеспечению

2.2.1 Назначение и область применения

Система централизованной авторизации предназначена для агрегации поддержки различных протоколов аутентификации и предоставления конечным пользователям объединённого интерфейса для аутентификации в сервисах компании ООО «БиАйВи». Кроме того, имея в своём распоряжении поддержку современных протоколов, работающих с использованием IdP, система получает возможность реализации технологии SSO, что сильно повысит удобство пользователей при получении доступа к сервисам ООО «БиАйВи» и положительно скажется на безопасности.

Областью применения системы является аутентификация и авторизация пользователей, являющихся клиентами ООО «БиАйВи».

2.2.2 Технические решения

2.2.2.1 Постановка задачи на разработку программы

Необходимо разработать систему централизованной аутентификации и авторизации. Система должна поддерживать работу с современными протоколами аутентификации и работать с другими системами аутентификации, основанными на этих протоколах.

Разрабатываемый программный продукт должен решать следующие задачи:

* Поддержка работы системы Сбербанк Онлайн, основанной на протоколе OpenID Connect
* Поддержка работы системы Госуслуги, основанной на протоколе OpenID Connect
* Поддержка протокола SAML
* Реализация технологии единого входа SSO
* Поддержка горизонтального масштабирования

2.2.2.2 Описание структуры программной системы

Система централизованной авторизации и аутентификации состоит из:

Сервера аутентификации и авторизации – это система реализующая взаимодействие с различными IdP. Процесс взаимодействия с конкретным IdP определяется протоколом, который он поддерживает.

Базы данных сервера аутентификации – эта БД хранит перечень зарегистрированных в инфраструктуре сервисов компании ООО «БиАйВи» пользователей. Каждому пользователю в зависимости от используемого метода аутентификации соответствует сервер IdP, который уполномочен произвести аутентификации этого пользователя. При необходимости в базе данных могут хранится и другие данные пользователя наподобие должности или даты рождения

Сервера балнсировки – этот сервер распределяет запросы между экземплярами системы аутентификации в зависимости от нагрузки.

2.2.2.3 Описание технологии программирования

Ввиду предполагаемой структурной сложности и объёмности необходимого кода, а также исходя из практики разработки в компании ООО «БиАйВи», был выбран объектно-ориентированный подход. Эта методология основывается на объектной декомпозиции, что позволяет удобно представить физическую и логическую структуру проектируемой системы. Такая декомпозиция предполагает разбиение на элементы. Каждый элемент, в соответствии с критерием декомпозиции, принадлежит к описываемой в виде объектов абстракции предметной области. Такое разбиение создаёт удобную и соответствующую предметной области структуру.

Выбор такого подхода к проектированию системы обусловлен следующими причинами:

1. объектно-ориентированная модель снижает возможность допущения логических ошибок при разработке любых систем, что приводит к меньшим затратам ресурсов на корректировки системы в последствии;
2. объектно-ориентированная модель системы составлена таким образом, что она близка к естественному восприятию предметной области человеком, а также способствует наглядности в представлении разрабатываемого продукта и доступности в освоении материала сторонними лицами при, вынужденной, корректировке системы;
3. объектно-ориентированная модель позволяет упростить процесс сопровождения, благодаря более простой процедуре внесения изменений в программный продукт;
4. объектно-ориентированная модель позволяет уменьшить зависимость различных модулей программы друг от друга, что упрощает разработку с использованием уже готовых решений ООО «БиАйВи»;
5. современные популярные языки программирования в основном предполагают использование объектно-ориентированного подхода;
6. использование объектно-ориентированного подхода позволяет быстрее реагировать на изменение требований.

2.2.2.5 Описание и обоснование выбора метода организации входных и выходных данных

Входными данными являются запросы на аутентификацию от сервисов. Вместе с запросом передаются токен выданный пользователю IdP (если таковой предусмотрен выбранным методом аутентификации).

Выходными данными системы является статус пользователя (аутентифицирован или нет), запрошенные сервисом атрибуты пользователя и токен внутреннего формата для записи в базу данных и использования для SSO.

2.2.2.6 Описание алгоритмов и функционирования программы

Во время работы система аутентификации взаимодействует со следующими объектами:

1. Сервисы – ресурсы к которым обращаются клиенты.
2. Клиенты – устройство или программы используемая пользователем для доступа к сервисам.
3. Пользователь – реальный человек, имеющий учётную запись.
4. IdP – доверенный сервер аутентификации конкретного пользователя.
5. БД – база данных зарегистрированных в сервисах ООО «БиАйВи» пользователей
6. Система аутентификации – разрабатываемая система.

Система аутентификации

БД

Клиент

IdP

1

2

3

4

5

6

7

Пользователь

Рисунок 8. Общая схема работы системы.

Процесс работы системы происходит по следующим этапам:

1. Пользователь, предварительно, проходит аутентификацию при помощи клиента, используя свой IdP. Клиент обращается к сервису, и передаёт сертификат со своими данными и подписью IdP.
2. Сервис запрашивает проверку аутентификации пользователя у системы аутентификации.
3. Система аутентификации проверяет, есть ли в её базе данных такой пользователь, и если есть, то имеет ли он уже тикет для допуска к сервисам.
4. В случае отсутствие тикета, система аутентификации обращается к Idp, который указан в базе данных для этого пользователя, запрашивает проверку аутентификации пользователя.
5. Проведя проверку, IdP отвечает о статусе пользователя.
6. В случае если пользователь аутентифицирован в Idp, в базе данных системы создаётся тикет для него. Затем он используется для осуществления доступа к сервису.
7. Получив тикет с подтверждением аутентификации, сервис предоставляет доступ.

2.2.3 Ожидаемые технико-экономические показатели

Результатом работы должно стать стабильное рабочее серверное приложение, позволяющее проводить аутентификацию пользователей, использующих различные протоколы аутентификации. Приложение должно поддерживать горизонтальную масштабируемость.

Расчёт экономической эффективности проекта приведён в разделе 5 отчёта.

2.3 Описание программы

2.3.1 Общие сведения

2.3.1.1 Необходимое программное обеспечение

Для корректной работы приложения на компьютере должна быть установлена операционная система Linux.

2.3.1.2 Используемые языки программирования

При разработке системы аутентификации использовался объектно-ориентированный язык программирования Java. Средой разработки являлась интегрированная среда разработки IntelliJ IDEA.

Java — объектно-ориентированный язык программирования, разрабатываемый компанией Sun Microsystems с 1991 года и официально выпущенный 23 мая 1995 года. На данный момент является одним из самых популярных языков программирования в корпоративной среде. Предоставляет огромное количество библиотек и удобный для разработки синтаксис.

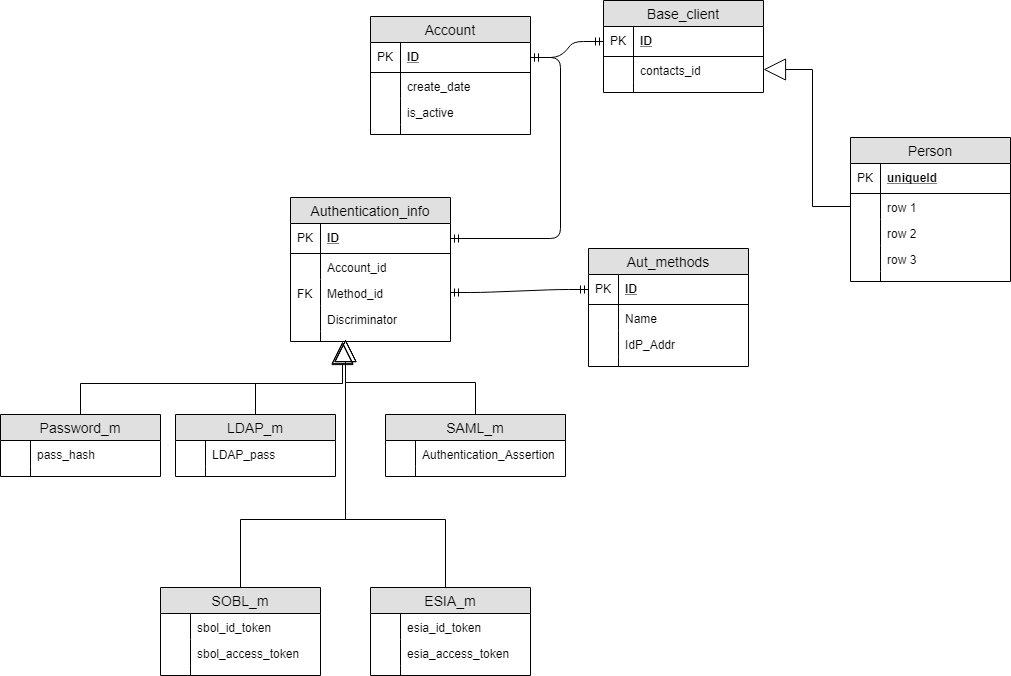
Кроме того, язык Java является приоритетным при разработке в компании ООО «БиАйВи».

2.3.2 Функциональное назначение

Централизованная система аутентификации предназначена для предоставления пользователям возможности получения доступа к сервисам ООО «БиАйВи» с использованием удобных им протоколов аутентификации.

2.3.3 Описание логической структуры

Для хранения аккаунтов используется база данных.



2.3.4 Входные и выходные данные

Система работает на архитектуре REST. REST-архитектура, построенная на базе протокола HTTP, использует методы отправки запросов этого протокола GET, POST, PUT, DELETE, для определения действия, которое необходимо выполнить с входящими параметрами. Соответственно общение с IdP происходит посредствам этих запросов.

Выходными данными системы является статус пользователя (аутентифицирован или нет) и токен внутреннего формата для записи в базу данных и использования для SSO.

2.4 Программа и методика испытаний

2.4.1 Объект испытаний

В рамках данного проекта объектом испытаний является следующие компоненты:

1. Аутентификация пользователей с использованием системы Сбербанк Онлайн.
2. Аутентификация пользователей с использованием системы Госуслуги
3. Аутентификация пользователей с использованием LDAP каталога
4. Аутентификация пользователей с использованием протокола Kerberos
5. Функциональность панели администратора.

2.4.2 Цель испытаний

Целью испытаний является проверка соответствия программного продукта функциональным требованиям и определение корректности работы системы с различными входными данными. Проверка соответствия программного продукта функциональным требованиям. И устранение найденных в ходе тестирования ошибок.

2.4.3 Требования к программе

Основным требованием к системе является корректное выполнение аутентификации для пользователя с использованием сопутствующего ему метода аутентификации. Также необходимо проверить отсутствие задержек при выполнении операций, максимальное приемлемое время выполнения аутентификации 1 сек.

Тестирование будет проводится вручную с использованием специально подготовленных учётных записей пользователей. В случае невозможности создания тестовой учётной записи на стороне клиента ООО «БиАйВи», будет использован локальный тестовый сервер.

|  |  |
| --- | --- |
| Требование | Способ оценки |
| Проверка аутентификации пользователя прошедшего вход в системе Сбербанк Онлайн | 1. Создание соответствующего тестового пользователя. Либо использование одного из пользователей клиента ООО «БиАйВи» 2. Аутентификация в системе Сбербанк Онлайн 3. Произведение пользователем запроса к тестовому сервису. 4. Оценка результата аутентификации пользователя. |
| Проверка аутентификации пользователя прошедшего вход в системе использующей протокол Kerberos | 1. Создание соответствующего тестового пользователя. 2. Аутентификация пользователя в домене, использующем Kerberos 3. Произведение им запроса к тестовому сервису. 4. Оценка результата аутентификации пользователя. |
| Проверка аутентификации пользователя прошедшего вход в системе Госуслуги | 1. Создание соответствующего тестового пользователя. Либо использование одного из пользователей клиента ООО «БиАйВи» 2. Произведение им запроса к тестовому сервису. Оценка результата аутентификации. |
| Проверка аутентификации пользователя прошедшего вход в каталоге LDAP | 1. Создание соответствующего тестового пользователя. 2. Аутентификация пользователя в каталоге LDAP. 3. Произведение им запроса к тестовому сервису. 4. Оценка результата аутентификации. Пользователя |
| Проверка работы журнала пользователей в панели администратора | 1. Войти в систему под учётной записью администратора 2. Открыть раздел журнал пользователей 3. Оценить полученные данные |
| Проверка работы панели администратора при внесении изменений в роли пользователей. | 1. Войти в систему под учётной записью администратора 2. Выбрать тестового пользователя в журнале пользователей 3. Открыть раздел Роли пользователя 4. Внести изменение в роли 5. Оценить результат внесения изменений |

2.4.4 Средства и порядок испытаний

Для проведения испытаний необходима тестовая учётная запись для каждого метода аутентификации. Тестовой учётной записью может быть, как специально созданная для этого запись, так и предоставляемая владельцем тестируемой системы аутентификации.

Также необходим тестовый сервис, его функциональность ограничивается выполнением аутентификации и записи своей работы в лог-файл.

2.4.5 Методы испытаний

|  |  |
| --- | --- |
| Метод проверки | Ожидаемый результат |
| **Проверка аутентификации пользователя прошедшего вход в системе Сбербанк Онлайн** | |
| Создание пользователя с аутентификацией на доверенном IdP Сбербанк Онлайн | Новый тестовый пользователь |
| Запрос тестового сервиса | Предоставление сервисом доступа. Отсутствие форм ввода логина и пароля для пользователя. |
| **Проверка аутентификации не зарегистрированного в Сбербанк Онлайн пользователя** | |
| Формирование учётных данных, для нового не зарегистрированного пользователя | Набор учётных данных не имеющие регистрации |
| Запрос тестового сервиса | Ответ сервиса об невозможности аутентификации из-за отсутствия регистрации пользователя |

3 Эксплуатационная документация на программный продукт

3.1 Руководство администратора

3.1.1 Назначение программного продукта

Система централизованной аутентификации и авторизации предназначена для предоставления удобного метода аутентификации для пользователей сервисов компании ООО «БиАйВи».

3.1.2 Необходимые технические характеристики сервера

Для работы программного продукта необходимо иметь сервер с установленной виртуальной машиной с операционной системой Ubuntu Server 18.04, имеющий на борту не менее 4 ГБ оперативной памяти, с, как минимум, двухъядерным процессором с частотой 1500 Мгц, и 20 Гб жёстким диском.

3.1.3 Установка

Установка приложения предполагает передачу системному администратору копании ООО «БиАйВи» готового Docker контейнера с сконфигурированной внутри системой.

3.1.4 Интерфейс пользователя

Интерфейс пользователя не требует реализации внутри системы, так как предоставляется сервисом, а в случае выбор аутентификации с использованием IdP предоставляется самим IdP.

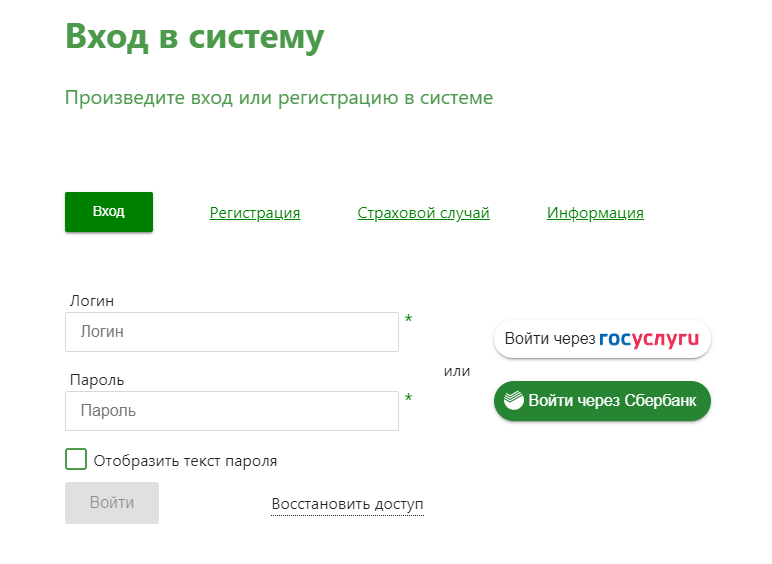


Рисунок 9.Форма входа в сервис ( личный кабинет СБСЖ) с выбором методв аутентификации

При выборе одного из основанных на IdP метода (Госуслуги или Сбербанк Онлайн) пользователь будет перенаправлен на соответствующий IdP, где будет проходить аутентификацию соответствующим способом.

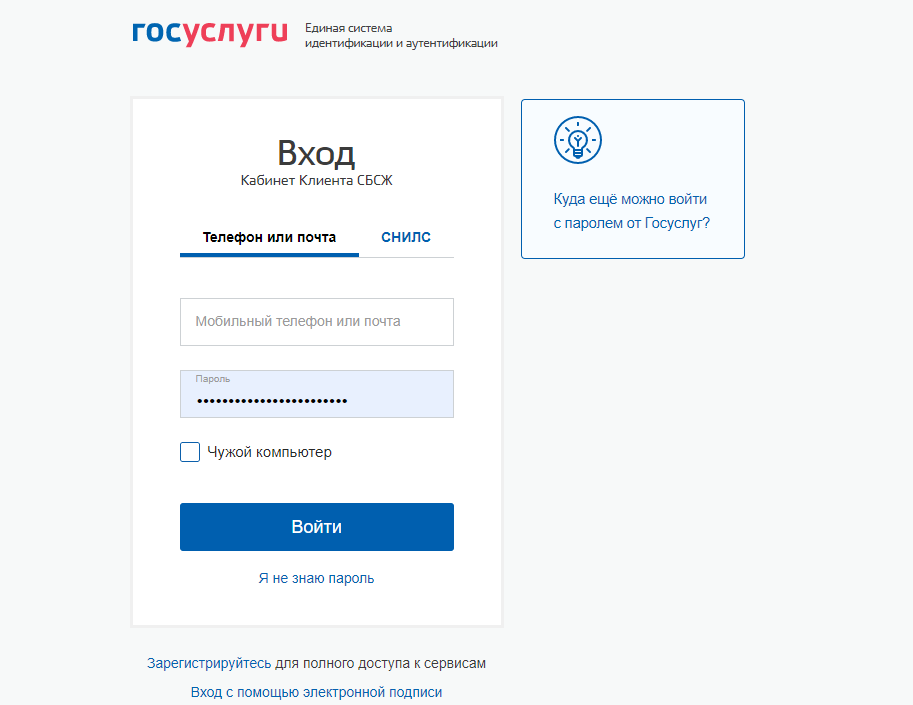


Рисунок 10. Результат выбора аутентификации через Госуслуги

3.1.5 Интерфейс администратора

Панель администратора позволяет просматривать список всех зарегистрированных пользователей.

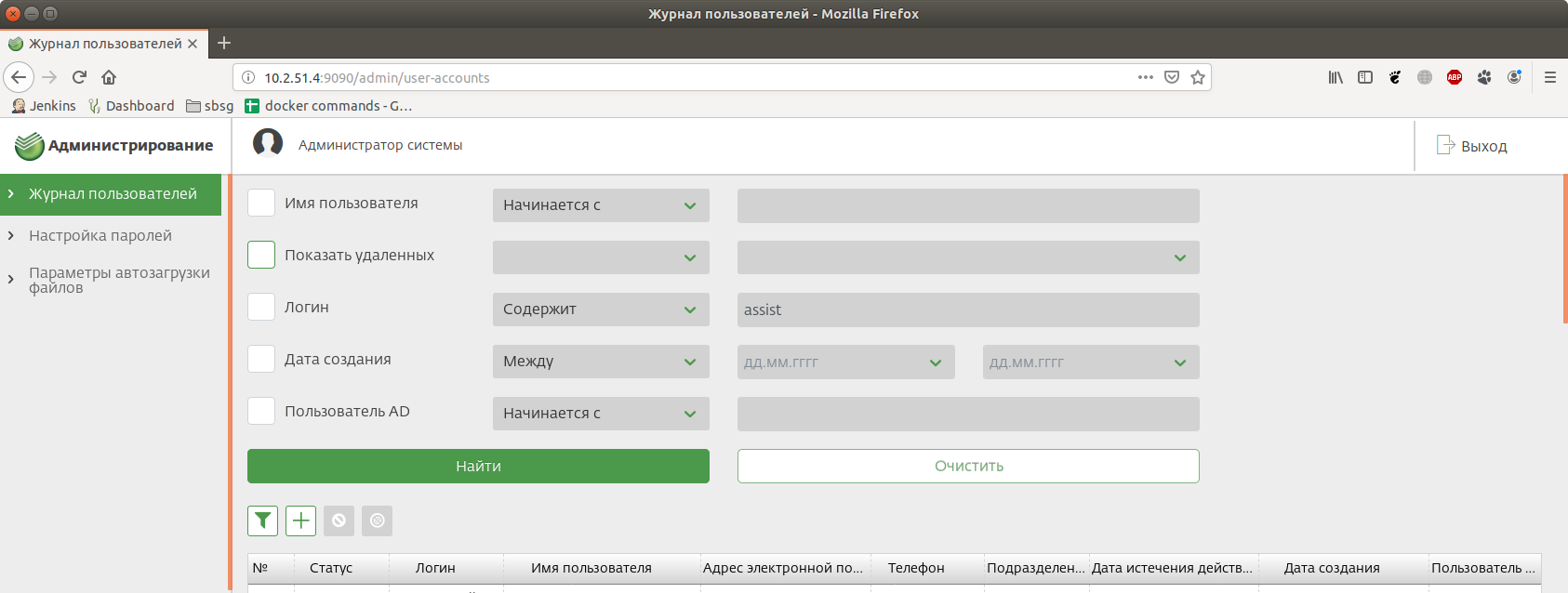


Рисунок 11. Журнал пользователей (информация о пользователях скрыта)

В панели администратора, кроме функционала по просмотру информации о пользователях, предусмотрены некоторые настройки системы.

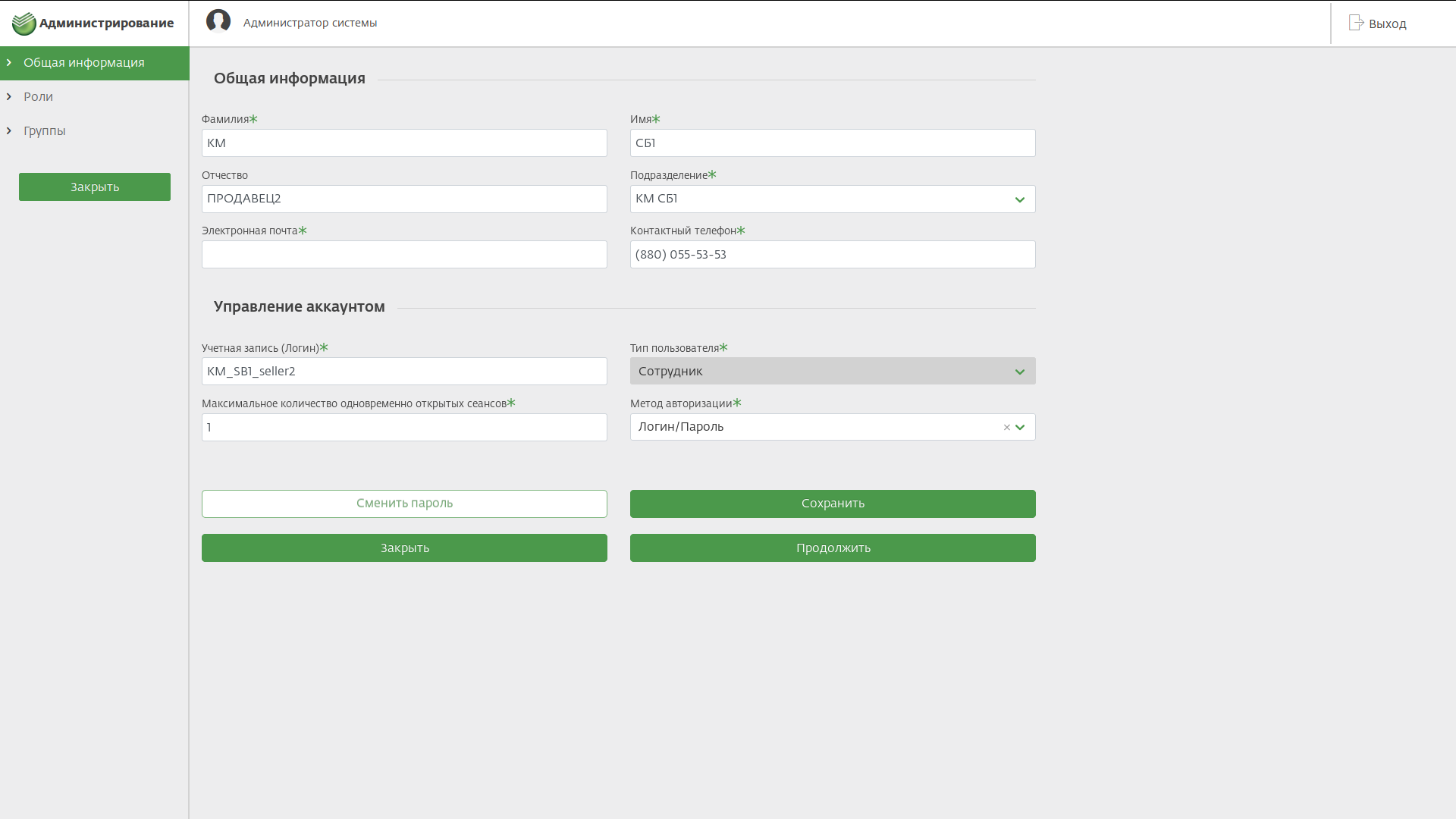


Рисунок 12. Панель администратора - общая информация

В пункте общая информация собран функционал по изменению паромеров аккаунта пользователя во внутренней в базе данных системы. Здесь можно изменить основные поля учётной записи такие как имя, электронная почта, телефон.

E:\Загрузки\diplom-master\diplom-master\scrin\Снимок экрана от 2019-05-27 11-54-50.pngКроме того, в случае необходимости можно изменить метод аутентификации пользователя.

Рисунок 13. Панель администратора - общая информация, выбор метода аутентификации.

4 Акт испытаний программного продукта

5 Экономическое обоснование

5.1 Краткая характеристика программного продукта

Разработка предназначена для предоставления пользователям возможности получения доступа к сервисам ООО «БиАйВи» с использованием удобных им протоколов аутентификации.

Разрабатываемое приложение обладает следующей функциональностью:

* Предоставление общего интерфейса для пользователей, использующих разные протоколы аутентификации
* Поддержка работы с протоколом Kerberos
* Поддержка работы с протоколом OpenID Connect
* Поддержка работы с LDAP аутентификацией
* Реализация технологии Single Sign-On

Данный программный продукт разработан специально для компании ООО «БиАйВи» и может работать только с пользователями сервисов компании. Основными пользователями продукта являются работники предприятий, которым ООО «БиАйВи» предоставляют доступ к своим сервисам.

Основной целью создания программного продукта является предоставление пользователям сервисов компании ООО «БиАйВи» более удобного способа аутентификации в них. Приложение имеет аналоги, но не один из них полностью не удовлетворяет поставленным требованиям, имея либо сильно ограниченный функционал, либо жёстко заданную структуру, не позволяющую легко интегрировать решение в существующую инфраструктуру.

5.2 Расчёт себестоимости разработки программной системы

5.2.1 Материальные затраты

В рамках данной разработки материальные затраты представлены расходами на бумагу и канцелярские товары, оплату доступа в Интернет и т.п.

Разработка велась удаленно (на дому), продолжительность работы составила 6 месяцев силами одного сотрудника. Расходы на канцелярские товары (ручки, тетради, маркеры для доски), использованные при разработке и проектировании, составили 800 рублей. Также была приобретена пачка бумаги формата A4 за 500 рублей. Доступ в интернет предоставляется по тарифу 450 рублей в месяц. Таким образом материальные затраты составили   
500+800+450\*6 = 4000 рублей.

5.2.2 Заработная плата с отчислением на социальные нужды

Основная заработная плата определяется по следующей формуле

(1)

где

– ставка заработной платы (месячная), руб.;

– время разработки программного средства (месяцах).

Отчисления на социальные нужды производятся согласно Федеральному закону № 212-ФЗ «О страховых взносах в Пенсионный фонд Российской Федерации, Фонд социального страхования Российской Федерации, Федеральный фонд обязательного медицинского страхования (редакция от 02.04.2014)», который устанавливает следующие тарифы страховых взносов:

* Пенсионный фонд Российской Федерации − 22 %;
* Фонд социального страхования Российской Федерации − 2,9 %;
* Федеральный фонд обязательного медицинского страхования − 5,1%

Суммарный процент отчислений составляет:

Сумма затрат предприятия на заявленную заработную плату работника вычисляется по формуле (2) и составляет 78000 рублей.

(2)

где

общие расходы предприятия,связанные с заработной платой работника (в рублях);

заработная плата работника (в рублях);

суммарный объём отчислений (в процентах).

Таким образом, расходы на заработную плату составили 78000 рублей.

5.2.3 Амортизационные отчисления и эксплуатационные расходы, связанные с использованием ЭВМ

Амортизационные отчисления определяются как расходы на машинное время. Расходы на машинное время рассчитываются по формуле:

(3)

где

– расходы на машинное время, руб.;

– стоимость 1 часа машинного времени, руб./машино-час;

– время использование ЭВМ для разработки ПС, час.

Стоимость 1 часа машинного времени можно рассчитать по цене ЭВМ, расходам, связанным с её эксплуатацией разработчиком, сроком полезного использования и среднему времени эксплуатации в течение дня с учетом наличия 254 дней в году:

(4)

Где

– покупная цена компьютера, руб.;

– срок службы компьютера, лет;

– время эксплуатации в день, ч.;

– эксплуатационные расходы.

Эксплуатационные расходы − это в первую очередь расходы на электроэнергию, потребляемую компьютером.

Эксплуатационные расходы вычисляются по формуле:

(руб / ч) (5)

где

– стоимость 1кВт·ч электроэнергии (в рублях): 3,6 рублей;

– суммарная потребляемая мощность вычислительной системы (в кВт): 0,75 кВт.

Срок полезного использования ЭВМ определен предприятием и составляет 4 года. По приведённой формуле стоимость 1 часа машинного времени:

(руб / ч). (6)

Поскольку время разработки составляло 6 месяцев, примем время использования ЭВМ (*ВРЭВМ*) за 1056 часов (6 месяцев по 22 рабочих дня разработки). Рассчитаем расходы на машинное время ():

(7)

Из этого следует, что эксплуатационные расходы составили примерно 8690 рублей.

5.2.4 Накладнее расходы

Учитываются расходы на управление, коммунальные услуги, аренду помещения. Величина накладных расходов определяется в процентах от основной заработной платы, составляет 20%, т.е. 12000 рублей:

(8)

5.2.5 Полная себестоимость

Себестоимость разработки программного средства представляет собой сумму затрат по экономическим элементам, приведенным в таблица 8.

Таблица 8 – Расчет полной себестоимости продукта

|  |  |
| --- | --- |
| Затраты | Сумма, руб |
| Материальные затраты | 4 000 |
| Заработная плата и социальные начисления | 78 000 |
| Амортизационные отчисления и эксплуатационные расходы | 8 690 |
| Накладные расходы | 12 000 |
| Итого: | 102 690 |

Следовательно, себестоимость разрабатываемой программы составила 102 690 рублей.

5.3 Оценка экономической эффективности разработки программной системы

Потенциальными пользователями системы являются работники клиентов компании ООО «БиАйВи». На данный момент клиентами являются такие организации как ООО СК «Сбербанк Страхование», АО «НПФ Сбербанка», ООО «Страховая Компания Согласие». Так как разрабатываемая система является частью инфраструктуры сервисов, то финансы на покрытие расходов, связанных с её разработкой, будут браться как доля от общей прибыли с предоставления сервисов.

Выручка компании ООО «БиАйВи» за 2017 год согласно данным Росстата составила 85 млн. рублей. Из этих средств на покрытие расходов на разработку системы заложены 0,5%.

Таким образом по состоянию на 2017 год:

(9)

Таким образом, при заложенных на разработку 425 000 рублей было затрачено 102 690 рублей, что сэкономило 322310 рублей.

(10)

Таким образом затраты на разработку укладываются в заданные рамки и даже приводят к экономии средств.

Внедрение разработки позволит сделать процедуру аутентификации пользователей в сервисах более удобной. А именно сократит количество учётных записей у пользователей и время на ввод учётных данных при входе, засечёт использования SSO аутентификации. В среднем выигрыш по времени для пользователей можно оценить в 20%.

5.4 Вывод

В результате проведенных исследований и расчетов можно говорить об экономической целесообразности разработки программного продукта, так как затраты укладываются в заданные рамки, а функционал разработки позволяет значительно сократить время на аутентификацию для пользователей.

Заключение

Список литературы

1. Хорев П.Б. Методы и средства защиты информации в компьютерных системах: Учеб. пособие для студ. высш. учеб. заведений -- М.: Издательский центр «Академия», 2005. -- 256 с.
2. Адаменко, Михаил Основы классической криптологии. Секреты шифров и кодов / Михаил Адаменко. - Москва: Машиностроение, 2014. - 256 c.
3. Бабаш, А. В. История криптографии. Часть I / А.В. Бабаш, Г.П. Шанкин. - М.: Гелиос АРВ, 2002. - 240 c.
4. Гупта, Арун Java EE 7. Основы / Арун Гупта. - М.: Вильямс, 2014. - 336 c
5. RFC 4120 – Стандарт службы сетевой аутентификации Kerberos (версия 5)
6. RFC 6113 - Обобщенный механизм предварительной аутентификации для Kerberos.
7. Инфраструктуры открытых ключей / Полянская О.Ю., Горбатов В.С. - M.: Интернет-университет информационных технологий - ИНТУИТ.ру, 2007
8. Обзор протокола Kerberos – режим доступа:  
   <https://pro-ldap.ru/tr/zytrax/tech/kerberos.html>

Приложения

Приложение А – примеры программного кода

package com.bivgroup.rest.services;

import com.bivgroup.\*;

import com.bivgroup.config.Config;

import com.bivgroup.rest.dictionaryHibernate.facade.DictionaryBaseFacade;

import com.bivgroup.utils.ParamGetter;

import com.bivgroup.utils.RequestWorker;

import com.fasterxml.jackson.databind.DeserializationFeature;

import com.fasterxml.jackson.databind.ObjectMapper;

import org.apache.http.HttpResponse;

import org.apache.http.NameValuePair;

import org.apache.http.client.entity.UrlEncodedFormEntity;

import org.apache.http.client.methods.HttpGet;

import org.apache.http.client.methods.HttpPost;

import org.apache.http.client.utils.URIBuilder;

import org.apache.http.impl.client.CloseableHttpClient;

import org.apache.http.impl.client.HttpClientBuilder;

import org.apache.http.message.BasicNameValuePair;

import org.apache.http.util.EntityUtils;

import org.apache.log4j.Logger;

import javax.ws.rs.Consumes;

import javax.ws.rs.POST;

import javax.ws.rs.Path;

import javax.ws.rs.Produces;

import javax.ws.rs.core.MediaType;

import java.io.IOException;

import java.net.MalformedURLException;

import java.net.URI;

import java.net.URISyntaxException;

import java.text.SimpleDateFormat;

import java.util.\*;

import java.util.logging.Level;

import static com.bivgroup.SBOLSessionController.SBOl\_BASE\_PROFILE\_ID\_PARAMNAME;

import static com.bivgroup.SBOLSessionController.SBOl\_BASE\_USERLOGIN\_PARAMNAME;

import static com.bivgroup.SBOLSessionController.SBOl\_BASE\_USERPASS\_PARAMNAME;

import com.bivgroup.rest.custom.event.ClientEventHelper;

import static com.bivgroup.rest.custom.event.ClientProfileEventConstants.\*;

import com.bivgroup.rest.custom.event.ClientProfileEventCustom;

import javax.ws.rs.core.Context;

import javax.ws.rs.core.HttpHeaders;

@Path("/rest/pa2SbolGate")

public class PA2SbolGate {

private Logger logger = Logger.getLogger(this.getClass());

private final RequestWorker requestWorker = new RequestWorker();

/\*\*

\* Ссылка на наш сервис, куда производится редирект успешного запроса

\*/

private final static String SBOL\_REDIRECT\_URI = "SBOL\_REDIRECT\_URI";

/\*\*

\* Ссылка на сервис СБОЛ для получения авторизационного ключа

\*/

private final static String SBOL\_AUTH\_URI = "SBOL\_AUTH\_URI";

/\*\*

\* Перечесление запрашиваемой у СБОЛ информации о пользователе

\*/

private final static String SBOL\_SCOPE = "SBOL\_SCOPE";

/\*\*

\* Мнемоника нашей системы (термин система-клиент) выдаваемая СБОЛ

\*/

private final static String SBOL\_CLIENT\_ID = "SBOL\_CLIENT\_ID";

private final static String SBOL\_X\_IBM\_CLIENT\_ID = "SBOL\_X\_IBM\_CLIENT\_ID";

/\*\*

\* Ссылка на сервис СБОЛ для получения токенов

\*/

private final static String SBOL\_TOKEN\_URI = "SBOL\_TOKEN\_URI";

/\*\*

\* Ссылка на сервис СБОЛ для получения инфы о пользователе

\*/

private final static String SBOL\_INFO\_URI = "SBOL\_INFO\_URI";

/\*\*

\* Статический параметр для нашей системы

\*/

private final static String SBOL\_CLIENT\_SECRET = "SBOL\_CLIENT\_SECRET";

// префиксы для имен сущностей (новый хибернейт)

private static final String DCT\_MODULE\_PREFIX\_CRM = "com.bivgroup.crm.";

/\*\*

\* Имя сущности 'Токен акаунта клиента' (таблица SD\_CLIENTPROFILE\_TOKEN)

\*/

private static final String CLIENT\_PROFILE\_EVENT\_ENTITY\_NAME = DCT\_MODULE\_PREFIX\_CRM + "ClientProfileEvent";

private static final String CLIENT\_PROFILE\_TOKEN\_ENTITY\_NAME = DCT\_MODULE\_PREFIX\_CRM + "ClientProfileToken";

private static final String CLIENT\_PROFILE\_ENTITY\_NAME = DCT\_MODULE\_PREFIX\_CRM + "ClientProfile";

private static final String PCLIENT\_VER\_ENTITY\_NAME = DCT\_MODULE\_PREFIX\_CRM + "PClient\_VER";

private final DictionaryBaseFacade dctionaryHibernate = new DictionaryBaseFacade();

private final ClientProfileEventCustom clientProfileEventCustom = new ClientProfileEventCustom();

/\*\*

\* Конфиг параметр-флаг "показывать всегда" кнопку

\*/

private static final String SBOL\_BUTTON\_SHOW\_ALWAYS = "SBOL\_BUTTON\_SHOW\_ALWAYS";

/\*\*

\* Конфиг параметр-флаг "никогда не показывать" кнопку

\*/

private static final String SBOL\_BUTTON\_SHOW\_NEVER = "SBOL\_BUTTON\_SHOW\_NEVER";

/\*\*

\* Конфиг параметр-флпг "показывать по наличию параметра" кнопку

\*/

private static final String SBOL\_BUTTON\_SHOW\_BY\_PARAM = "SBOL\_BUTTON\_SHOW\_BY\_PARAM";

private String SBOL\_ANSWER\_PREPARSER = "";

private String SBOL\_REQUEST = "";

@POST

@Path("/dsPA2SbolGetAuthCodeLink")

@Produces(MediaType.APPLICATION\_JSON)

public Map<String, Object> dsPA2SbolGetAuthCodeLink(Map<String, Object> params) {

logger.debug("dsPA2SbolGetAuthCodeLink start...");

Map<String, Object> result = new HashMap<String, Object>();

try {

Config config = Config.getConfig("pa2sbolws");

String returnURL = getBuildedURLStringByConfig(config);

logger.debug("Final URL: " + returnURL);

if (config.getParam(SBOL\_BUTTON\_SHOW\_ALWAYS,null).equalsIgnoreCase("true"))

return buildLinkResponse(returnURL);

if (config.getParam(SBOL\_BUTTON\_SHOW\_NEVER,null).equalsIgnoreCase("true"))

return buildLogicErrorResponse();

if (config.getParam(SBOL\_BUTTON\_SHOW\_BY\_PARAM,null).equalsIgnoreCase("false"))

return buildLogicErrorResponse();

if (params.containsKey("showButtonSBOL") && params.get("showButtonSBOL").equals("true"))

return buildLinkResponse(returnURL);

else

return buildLogicErrorResponse();

}

catch (Exception e)

{

logger.error("Something gonna wrong in dsPA2SbolGetAuthCodeLink" + e);

return buildExceptionErrorResponse();

}

}

private Map<String,Object> buildExceptionErrorResponse() {

Map<String, Object> result = new HashMap<String, Object>();

result.put("error","2");

result.put("errorText","Внутренняя ошибка.");

return result;

}

private Map<String,Object> buildLogicErrorResponse() {

Map<String, Object> result = new HashMap<String, Object>();

result.put("errorText","Ссылку отображать не надо");

result.put("error","1");

return result;

}

private Map<String,Object> buildLinkResponse(String returnURL) {

Map<String, Object> result = new HashMap<String, Object>();

result.put("error","0");

result.put("url",returnURL);

return result;

}

private String getBuildedURLStringByConfig(Config config) throws URISyntaxException, MalformedURLException {

URI authSbolURI = getConfigURI(config);

URIBuilder authUri = new URIBuilder(authSbolURI);

authUri.addParameter("response\_type", "code");

authUri.addParameter("scope", config.getParam(SBOL\_SCOPE, null)); // все необходимые скопы с инфой о пользователе

authUri.addParameter("client\_type", "PRIVATE");

authUri.addParameter("client\_id", config.getParam(SBOL\_CLIENT\_ID, null));

authUri.addParameter("state", "фыва");

authUri.addParameter("nonce", "фыва");

authUri.addParameter("redirect\_uri", config.getParam(SBOL\_REDIRECT\_URI, null));

return authUri.build().toURL().toString();

}

private URI getConfigURI(Config config) throws URISyntaxException {

try {

return new URI(config.getParam(SBOL\_AUTH\_URI, null));

} catch (URISyntaxException ex) {

java.util.logging.Logger.getLogger(PA2SbolGate.class.getName()).log(Level.SEVERE, null, ex);

throw ex;

}

}

@POST

@Path("/dsPA2SbolWebAuth")

@Produces(MediaType.APPLICATION\_JSON)

@Consumes(MediaType.APPLICATION\_JSON)

public B2BSbolResponse dsPA2SbolWebAuth(B2BSbolRequest requestFromInterface, @Context HttpHeaders headers) {

logger.debug("dsPA2SbolWebAuth start...");

Config config = Config.getConfig("pa2sbolws");

logger.debug("dsPA2SbolWebAuth start. requestFromInterface: " + requestFromInterface.toString());

Map<String, Object> headersMap = processRequestHeaders(headers);

try {

saveEventTryToAuth(headersMap);

CloseableHttpClient client = HttpClientBuilder.create().build();

SBOLSessionController sessionController = new SBOLSessionController(20L);

//получение токенов (access\_token и id\_token)

B2BSbolTokenAnswer answer = this.getTokenAnswer(requestFromInterface, config, client);

//если токен в нашей БД имеется - авторизовать пользователя, если нет - получить данные клиента, сохранить ему токен и авторизовать.

String sub;

if (answer.getId\_token() != null) {

String idToken = answer.getId\_token();

B2BSbolIdToken token = parseDataBase64ToClass(idToken, B2BSbolIdToken.class);

sub = token.getSub();

}

else

{

logger.error("SBOLIntegration event problem service SBOL in get token. headersMap: " + headersMap.toString()

+ " answer: " + answer.toString() + " SBOL\_ANSWER\_PREPARSER: " + SBOL\_ANSWER\_PREPARSER + " SBOL\_REQUEST: "

+ SBOL\_REQUEST);

saveEventErrorServiceSbol(headersMap, SBOL\_ANSWER\_PREPARSER, SBOL\_REQUEST);

return new B2BSbolResponse("4", "Возникла проблема с сервисами СБОЛ, попробуйте позже.");

}

List<Map<String, Object>> clientProfileList = getProfileListByIdToken(sub);

String login;

String clientProfileId;

if ((clientProfileList != null) && (!clientProfileList.isEmpty())) {

//токен нашёлся, сразу авторизуем клиента

Map<String, Object> clientProfileMap = ParamGetter.getMapParamName(clientProfileList.get(0), "clientProfileId\_EN");

login = ParamGetter.getStringParam(clientProfileMap, "login");

clientProfileId = ParamGetter.getStringParam(clientProfileMap, "id");

} else {

//токен не нашёлся, делаем запрос данных пользователя, по ним запрос в нашу бд.

B2BSbolUserResponse userInfoResponse = getUserInfo(config, client, answer);

if (userInfoResponse.getFamily\_name() != null && userInfoResponse.getPassport\_number() != null) {

Map<String, Object> dataForNewSessionAndSavingToken = findUserAndGetHisClientProfileIdAndLogin(userInfoResponse);

if (dataForNewSessionAndSavingToken.isEmpty())

{

logger.debug("SBOLIntegration event block account, headersMap: " + headersMap.toString() + " userInfoResponse: " + userInfoResponse);

saveEventErrorNoAccount(userInfoResponse, headersMap);

return new B2BSbolResponse("3", "Учетная запись не найдена. Для входа через Сбербанк Онлайн вы должны быть зарегистрированы в личном кабинете Сбербанк страхование жизни по адресу https://lk.sberbank-insurance.ru");

}

login = ParamGetter.getStringParam(dataForNewSessionAndSavingToken, "login");

clientProfileId = ParamGetter.getStringParam(dataForNewSessionAndSavingToken, "clientProfileId");

Long stateId = ParamGetter.getLongParam(dataForNewSessionAndSavingToken, "stateId");

if (stateId == 9101L) {

logger.debug("SBOLIntegration event block account, headersMap: " + headersMap.toString() + "clientProfileId " + clientProfileId + "login "+ login);

saveEventErrorAccountBlock(login, clientProfileId, headersMap);

return new B2BSbolResponse("6", "Ваш аккаунт заблокирован");

}

//сохраняем токен пользователю

saveTokenToUser(dataForNewSessionAndSavingToken, sub);

} else {

logger.error("problem service SBOL in get userInfo. headersMap: " + headersMap.toString() + " answer: " + answer.toString()

+ " SBOL\_ANSWER\_PREPARSER: " + SBOL\_ANSWER\_PREPARSER + " SBOL\_REQUEST: " + SBOL\_REQUEST);

saveEventErrorServiceSbol(headersMap, SBOL\_ANSWER\_PREPARSER, userInfoResponse, SBOL\_REQUEST);

return new B2BSbolResponse("5", "Возникла проблема с сервисами СБОЛ, попробуйте позже.");

}

}

saveEventToUser(login, clientProfileId, headersMap);

String session = sessionController.createSession(buildParamsMapForSession(clientProfileId, login, sub));

return buildOkResponseWithSession(session);

} catch (Exception e) {

try {

saveEventErrorInternal(headersMap, SBOL\_ANSWER\_PREPARSER, SBOL\_REQUEST);

} catch (Exception ex) {

logger.error("SBOLIntegration event : something gona wrong in dsPA2SbolWebAuth event: " + ex + " headersMap: " + headersMap.toString()

+ " SBOL\_ANSWER\_PREPARSER: " + SBOL\_ANSWER\_PREPARSER + " SBOL\_REQUEST: " + SBOL\_REQUEST);

return new B2BSbolResponse("2", "Внутренняя ошибка при записи события.");

}

logger.error("SBOLIntegration : something gona wrong in dsPA2SbolWebAuth: " + e + " headersMap: " + headersMap.toString()

+ " SBOL\_ANSWER\_PREPARSER: " + SBOL\_ANSWER\_PREPARSER + " SBOL\_REQUEST: " + SBOL\_REQUEST);

return new B2BSbolResponse("1", "Внутренняя ошибка.");

}

}

/\*\*

\* Метод для декодирование ответов от СБОЛ в виде Base64 и приведение его к классу

\*

\* @param data строка с данными в формате Base64 Mime

\* @param clazz класс, к которому требуется привести данные

\* @param <T> тип класса

\* @return экземпляр класса

\* @throws IOException

\*/

private <T> T parseDataBase64ToClass(String data, Class<T> clazz) throws IOException {

//вынимаем вторую часть токена называемую payload

String payload = data.split("\\.")[1];

//декодируем её и отправляем на десириализацию

String decodedDataString = new String(Base64.getUrlDecoder().decode(payload.getBytes("UTF-8")),"UTF-8");

ObjectMapper mapper = new ObjectMapper();

mapper.configure(DeserializationFeature.FAIL\_ON\_UNKNOWN\_PROPERTIES, false);

return mapper.readValue(decodedDataString, clazz);

}

private void saveEventErrorAccountBlock(String login, String clientProfileId, Map<String, Object> headersMap) throws Exception {

ClientEventHelper clientEvent = new ClientEventHelper();

clientEvent

.from(headersMap)

.setEventType(EVENT\_SYSNAME\_LOGIN\_WRONG\_CREDENTIALS)

.put(EVENT\_AUTH\_CONTRACT\_LOGIN\_FIELDNAME, login)

.put(EVENT\_CLIENTPROFILE\_FIELDNAME, clientProfileId)

.setCustomText("Попытка входа СБОЛ. Аккаунт заблокирован");

Map<String, Object> checkEvent = createEvent(clientEvent);

}

private void saveEventToUser(String login, String clientProfileId, Map<String, Object> headersMap) throws Exception {

ClientEventHelper clientEvent = new ClientEventHelper();

clientEvent

.from(headersMap)

.setEventType(EVENT\_SYSNAME\_SBOL\_LOGIN)

.put(EVENT\_AUTH\_CONTRACT\_LOGIN\_FIELDNAME, login)

.put(EVENT\_CLIENTPROFILE\_FIELDNAME, clientProfileId)

.setCustomText("Вход через СБОЛ");

Map<String, Object> checkEvent = createEvent(clientEvent);

}

private void saveEventTryToAuth(Map<String, Object> headersMap) throws Exception {

ClientEventHelper clientEvent = new ClientEventHelper();

clientEvent

.from(headersMap)

.setEventType(EVENT\_SYSNAME\_SBOL\_LOGIN\_TRY)

.setCustomText("Попытка входа СБОЛ");

Map<String, Object> checkEvent = createEvent(clientEvent);

}

private void saveEventErrorNoAccount(B2BSbolUserResponse userInfoResponse, Map<String, Object> headersMap) throws Exception {

String fullNameUser = userInfoResponse.getFamily\_name()+ " " + userInfoResponse.getGiven\_name()+ " " + userInfoResponse.getMiddle\_name();

ClientEventHelper clientEvent = new ClientEventHelper();

clientEvent

.from(headersMap)

.setEventType(EVENT\_SYSNAME\_SBOL\_LOGIN\_TRY\_USER\_NOT\_FOUND)

.put(EVENT\_SBOL\_SERIES\_NUMBER\_DOC\_USER\_FILEDNAME, userInfoResponse.getPassport\_number())

.put(EVENT\_SBOL\_BIRTHDATE\_USER\_FILEDNAME, userInfoResponse.getBirthdate())

.put(EVENT\_SBOL\_FULLNAME\_USER\_FILEDNAME, fullNameUser)

.setCustomText("Попытка входа СБОЛ, пользователь не зарегестрирован в ЛК");

Map<String, Object> checkEvent = createEvent(clientEvent);

}

private void saveEventErrorServiceSbol(Map<String, Object> headersMap, String answer, String request) throws Exception {

ClientEventHelper clientEvent = new ClientEventHelper();

clientEvent

.from(headersMap)

.setEventType(EVENT\_SYSNAME\_SBOL\_LOGIN\_TRY\_PROBLEM\_SERVICE)

.put(EVENT\_SBOL\_ANSWER\_FILEDNAME, answer)

.put(EVENT\_SBOL\_REQUEST\_FILEDNAME, request)

.setCustomText("Попытка входа СБОЛ, проблема с сервисами СБОЛ");

Map<String, Object> checkEvent = createEvent(clientEvent);

}

private void saveEventErrorServiceSbol(Map<String, Object> headersMap, String answer, B2BSbolUserResponse userInfoResponse, String request) throws Exception {

ClientEventHelper clientEvent = new ClientEventHelper();

clientEvent

.from(headersMap)

.setEventType(EVENT\_SYSNAME\_SBOL\_LOGIN\_TRY\_PROBLEM\_SERVICE)

.put(EVENT\_SBOL\_ANSWER\_FILEDNAME, answer)

.put(EVENT\_SBOL\_RESPONSE\_FILEDNAME, userInfoResponse.toString())

.put(EVENT\_SBOL\_REQUEST\_FILEDNAME, request)

.setCustomText("Попытка входа СБОЛ, проблема с сервисами СБОЛ");

Map<String, Object> checkEvent = createEvent(clientEvent);

}

private void saveEventErrorInternal(Map<String, Object> headersMap, String answer, String request) throws Exception {

ClientEventHelper clientEvent = new ClientEventHelper();

clientEvent

.from(headersMap)

.setEventType(EVENT\_SYSNAME\_SBOL\_LOGIN\_TRY\_INTERNAL\_ERROR)

.put(EVENT\_SBOL\_ANSWER\_FILEDNAME, answer)

.put(EVENT\_SBOL\_REQUEST\_FILEDNAME, request)

.setCustomText("Попытка входа СБОЛ, внутренняя ошибка");

Map<String, Object> checkEvent = createEvent(clientEvent);

}

private void saveTokenToUser(Map<String, Object> dataForNewSessionAndSavingToken, String sub) throws Exception {

Map<String, Object> newTokenMap = new HashMap<>();

newTokenMap.put("clientProfileId", dataForNewSessionAndSavingToken.get("clientProfileId"));

newTokenMap.put("discriminator", "sbol");

newTokenMap.put("idDevice", "web");

newTokenMap.put("hash", sub);

dctionaryHibernate.dctCrudByHierarchy(CLIENT\_PROFILE\_TOKEN\_ENTITY\_NAME, newTokenMap);

}

private Map<String, Object> findUserAndGetHisClientProfileIdAndLogin(B2BSbolUserResponse userResponse) throws Exception {

Map<String, Object> result = new HashMap<>();

try {

Map<String, Object> findClientParams = new HashMap<>();

//дату переводим в другой формат

String formattedBirthDate;

try {

SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");

Date d = sdf.parse(userResponse.getBirthdate());

sdf.applyPattern("dd.MM.yyyy");

formattedBirthDate = sdf.format(d);

} catch (Exception e) {

formattedBirthDate = userResponse.getBirthdate();

}

//findClientParams.put("dateOfBirth$date", formattedBirthDate);

//ФИО оставляем с заглавной первой буквой

findClientParams.put("surname", firstLetterHighOnly(userResponse.getFamily\_name()));

findClientParams.put("patronymic", firstLetterHighOnly(userResponse.getMiddle\_name()));

findClientParams.put("name", firstLetterHighOnly(userResponse.getGiven\_name()));

String seriesAndNumber = userResponse.getPassport\_number();

String series = seriesAndNumber.substring(0, 4);

String number = seriesAndNumber.substring(4, seriesAndNumber.length());

// Map<String, Object> clientDocumentParams = new HashMap<>();

// clientDocumentParams.put("no", series);

// clientDocumentParams.put("series", number);

// List<Map<String, Object>> clientDocumentsParams = new ArrayList<>();

// clientDocumentsParams.add(clientDocumentParams);

// findClientParams.put("documents", clientDocumentsParams);

List<Map<String, Object>> clientListResult = dctionaryHibernate.dctFindByExample(PCLIENT\_VER\_ENTITY\_NAME, findClientParams, true);

Map<String, Object> clientProfileResult = new HashMap<>();

if (clientListResult != null && !clientListResult.isEmpty()) {

for (Map<String, Object> clientMap: clientListResult) {

if (clientMap.get("dateOfBirth$date") != null) {

if (formattedBirthDate.equals(clientMap.get("dateOfBirth$date").toString())) {

//совпала дата. нашли нужного.

if (clientMap.get("documents") != null) {

List<Map<String ,Object>> docList = (List<Map<String ,Object>>) clientMap.get("documents");

for (Map<String, Object> docMap: docList) {

if (docMap.get("series") != null && docMap.get("no") != null) {

String curSer = docMap.get("series").toString().replaceAll(" ","");

String curNo = docMap.get("no").toString().replaceAll(" ","");

if (series.equals(curSer) && number.equals(curNo)) {

//TODO добавить аналогичную проверку документа удостоверяющего личность

Long clientId = ParamGetter.getLongParam(clientMap, "id");

Map<String, Object> findClientProfileParams = new HashMap<>();

findClientProfileParams.put("clientId", clientId);

List<Map<String, Object>> clientProfileListResult = dctionaryHibernate.dctFindByExample(CLIENT\_PROFILE\_ENTITY\_NAME, findClientProfileParams);

if (clientProfileListResult != null && !clientProfileListResult.isEmpty()) {

clientProfileResult = clientProfileListResult.get(0);

}

String login = ParamGetter.getStringParam(clientProfileResult, "login");

String clientProfileId = ParamGetter.getStringParam(clientProfileResult, "id");

Long stateId = ParamGetter.getLongParam(clientProfileResult, "stateId");

result.put("login", login);

result.put("clientProfileId", clientProfileId);

result.put("stateId", stateId);

}

}

}

}

}

}

}

}

return result;

} catch (Exception e) {

logger.error("SBOL findUserAndGetHisClientProfileIdAndLogin: "+e);

throw e;

}

}

private String firstLetterHighOnly(String s) {

StringBuilder sb = new StringBuilder(s.toLowerCase());

sb.setCharAt(0, Character.toUpperCase(sb.charAt(0)));

return sb.toString();

}

private B2BSbolResponse buildOkResponseWithSession(String session) {

B2BSbolResponse response = new B2BSbolResponse("0", "Авторизация прошла успешно");

response.setResult(new B2BSbolResponseResult(session));

return response;

}

private Map<String, Object> buildParamsMapForSession(String clientProfileId, String login, String password) {

Map<String, Object> paramsForSessionCreation = new HashMap<>();

paramsForSessionCreation.put(SBOl\_BASE\_PROFILE\_ID\_PARAMNAME, clientProfileId);

paramsForSessionCreation.put(SBOl\_BASE\_USERLOGIN\_PARAMNAME, login);

paramsForSessionCreation.put(SBOl\_BASE\_USERPASS\_PARAMNAME, password);

return paramsForSessionCreation;

}

private List<Map<String, Object>> getProfileListByIdToken(String id\_token) throws Exception {

//тест получения данных из clientprofile\_token (id=8404793 тестовой базы)

Map<String, Object> clientProfileParams = new HashMap<>();

clientProfileParams.put("hash", id\_token);

clientProfileParams.put("discriminator", "sbol");

return dctionaryHibernate.dctFindByExample(CLIENT\_PROFILE\_TOKEN\_ENTITY\_NAME, clientProfileParams);

}

private B2BSbolUserResponse getUserInfo(Config config, CloseableHttpClient client, B2BSbolTokenAnswer answer) throws IOException {

try {

HttpGet requestUserInfo = new HttpGet(config.getParam(SBOL\_INFO\_URI, null));

requestUserInfo.setHeader("Accept", "application/json");

requestUserInfo.setHeader("Authorization", "Bearer " + answer.getAccess\_token());

requestUserInfo.setHeader("rquid", "a248b91bd93547918a18c29e499d4e9c");

requestUserInfo.setHeader("x-ibm-client-id", config.getParam(SBOL\_CLIENT\_ID, null));

SBOL\_REQUEST = requestUserInfo.toString();

HttpResponse responseUserInfo = client.execute(requestUserInfo);

String userInfoResponseStr = EntityUtils.toString(responseUserInfo.getEntity(), "UTF-8");

logger.debug("SBOL UserInfo Response : " + userInfoResponseStr);

SBOL\_ANSWER\_PREPARSER = userInfoResponseStr;

return parseDataBase64ToClass(userInfoResponseStr, B2BSbolUserResponse.class);

} catch (Exception e) {

logger.error("UserInfo SBOL not working: " + e + " SBOL\_ANSWER\_PREPARSER: " + SBOL\_ANSWER\_PREPARSER + " SBOL\_REQUEST: " + SBOL\_REQUEST);

throw e;

}

}

private B2BSbolTokenAnswer getTokenAnswer(B2BSbolRequest requestFromInterface, Config config, CloseableHttpClient client) throws IOException {

String sbolMarkerAnswerResponse = "";

try {

HttpPost request = new HttpPost(config.getParam(SBOL\_TOKEN\_URI, null));

request.setHeader(HttpHeaders.ACCEPT, "application/json");

request.setHeader(HttpHeaders.CONTENT\_TYPE, "application/x-www-form-urlencoded");

String replaceString = UUID.randomUUID().toString().replace("-", "");

request.setHeader("rquid", replaceString);

request.setHeader("x-ibm-client-id", config.getParam(SBOL\_CLIENT\_ID, null));

ArrayList<NameValuePair> postParameters;

postParameters = new ArrayList<>();

postParameters.add(new BasicNameValuePair("grant\_type", "authorization\_code"));

postParameters.add(new BasicNameValuePair("client\_id", config.getParam(SBOL\_CLIENT\_ID, null)));

postParameters.add(new BasicNameValuePair("client\_secret", config.getParam(SBOL\_CLIENT\_SECRET, null)));

postParameters.add(new BasicNameValuePair("code", requestFromInterface.getCode()));

postParameters.add(new BasicNameValuePair("redirect\_uri", config.getParam(SBOL\_REDIRECT\_URI, null)));

request.setEntity(new UrlEncodedFormEntity(postParameters, "UTF-8"));

logger.debug(request.toString());

SBOL\_REQUEST = request.toString();

//TODO запрос должен быть SSL, в данный момент Bad Request

HttpResponse response = client.execute(request);

sbolMarkerAnswerResponse = EntityUtils.toString(response.getEntity());

SBOL\_ANSWER\_PREPARSER = sbolMarkerAnswerResponse;

logger.debug("SBOL JSON with accessToken and idToken response : " + sbolMarkerAnswerResponse);

return requestWorker.deserializeJSON(sbolMarkerAnswerResponse, B2BSbolTokenAnswer.class);

} catch (Exception e) {

logger.error("Tokens to SBOL not working: " + e + " SBOL\_ANSWER\_PREPARSER: " +

SBOL\_ANSWER\_PREPARSER + " SBOL\_REQUEST: " + SBOL\_REQUEST);

}

return null;

}

protected Map<String, Object> createEvent(ClientEventHelper ev) throws Exception {

ev.reloadPropertiesToParams();

Map<String, Object> res = clientProfileEventCustom.clientProfileEventCreate(ev.getParams());

ev.getParams().clear();

return (res.get(RESULT) != null) ? ((Map<String, Object>) res.get(RESULT)) : null;

}

private Map<String, Object> processRequestHeaders(HttpHeaders headers) {

Map<String, Object> result = new HashMap<>();

/\*

\* Здесь можно писать сценарии обработки заголовков входящих запросов.

\* \*/

// находим цепочку ip

try {

String IPs = headers.getRequestHeaders().getFirst(IP\_ROUTE\_HEADER\_NAME);

if (IPs == null) IPs = DEFAULT\_IP;

Map<String, String> ipMap = new HashMap<>();

ipMap.put(IP\_ADDRESES\_ROUTE\_PARAMNAME, IPs);

result.put(IP\_ADDRESES\_MAP\_PARAMNAME, ipMap);

result.put(IP\_ADDRESES\_SESSION\_PARAMNAME, IPs);

} catch (NullPointerException ex) {

result.put(IP\_ADDRESES\_SESSION\_PARAMNAME, DEFAULT\_IP);

}

return result;

}

}

LDAP – база данных для учётных данных (логины, права, и структура) (иерархическая (дерево)) + стандартные способы доступа.

Kerberos, OIDC… - протоколы (набор соглашений) по передаче данных между участниками(определёнными) сети.

НУЖНО – Сервер(KDC) реализующий(исполняющий) ПРОТОКОЛ, и хранящий данный в LDAP. (KDC выдаёт билеты и всё) (ИНФОРМАЦИЯ О РАСПОЛОЖЕНИИ KDC, алгоритмах шифрования в файле КОНФИГУРАЦИИ)

НУЖНО – Клиентская сторона способная создать нового USER’а и зарегистрироваться на KDC (ИНФОРМАЦИЯ О РАСПОЛОЖЕНИИ KDC, алгоритмах шифрования в файле КОНФИГУРАЦИИ)

НУЖНО – Пристройка(?!) к сервисам, которая может регистрироваться в KDC и принимать сформированные ПРОТОКОЛОМ билеты (ещё есть refresh ticket).

Какой сервер?? Как общаться с сервером?

(OIDC - общается json, Kerberos – не определенно (а как тогда? (GSSAPI ??)))

Критерии сравнения систем авторизации

1. Безопасность: количество участков системы, на которых возможен доступ третьих лиц, который может привести к потере безопасности (нужно описать все участки системы)
2. Удобство для пользователя (наличие SSO?) (кол-во действий для авторизации и необходимость повторной авторизации)
3. Удобство для ресурса (кол-во манипуляций, масштабируемость)
4. Стоимость?
5. Оборудование?
6. Поддержка работы? (выдача учёток)
7. Перспективность технологий (год? Популярность?)

Цель

Цель – оптимизация процессов авторизации и аутентификации в инфраструктуре сервисов компании ООО «БиАйВи»

Описание

Предметной областью является аутентификация/авторизация.

Это неотъемлемая часть всех современных программных продуктов. Необходима и для реализации конфиденциальности, и для безопасности, и для удобства пользователей.

Для успешного прохождения авторизации необходимы:

1. Идентификация – заявление о том, кем является пользователь
2. Аутентификация – проверка подлинности пользователя
3. Авторизация – выдача соответствующих прав доступа к запрашиваемому ресурсу

Одна из основных функций авторизации безопасность. Так как все процессы авторизации происходят удалённо, в рамках компьютерной сети, и защита таких сетей от несанкционированного доступа на сто процентов невозможна. Следовательно, необходимо обеспечить конфиденциальность данных с учётом этих уязвимостей. Первым этапом для решения этой задачи является выделение всех таких небезопасных участков системы.

* Программа, производящая авторизацию со стороны пользователя
* Каналы передачи данных
* Устройство (сервер) и программа производящие авторизацию
* ???

На каждом таком участке системы злоумышленник может производить:

1. Пассивное наблюдение - злоумышленник, перехватывая данные использует их для получения конфиденциальной информации.
2. Воздействие на обменную информацию - злоумышленник, перехватывая и манипулируя информацией, способен получить доступ к конфиденциальным данным/системам.
3. Пароли или иные удостоверяющие данные никогда не пересылаются по сети. Подразумевается, что сетевой трафик может быть прослушан, может произойти подмена сообщения или любая другая пакость.
4. Выдвигается обязательное требование, что информация о паролях/удостоверяющих данных хранится в единственном защищённом месте (Центре распределения ключей Kerberos). Поэтому удостоверяющие данные никогда не сохраняются на том хосте, который пользователь использует для входа/логина. После того, как произошёл первоначальный обмен в рамках аутентификации, этот хост должен забыть сведения о пароле.
5. Хосты/серверы приложений должны быть в состоянии подтвердить свою идентификационную сущность любому, кто запрашивает подобные доказательства.
6. Все коммуникации между аутентифицированными пользователями (клиентами) и сервисами приложений должны иметь возможность быть зашифрованными. С этой целью поддерживаются и могут применяться различные алгоритмы шифрования (все [симметричные](https://pro-ldap.ru/tr/zytrax/tech/encryption.html#symmetric)).

Для

Терминология Kerberos

Как и у всех серьёзных систем, у Kerberos есть своя уникальная терминология, а кое-какие общие термины в контексте Kerberos обретают новый смысл. Постараемся раскрыть их простыми словами (во возможности):

|  |  |
| --- | --- |
| Область действия (Realm) | Realm просто означает совокупность пользователей и серверов приложений, которые охватывает (или имеет о них информацию) Центр распределения ключей (KDC). Так, для того чтобы пользователь подсоединился (или вошёл) в Realm, у Сервера аутентификации (Authentication Server) этой области Realm должны быть сведения об удостоверяющих данных этого пользователя (и другая информация о нём), хранящиеся в защищённой базе данных безопасности того или иного вида (форма хранения не определяется в RFC). В терминологии Microsoft это будет называться Доменом (Domain). Области Realm могут доверять другим Realm, в этом случае доверяющие друг другу области должны быть взаимно аутентифицированными (Cross-Authenticated).  Форма имени Realm — ...@REALM (регистр символов имеет значение). Например, если Realm называется JOE, то его Realm-имя будет @JOE (что отличается от Realm-имени @joe), а если Realm называется EXAMPLE.COM, то его Realm-имя будет @EXAMPLE.COM. (Примечание: Согласно текущей рекомендации ([раздел 6.1 RFC 4120](http://tools.ietf.org/html/rfc4120#section-6.1)) в качестве имени REALM следует использовать доменное имя, которое часто преобразуется в верхний регистр.) Несмотря на то, что последняя форма может напоминать адрес электронной почты, никакого отношения к электронной почте она не имеет. Если буквы большие, это наверняка REALM, а не почта. |
| Принципал (Principal) | Принципал — это строка, полностью идентифицирующая пользователя службы Kerberos. Он имеет форму thing@REALM. Принципал может быть именем сервиса, выполняющегося на хосте (мы будем называть его принципалом сервиса (Service-Principal)), или именем пользователя (мы будем называть его принципалом пользователя (User-Principal)). Принципалы формируют индексное поле для информации об объекте, хранящейся в базе данных безопасности Kerberos (в Центре распределения ключей или KDC). Форматы принципалов для пользователей и сервисов различаются.  Имя принципала пользователя — это приблизительный эквивалент имени пользователя или имени учётной записи. Оно имеет форму principal-name[/instance-name]@REALM (где часть */instance-name* является опциональной). Например, если имя пользователя в принципале пользователя — alice, а Realm — joe, то полный принципал будет alice@joe. Расширение instance-name позволяет любому пользователю иметь более одного принципала. Так, если alice является администратором области Realm joe, имя её принципала будет alice/admin@joe, и у этого принципала будут другие права (и удостоверяющие данные).  Если речь идёт о принципале сервиса, то форма имени принципала становится service-name/QDN@REALM, где QDN — это доменное имя хоста (без точки в конце, как того требует FQDN), на котором работает сервис, а service-name — это специфичная для приложения строка, идентифицирующая сервис на этом хосте. Некоторые типы сервисов используют ключевое слово host. Так, для сервиса ftp, работающего на хосте с именем fileserver.example.com в области Realm @EXAMPLE.COM, имя принципала сервиса будет ftp/fileserver.example.com@EXAMPLE.COM. |
| Разрешение (Ticket) | Разрешение — это структура данных, содержимое которой известно только издателю этого разрешения, и какой-либо стороне или сторонам, к которым это разрешение имеет отношение. Промежуточные хосты, такие как клиентский хост, рассматривают эти разрешения как неразбираемый набор бит и просто передают их на конечный пункт назначения. В Kerberos разрешения могут быть либо Разрешениями на получение разрешения (Ticket Granting Tickets, TGT), — по сути представляют собой доказательства успешно пройденной аутентификации, — либо Сервисными разрешениями (Service Tickets, ST), — выдаются Службой выдачи разрешений (Ticket Granting Service, TGS) и позволяют пользователю получить доступ к требуемому Сервису приложений (Application Service). |

Полезные штуки

1. <https://pro-ldap.ru/tr/zytrax/tech/kerberos.html> - немного теории по Kerberos есть схема (С ссылками. Годно)
2. <https://www.intuit.ru/studies/courses/3580/822/lecture/30592?page=2> – Теория по криптографическим системам.
3. <https://it.wikireading.ru/59914> - про распределение ключей в Kerberos (+ там много всего)
4. <https://web.mit.edu/Kerberos/krb5-1.13/doc/admin/pkinit.html> - PKINIT (англ.)
5. <https://pro-ldap.ru/tr/zytrax/tech/ssl.html> - о x.509 и сертификатах.