

МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Ярославский государственный университет имени П. Г. Демидова»

Кафедра компьютерных сетей

Сдано на кафедру

«_____» _____ 2022 г.

Заведующий кафедрой,
д. ф.-м. н., профессор

_____ С. Д. Глызин

Выпускная квалификационная работа

**Разработка сервиса
централизованного управления паролями**

по направлению
01.03.02 Прикладная математика и информатика

Научный руководитель
к. ф.-м. н., доцент

_____ С. В. Алешин

«_____» _____ 2022 г.

Студент группы ИВТ-41БО

_____ Н. С. Батогов

«_____» _____ 2022 г.

Ярославль, 2022

Реферат

Объем 18 с., 1 гл., 0 рис., 0 табл., 9 источников, 1 прил.

Ключевые слова: **управление паролями, пароль, генерирование пароля, хранение паролей, безопасность компании, кибербезопасность, защита данных, одностраничное приложение.**

Объект исследования - централизованное корпоративное управление паролями.

Цель работы - создание сервиса для удобного, безопасного и централизованного управления паролями в компании.

В результате работы был спроектирован, реализован и протестирован сервис, который позволяет решить данную задачу.

Область применения - небольшая компания, которой нужно удобно организовать пароли для почты, сайтов, серверов, карт и других ресурсов.

Содержание

Введение	4
1. Разработка сервиса	6
1.1. Архитектура сервиса	6
1.2. Реализация серверной части	8
1.2.1. Архитектура серверной части	8
1.2.2. Обзор технологий для написания серверной части	9
1.3. Клиентская часть	12
1.3.1. Архитектура клиентской части	12
1.3.2. Обзор технологий для написания клиентской части	13
1.4. Защита данных	14
Заключение	15
Список литературы	16
Приложение А. Исходный код класса защиты данных на TypeScript	17

Введение

В настоящее время разработка веб-приложений является одной из самых перспективных отраслей современного ИТ. Веб-приложение - это полноценная программа, которую пользователи используют через браузер. Именно поэтому данный вид разработки получил такой большой интерес в последнее время, ведь браузеры есть у каждого. Многие предприниматели переносят свой бизнес в диджитал сферу и выбирают для этого именно веб-приложение. Почти любое веб-приложение можно реализовать в короткие сроки по сравнению с обычным десктопным приложением, а так же любое такое приложение будет кроссплатформенным, ведь оно работает только в браузере клиента. С помощью такого метода программирования можно с легкостью реализовывать даже самые сложные бизнес-процессы и задумки. В рамках данной работы мы будем использовать термин "сервис".

Современный Интернет невозможно представить без паролей, систем шифрования, аутентификации и авторизации пользователей. Одна из самых важных частей каждого человека в сети - это его пароли, но, к сожалению, достаточно много людей не хотят уделять внимание их безопасности. Когда же дело касается безопасности корпоративных паролей, то разговор идет не только о надежной защите данных конкретного пользователя, но и о сохранении имиджа и репутации всей компании. Одновременно с этим, многие компании в своей работе используют незащищенные и неудобные инструменты управления и хранения паролей. Не стоит забывать, что от правильной тактики хранения паролей зависит и уровень защиты от кибератак. Для компании критически важно, чтобы у администратора системы хранения паролей было понимание того, у кого есть тот или иной доступ к информации. Многие используют слишком примитивные способы хранения паролей, например текстовые файлы на рабочем столе компьютера, письма или документы в облаке. Для того, чтобы избежать таких способов и грамотно управлять корпоративными паролями требуется специальный инструмент, заточенный под конкретные нужды компании. Такие инструменты называют менеджерами паролей. В большинстве случаев, компании предпочитают рациональный, надежный и удобный подход к внутреннему приложению собственной команды. Следовательно, наилучшим способом реализации сервиса для централизованного управления паролями можно считать именно веб-приложение. Объектом исследования является централизованное корпоративное управление паролями.

Предметом исследования является сервис для безопасного, централизованного и удобного управления паролями.

Целью выпускной квалификационной работы является создание сервиса для удобного, безопасного и централизованного управления паролями в ком-

пании. Данный сервис позволит небольшим компаниям грамотно работать со своими паролями, а так же не переживать за безопасность их хранения.

Для достижения этой цели необходимо решить следующие задачи:

- произвести аудит существующих сервисов для хранения паролей(менеджеров паролей)
- создать клиентскую часть(интерфейс) сервиса
- создать серверную часть сервиса

???

1. Разработка сервиса

1.1. Архитектура сервиса

Приложение написано в стиле SPA(Single page application). SPA - это одно-страничное приложение, которое работает в браузере и не требует перезагрузки за счет динамического обновления контента с помощью технологии AJAX(Asynchronous Javascript and XML). При использовании AJAX мы получаем много преимуществ, например:

- 1) активное взаимодействие пользователя с веб-страницей(интерактивность)
- 2) плавная работа с приложением
- 3) удобное использование всей страницы
- 4) уменьшение трафика пользователя
- 5) снижение нагрузки на сервер
- 6) положительное влияние на пользовательский опыт

На рис. 1 можно посмотреть отличия классического сайта от приложения в стиле SPA. В нашем сервисе данные передаются через формат JSON.

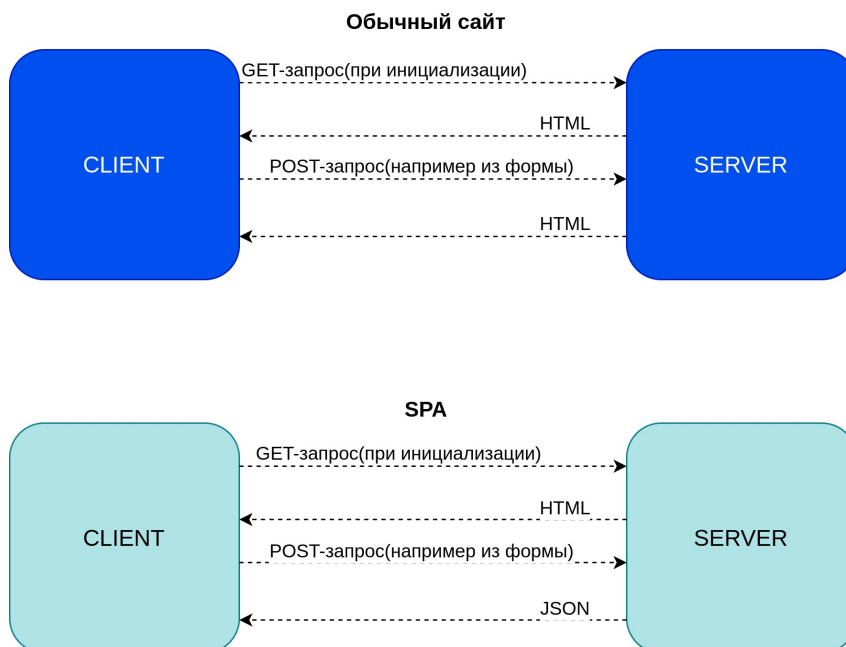


Рис. 1 — Отличия сайта от SPA

Для написания в данном стиле программисту желательно использовать специальную библиотеку для написания клиентского кода. Самые любимые сообществом JavaScript-библиотеки для такой задачи на рынке сейчас представлены на рис. 2.






1		React A declarative, efficient, and flexible JavaScript library for building user inte...	+18.5k★
2		Vue.js A progressive, incrementally-adoptable framework for building UI on the ...	+14.3k★
3		Svelte Cybernetically enhanced web apps	+13.6k★
4		Angular The modern web developer's platform	+9.3k★
5		Solid A declarative, efficient, and flexible JavaScript library for building user inter...	+8.5k★

Рис. 2 — библиотеки JavaScript с наибольшим количеством звезд GitHub

Таким образом, все вышеперечисленные достоинства одностраничного приложения делают его оптимальным выбором для реализации сервиса.

Взаимодействие между клиентом и сервером происходит в классическом REST(Representational state transfer) API стиле через протокол HTTP. REST API - это способ взаимодействия клиента с сервером при котором клиент отправляет запросы на сервер, тот в свою очередь их обрабатывает и посылает ответ. API в данном случае выступает посредником между ними. На рис. 3. представлена схема работы REST API.

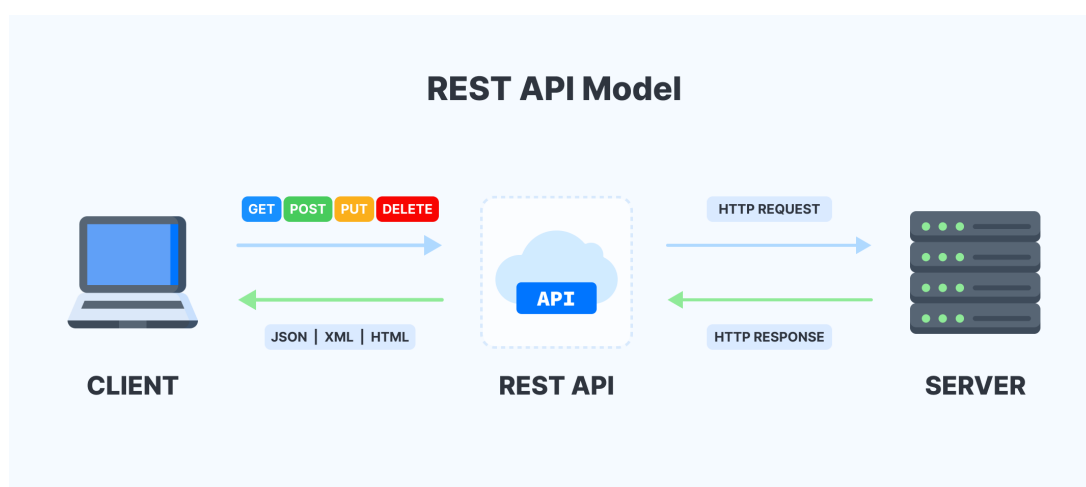


Рис. 3 — схема работы REST API

У HTTP есть свои особенности - ресурсы и HTTP-глаголы. Разобраться с данными терминами лучше всего на примере. Предположим что у нас есть ссылка <http://test.ru/folders/1>. Ее можно разбить на 3 составляющих:

- 1) `http://` - протокол для передачи данных
- 2) `test.ru/` - сервер
- 3) `folders/1` - путь к ресурсу, в данном случае первая папка

Ресурс мы можем ассоциировать с каким-то объектом. Для понимания как именно нам нужно работать с этим объектом нам нужны HTTP-глаголы. HTTP-

глаголы - это элемент протокола HTTP, который используется в каждом запросе, чтобы указать, какое действие нужно выполнить над данным ресурсом.[2] Выделяют 5 основных типов HTTP-глаголов:

- 1) GET - для чтения(получения) ресурса. В случае удачи возвращает код состояния 200(OK) и представление запрашиваемого ресурса. Он является идемпотентным.
- 2) POST - для создания новых ресурсов. В случае успеха возвращает HTTP код 201(CREATED) и в заголовке Location адрес созданного ресурса
- 3) DELETE - для удаления ресурса с конкретным ID. В случае успеха возвращает код 200(OK) вместе с данными удаленного ресурса или код 204(NO CONTENT) без тела.
- 4) PUT - для полного обновления ресурса. При успехе возвращается HTTP 200(OK)
- 5) PATCH - для частичной модификации ресурса, он содержит только изменяемые данные.

Основные плюсы REST:

- надежность
- хорошая масштабируемость
- производительность
- отказоустойчивость
- простота поддержки

Для реализации функционала уведомлений была использована технология SSE(server-sent events). Она позволяет с легкостью передавать уведомления от сервера к клиенту. Клиент подписывается на события, которые будет отправлять сервер и как только происходит событие, клиент моментально получает уведомление. Мы выбрали именно эту технологию, а не подобные ей(WebSocket, Long Pooling и т.д.) потому, что SSE проще, поддерживает автоматическое пере-подключение в случае обрыва соединения, а так же для наших целей не нужно полноценное двунаправленное соединение, ведь данные от клиента к серверу мы передаем по HTTP. Клиент посылает запрос на сервер для открытия соединения, а сервер в ответ посылает заголовок 'Content-Type: text/event-stream' и с этого момента соединение считается открытым.

1.2. Реализация серверной части

1.2.1. Архитектура серверной части

Каждый модуль системы реализует компоненты:

- 1) модель - описывает саму сущность модуля, например “Папка”. Это объектное представление сущности из базы данных
- 2) репозиторий - слой доступа к данным в БД. По сути, это паттерн, который инкапсулирует в себе все, что относится к хранению данных. В нашем приложении он служит для того, чтобы отделить бизнес-логику от конкретной реализации поставщика базы данных, а так же для уменьшения связанности модулей приложения и предотвращения круговых зависимостей. Для того, чтобы избежать подключения всего модуля сущности для простого доступа к данным(например, CRUD - Create, Read, Update, Delete), можно подключить отдельно модуль с репозиторием. В таком случае бизнес логика будет недоступна
- 3) сервис - описывает всю бизнес-логику, которая относится к конкретной модели.
- 4) контроллер - принимает входящие запросы, делегирует их сервису и возвращает ответ. Он выступает в роли промежуточного звена между клиентом и бизнес-логикой.

На рис. 4. можно посмотреть схематичное представление архитектуры серверной части сервиса.

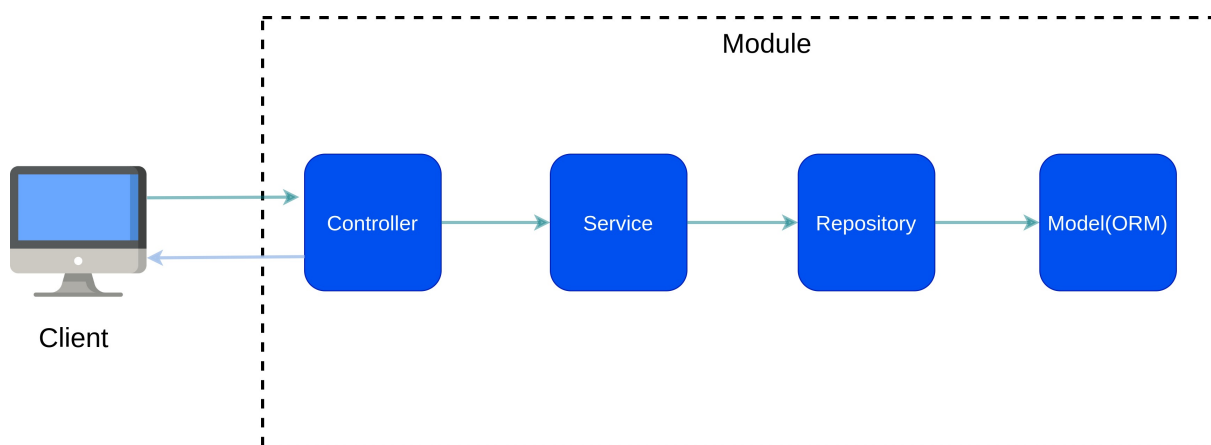


Рис. 4 — Архитектура серверной части

В ходе создания сервиса была спроектирована и создана база данных. Таблицы и их связи представлены ниже на рис. 5.

1.2.2. Обзор технологий для написания серверной части

В качестве языка программирования мы выбрали TypeScript. TypeScript - это компилируемый в JavaScript язык программирования. Он позволяет писать строго типизированный код и следовать лучшим ООП практикам. Особенно хорошо, что в TypeScript есть интерфейсы, это сильно улучшает стиль программирования и предотвращает многие ошибки.

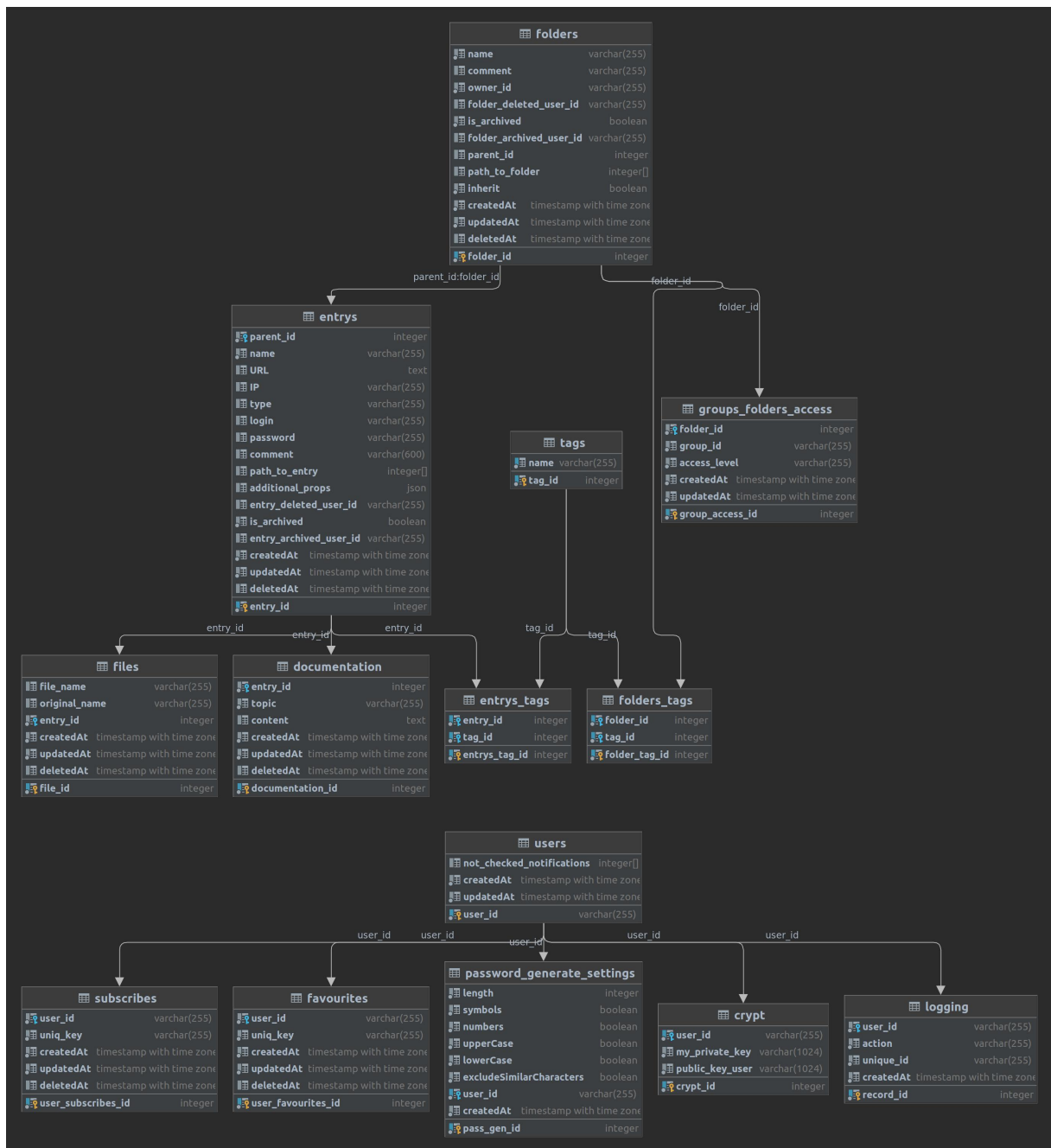


Рис. 5 — Диаграмма базы данных сервиса

Для написания backend-части сервиса использовался Nest.js - это мощный и гибкий фреймворк для написания эффективного серверного кода. Он построен на Express.js - самом популярном решении для создания серверного кода для платформы Node.js.

Node.js - это платформа написанная на C++ с открытым исходным кодом для работы с языком JavaScript. Она построена на движке Chrome V8 и позволяет писать серверный код для веб-приложений. В основе платформы — событийно-управляемая модель с неблокирующими операциями ввода-вывода, что делает ее эффективной и легкой.[3]

Использование фреймворков помогает нам писать меньше шаблонного

кода и сосредоточиться на решении конкретных бизнес-задач. Преимущества Nest.js:

- 1) обеспечивает заранее масштабируемую и чистую архитектуру, требует от программиста писать хороший код
- 2) поддерживает множество инструментов “из коробки”
- 3) присутствуют мощные инструменты командного интерфейса для быстрой разработки модулей
- 4) хорошая реализация инъекции зависимостей(Dependency Injection) из коробки
- 5) поддержка сообщества

Создатели данного фреймворка вдохновлялись Angular и многие вещи написаны с использованием декораторов. Декораторы - это способ “оборачивания” функций, они позволяют модифицировать поведение классов.

Для хранения данных в сервисе использовалась СУБД PostgreSQL. Данная СУБД имеет ряд преимуществ по сравнению с другими СУБД, например:

- можно работать с БД неограниченного размера
- поддерживает индексы, триггеры, функции
- имеет в своем арсенале форматы JSON и ARRAY

Для связи базы данных и объектов в приложении использовалась технология ORM(Object-Relational Mapping - объектно-реляционное отображение) и ее конкретная реализация - Sequelize ORM. Использование ORM аргументировано тем, что такой способ позволяет не зависеть от способа хранения объектов(и самой БД), с легкостью переключаться между SQL и noSQL БД. Так же, ORM сильно повышает эффективность разработки и позволяет оперировать объектами-сущностями в ООП стиле.

Для решения аутентификации, авторизации и контроля доступа использовался продукт Keycloak. Преимущества решения:

- 1) единый вход в приложение(SSO)
- 2) LDAP/Active Directory интеграция и быстрый перенос пользователей
- 3) хороший и документированный API

Документация конечных точек API для серверной части задокументирована с помощью OpenAPI 3.0 Swagger. Данная технология позволяет не только интерактивно просматривать все эндпоинты приложения, но и отправлять запросы прямо из интерфейса Swagger. Вся документация к конечным точкам приложения находится по пути <http://host:3000/api/docs> и представляет собой документацию Swagger, пример на рис. 6.

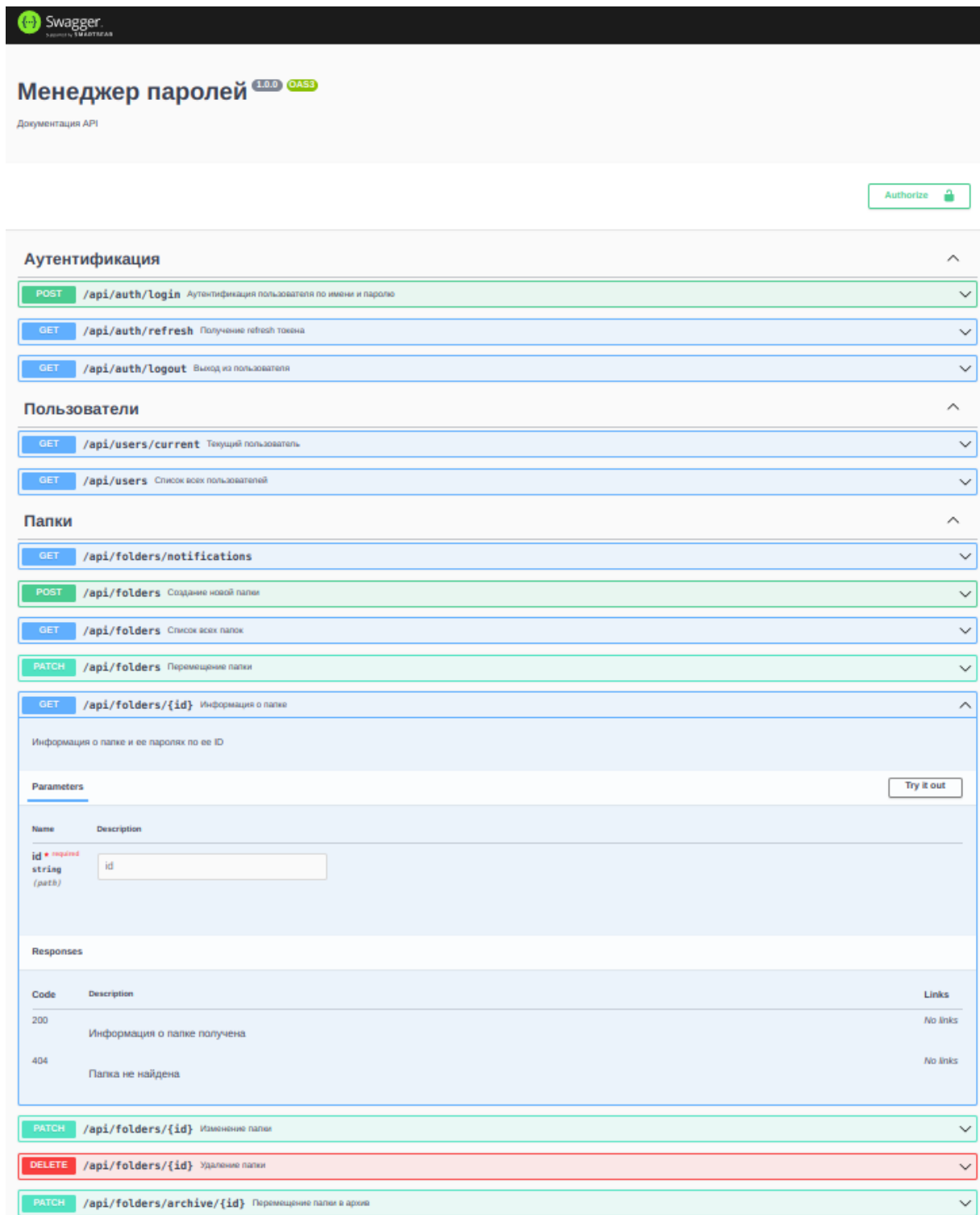


Рис. 6 — Пример документации конечных точек Swagger

1.3. Клиентская часть

1.3.1. Архитектура клиентской части

Клиентская часть написана с использованием методологии атомарного дизайна. Она предполагает, что все приложение разделяется на атомы, молекулы, организмы, шаблоны и целые страницы. ?????????? —TODO(написать про vuch

тут больше и как состояние работает)

1.3.2. Обзор технологий для написания клиентской части

Для клиентской части приложения использовался язык программирования JavaScript и фреймворк Vue.js 2. Для интерфейсов средней сложности это идеальный выбор, потому что данное решение легко масштабировать, у него прекрасная документация, а так же он очень производительный.

В роли менеджера состояния у нас выступает библиотека и одновременно паттерн управления состоянием Vuex 3. Это самое популярное решение для Vue, потому что данные хранятся с правилами, которые гарантируют изменение только предсказуемым образом. По мере роста приложения возникает потребность передавать данные между компонентами и это начинает приносить неудобства в использовании данных. Vuex призван решить эту проблему. В нем данные хранятся централизованно. На рис. 7. показано как происходит работа с данными.

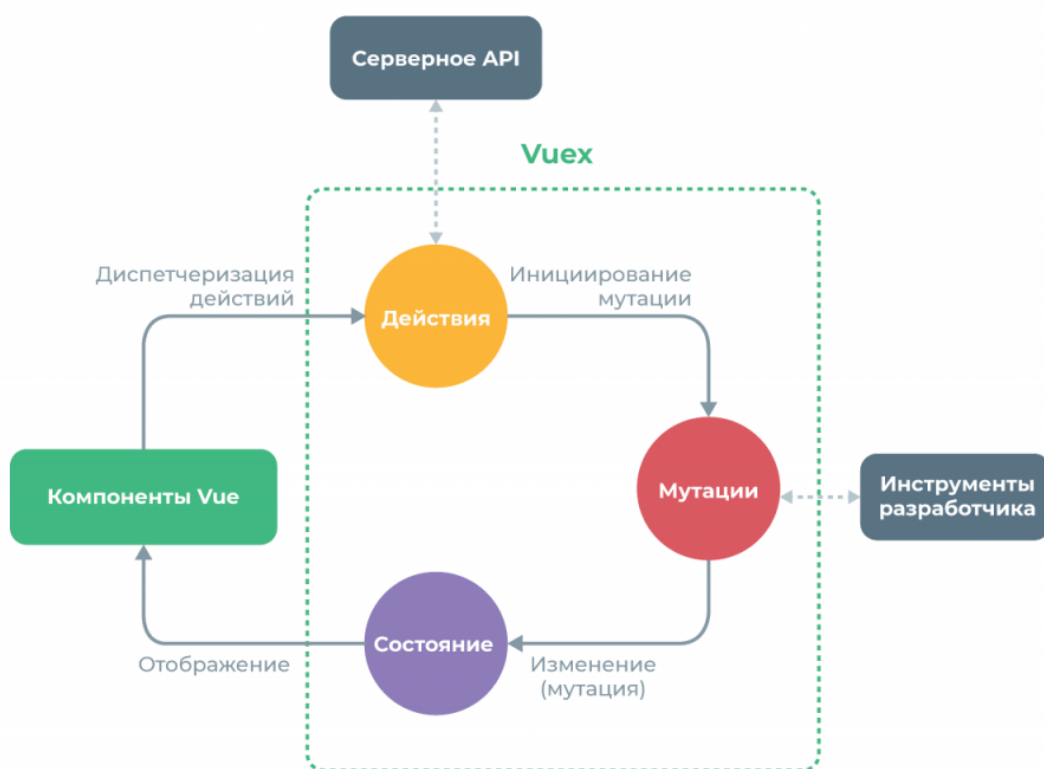


Рис. 7 — работа с данными во Vuex

Управление маршрутами происходит с помощью Vue Router - официального роутера для Vue.js.

В качестве UI библиотеки для написания компонентов интерфейса мы использовали библиотеку Vuetify 2.6. Данное решение хорошо документировано и

позволяет писать масштабируемые компоненты.

1.4. Защита данных

В приложении проводится безопасное сетевое взаимодействие между сервером и клиентом. Это означает, что данные о паролях не передаются в открытом виде по HTTP. В качестве алгоритма шифрования мы взяли RSA. Шаги по шифрованию паролей в сервисе:

- 1) при аутентификации клиент генерирует приватный и публичный 1024-битные ключи
- 2) клиент отправляет серверу публичный ключ
- 3) сервер сохраняет публичный ключ от клиента в базе данных, генерирует пару: свой приватный ключ и свой публичный ключ и отправляет публичный ключ ответом на запрос клиенту
- 4) пароли передаются в виде message, signature , здесь message - сообщение для шифрования, а signature - подпись
- 5) для расшифровки отправленного с клиента пароля, сервер проверяет сообщение от клиента по подписи и сохраненному в БД публичному ключу
- 6) для передачи с сервера на клиент зашифрованного пароля мы шифруем сообщение с помощью публичного ключа, генерируем подпись для зашифрованного сообщения с помощью приватного ключа

Реализация данного алгоритма представлена в приложении А.

Заключение

В заключении подводятся итоги выполненной работы, рассказывается о том, что удалось и что не удалось сделать, описываются перспективы продолжения исследований.

Перспективы исследования. В дальнейшем планируется создание дополнительного функционала, например создание публичных паролей с разовыми токенами для авторизации, чтобы доступ в некоторым паролем имели не только сотрудники компании. Так же планируется улучшить эффективность серверной и клиентской части приложения, для этого будут использоваться специальные инструменты замера метрик скорости работы приложения.

Список литературы

- [1] TeX в ЯрГУ [Электронный ресурс]. URL: <http://www.tex.uniyar.ac.ru> (дата доступа: 20.05.2017).
- [2] REST, что же ты такое? URL: <https://systems.education/what-is-rest> (дата доступа: 14.05.2022).
- [3] Node.js URL: <https://blog.skillfactory.ru/glossary/node-js/> (дата доступа: 14.05.2022).
- [4] Tantau T. PGF — Create PostScript and PDF graphics in TeX [Электронный ресурс]. URL: <https://www.ctan.org/pkg/pgf> (дата доступа: 17.05.2017).
- [5] Кирютенко Ю.А. TikZ & PGF. Создание графики в LaTeX_{2 ϵ} -документах. Ростов-на-Дону, 2014. 277 с. URL: <https://open-edu.sfedu.ru/files/pgf-ru-all-method.pdf>
- [6] Cook S.A. The complexity of theorem-proving procedures // Proceedings of the third annual ACM symposium on Theory of computing. ACM, 1971. P. 151–158.
- [7] Пупырев С. Н., Тихонов А. В. Визуализация динамических графов для анализа сложных сетей // Модел. и анализ информ. систем. 2010. Т. 17, № 1. С. 117–135.
- [8] Кузьмин И. Г. Некоторые проблемы государственных финансов в современной России // Российские предприятия в системе рыночных отношений : материалы научно-практич. конф. Ярославль, 17–18 окт. 2000 г. / отв. ред. Л.Б. Парфенова. Ярославль, 2000. С. 86–90.
- [9] ГОСТ Р 517721-2001. Аппаратура радиоэлектронная бытовая. Входные и выходные параметры и типы соединений. Технические требования. Введ. 2002-01-01. М. : Изд-во стандартов, 2001. IV, 27 с.

Исходный код класса защиты данных на TypeScript

```
1 @Injectable()
2 export class CryptService {
3     constructor(
4         @InjectModel(Crypt)
5         private readonly cryptModel: typeof Crypt,
6     ) {}
7
8     async createUsersKeys(user, userId: string): Promise<keysType> {
9         const keySize = 1024;
10        const keys = sfet.rsa.generateKeysSync(keySize);
11        try {
12            await this.cryptModel.upsert({
13                user_id: userId,
14                my_private_key: keys.private,
15                public_key_user: user.pubKey || '',
16            } as Crypt);
17        } catch (e) {
18            throw new ConflictException();
19        }
20        return keys;
21    }
22
23    async encryptData(currentUser, msg: string): Promise<Record<string, s
24        const user = await this.getCryptUserById(currentUser.id);
25        const publicKey = user.public_key_user;
26        const privateKey = user.my_private_key;
27        try {
28            const newMsg = await sfet.rsa.encrypt(publicKey, msg);
29            const sign = sfet.rsa.sign(privateKey, newMsg);
30
31            return {
32                signature: sign,
33                message: newMsg,
34            };
35        } catch (e) {
```

```

36         throw new ForbiddenException('Error with encrypt');
37     }
38 }
39
40 async decryptData(
41     currentUser,
42     { message: msg, signature: sign },
43 ): Promise<decryptType> {
44     const user = await this.getCryptUserById(currentUser.id);
45     const publicKey = user.public_key_user;
46     const privateKey = user.my_private_key;
47
48     if (!sfet.rsa.verify(publicKey, msg, sign)) {
49         return { status: false, decryptPassword: '' };
50     }
51     const res = sfet.rsa.decrypt(privateKey, msg);
52     return { status: true, decryptPassword: res };
53 }
54
55 async getCryptUserById(user_id: string) {
56     return this.cryptModel.findOne({
57         where: {
58             user_id,
59         },
60         attributes: ['public_key_user', 'my_private_key'],
61     });
62 }
63 }

```