Проектирование микросервиса файлового хранилища

А. В. Нужных, email: nuzhnykh.avpp@gmail.com [1]

[1] Воронежский государственный университет

Аннотация. *В данной работе рассматриваются вопросы проектирования микросервиса облачного хранилища файлов, предоставляющего доступ через API*

Ключевые слова: *Частное файловое хранилище, микросервис, проектирование API*

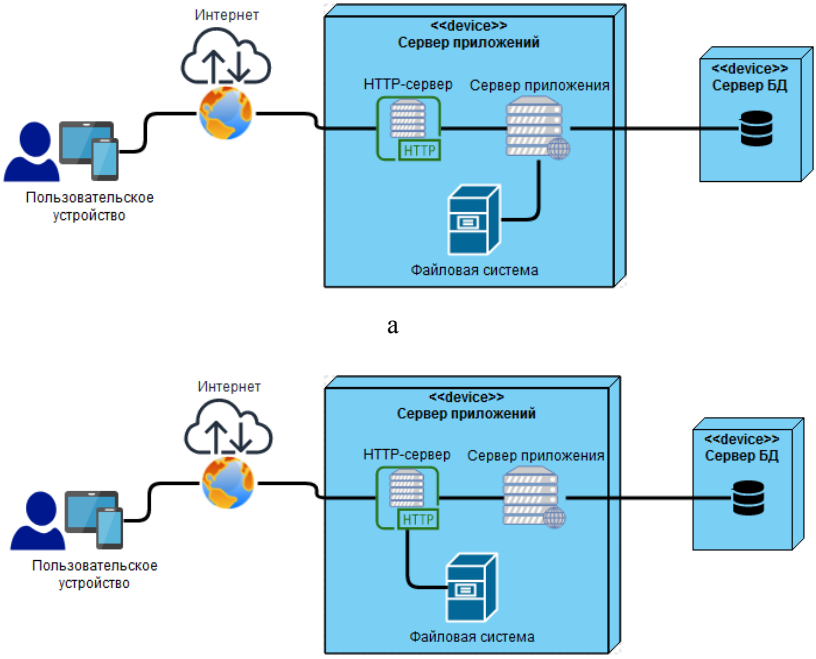
Введение

Одной из основных причин создания Web-приложений является необходимость хранить полезную информацию, называемую контентом. Содержание и внешний вид этой информации в большей степени определяется тематикой и назначением разрабатываемого Web-ресурса. Тем не менее, для организации удобного хранения этой информации, требуется её структурировать, чтобы иметь возможность взаимодействовать с ней. Для этого, в основном, используются различные СУБД. Наиболее распространённый и простой – табличный способ представления информации с применением реляционной модели данных [1]. Также встречаются и другие подходы, в частности, различные виды, так называемых, NoSQL БД [2]. Тем не менее, некоторую информацию удобнее хранить в виде обычных файлов в файловой системе. В основном, это та информация, которую конечные пользователи и так хранят в виде файлов: изображения, аудио/видео файлы, бинарные файлы, документы, архивы и так далее. В связи с этим, при разработке Web-приложений, необходимо предусмотреть возможность хранения файлов и предоставления конечным пользователям доступа к ним.

1. Анализ подходов

Рассмотрим возможные способы реализации такого файлового хранилища. Наиболее простым вариантом является использование файловой системы того сервера, на котором запущено Web-приложение. При использовании трёхуровневой архитектуры [3] в современных реалиях, на физическом сервере запущен http-серевер, принимающий запросы из внешней сети, и несколько серверов различных Web-

приложений. Http-сервер производит перенаправление входящих соединений на соответствующий сервер Web-приложения, который уже и производит фактическую обработку запросов. В случае работы с данными в виде файлов, как на загрузку, так и на отправку, можно организовать работу, как показано на рис. 1а. В этом случае само Web-приложение отвечает за взаимодействие с файлами. При этом стоит учитывать, что всё время работы с файлом, соединение остаётся открытым. А следовательно, канал связи и виртуальный порт приложения будут заняты, что может привести к задержкам у других пользователей, которые в данный момент пользуются другими услугами того же Web-ресурса.



а – Web-приложение обрабатывает запросы к файлам, б – HTTP-сервер обрабатывает запросы к файлам

Рис. 1. Способы обработки запросов на загрузку и скачивание файлов

Можно настроить http-сервер так, чтобы он сам напрямую взаимодействовал с файловой системой (рис. 1б), в случае запроса доступа к файлу [4, 5]. Однако, этот способ недопустим в случаях, когда доступ к файлам должны иметь авторизованные пользователи, за что должно отвечать само Web-приложение или его подсистема аутентификации и авторизации. Кроме того, это не освобождает канал связи и такой способ подразумевает работу с файловой системой самого сервера, что заставляет операционную систему тратить дополнительные вычислительные ресурсы. Конечно, можно смонтировать внешнюю файловую систему, но это повлечёт за собой увеличение трафика, так как при попытке получить доступ к файлу, потребуется сначала скачать его с внешнего сервера.

Ещё одним способом является создание FTP-сервера. В общем-то, это достаточно простое решение с точки зрения развёртывания. Однако этот подход имеет некоторые недостатки, например, в случае, если предоставлять пользователям прямой доступ к нему, с целью уменьшения нагрузки на сервер основного приложения. В частности, могут быть сложности с настройкой авторизованного доступа. Также возможны проблемы, связанные с поддержкой этого протокола браузерами [6].

Несомненно, одним из лучших решений может быть использование уже готовых сервисов. Например, можно интегрироваться с различными облачными хранилищами. Или вообще воспользоваться услугами Firebase [7], где, кроме хранилища файлов, можно найти ещё очень много полезных инструментов. Однако, некоторые файлы нельзя хранить на зарубежных серверах [8]. А использование отечественных ресурсов может быть недопустимым из-за внутренних правил компании.

Таким образом, возникает потребность в разработке своей собственной платформы для хранения файлов в виде отдельного Web-приложения, которую можно легко развернуть на собственных вычислительных мощностях.

2. Проектирование

Функциональность хранения файлов идеально подходит под концепцию микросервисной архитектуры. Проектируемое файловое хранилище будет, как раз, таким микросервисом, который можно будет удобно и быстро интегрировать с другими подсистемами более сложного приложения. На рис. 2 представлена диаграмма развёртывания разрабатываемого микросервиса. Из неё видно, что файловое хранилище может быть организовано на отдельном сервере и конечные пользователи будут общаться с ним напрямую, не занимая ресурсов основного приложения.

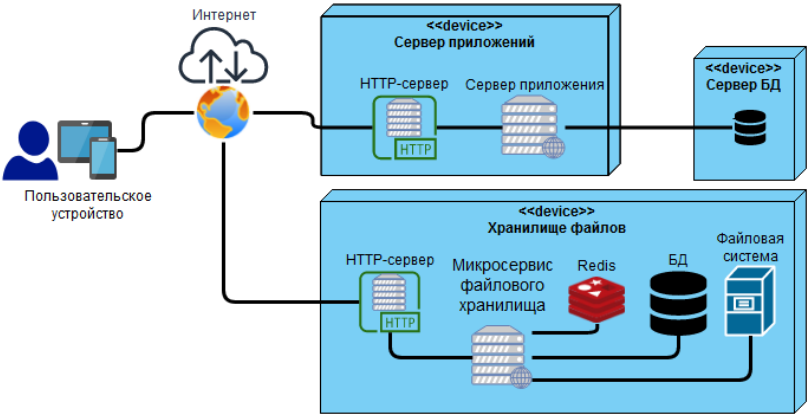


Рис. 2. Диаграмма развёртывания проектируемого микросервиса

Разработка простого HTTP API позволит максимально упростить взаимодействие с сервисом, как со стороны клиентского приложения, так и со стороны других микросервисов всей инфраструктуры. На основе диаграммы вариантов использования (рис. 3), предлагается разработать следующий API:

- Получение доступа к файлу: GET /file/<uid>;
- Загрузка нового файла на сервер: POST /file/<uid>;
- Запрос списка файлов по виртуальному пути: GET /list?path=””;
- Удаление файла(ов): DELETE /file?path=””;
- Запрос одноразовогоURL адреса для загрузки нового файла на сервер с указанием обратных вызовов:
  GET /request/new?success_callback=””&fail_callback=””;
- Запрос одноразового URL адреса для получения доступа к файлу по указанному пути: GET /request/file?path=””

Само хранение файлов будет в любом случае организовано в файловой системе. Так как файловое хранилище разрабатывается в виде микросервиса, то он будет предоставлять свои услуги по хранению файлов разным приложениям. В связи с этим, стоит организовать независимое хранение файлов для каждого потребителя услуги. То есть, для каждого владельца будет использоваться своя собственная директория. Но кроме этого требуется хранить информацию о владельце файла (Web-приложении), логическом пути, используемом в приложении для идентификации файла и, возможно, некоторую другую информацию. Исходя из этого, можно создать БД, схема которой показана на рис. 4. При этом доступ стоит организовать через ORM, за

счёт чего можно отложить принятие решения о фактически используемой БД до момента развёртывания.

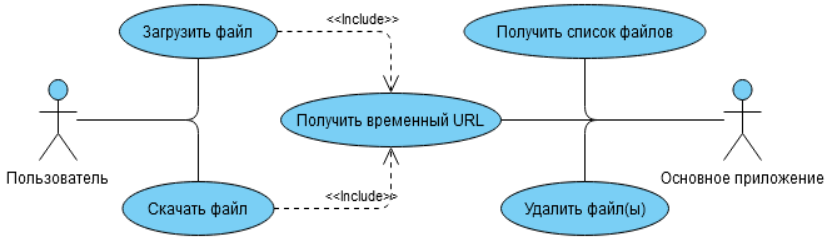


Рис. 3. Диаграмма вариантов использования

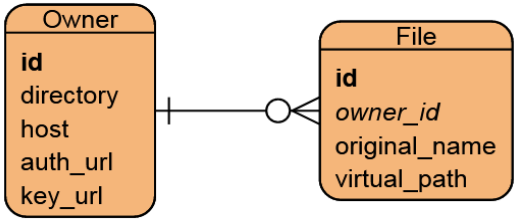


Рис. 4. ER-диаграмма БД

3. Авторизованный доступ

При проектировании рассматриваемого микросервиса, требуется учесть необходимость реализации авторизованного доступа к хранимым файлам. Одним из способов реализации такого поведения, является использование одноразовых уникальных ссылок. Данный механизм похож на распространённый вариант аутентификации по токену, как, например, в OAuth 2 [9]. На рис. 5 показан общий процесс получения доступа к файлу. При этом на стороне авторизующего сервера должна быть реализована обработка запросов от микросервиса файлового хранилища через соответствующее API (URL-адреса запросов хранятся в БД в сущности Owner). Также на стороне микросервиса должно быть развёрнута дополнительная БД для хранения информации о временных ссылок. Для этого замечательно подойдёт резидентная БД, например, Redis [10]. Или можно воспользоваться другим подобным key-value хранилищем.

Наиболее гибким вариантом реализации доступа к хранилищу одноразовых URL, является использование паттерна проектирования адаптер [11]. В этом случае можно описать необходимый программный интерфейс (рис. 6), требуемый для работы, а при фактическом

развёртывании микросервиса, указать желаемую реализацию. Такой подход позволит сделать проектируемое приложение наиболее независимым от фактических реалий его использования, что соответствует концепциям SOLID [12].

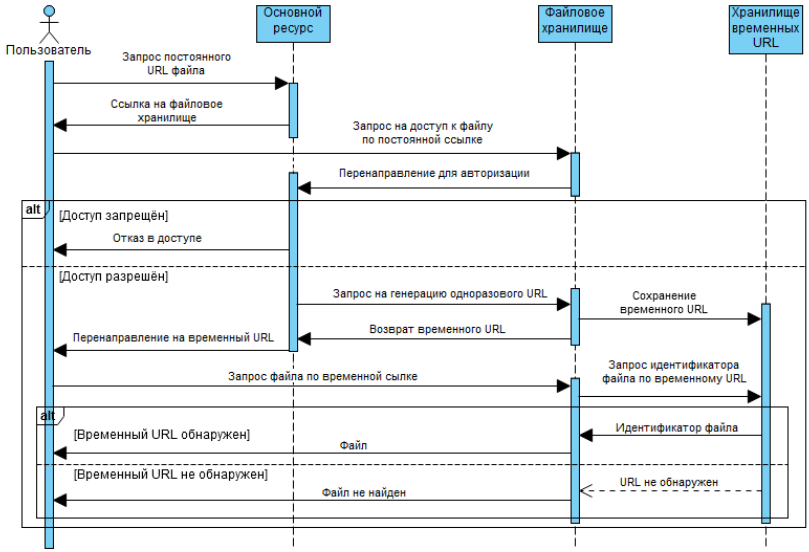


Рис. 5. Диаграмма последовательности для получения доступа к файлу

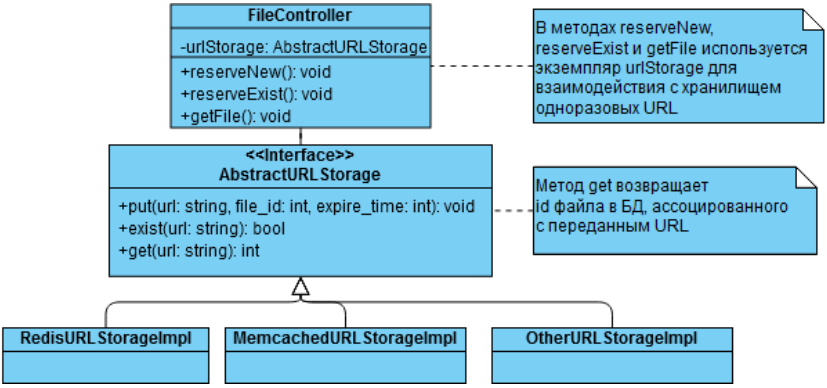


Рис. 6. Паттерн «Адаптер» для доступа к хранилищу временных одноразовых URL.

Так как API микросервиса является публично доступным и в нём предусмотрены команды запроса временных ссылок, которые должны посылаться основным приложением, то микросервису необходимо уметь аутентифицировать источник запроса. С этой целью предлагается использовать инструмент цифровой подписи. Например, можно воспользоваться JWT [13]. Для этого в основном приложении необходимо реализовать возможность предоставления своего публичного ключа. Тогда, микросервис, при обработке запроса на генерацию одноразового URL для доступа к файлу или для загрузки файла, может сделать запрос на получение актуального публичного ключа и удостовериться в легальности обрабатываемого запроса.

Заключение

Таким образом, получилось спроектировать достаточно удобный, гибкий и, в то же время, независимый микросервис файлового хранилища. Его можно развернуть совместно с несколькими основными приложениями. Конечно, описанное приложение можно развивать в нескольких направлениях. В частности, стоит добавить удобный пользовательский интерфейс в виде панели администратора, где можно было бы регистрировать внешние сервисы, просматривать статистику и выполнять какие-либо другие действия. Можно сделать систему меток или хэш-тегов для файлов и возложить на микросервис задачи, связанные с поиском необходимых файлов. Или можно развить его до простого внутрикорпоративного облачного хранилища.

Однако уже на данном этапе оно вполне может справиться с элементарными обязанностями, возложенными на него: хранить файлы и предоставлять к ним доступ авторизованным пользователям. При этом снижается нагрузка на основной сервер, которому не придётся обрабатывать долгие запросы по скачиванию и загрузке файлов.

Список литературы

1. Толстобров, А.П. Управление данными : учебное пособие / А.П. Толстобров. – Воронеж: ИПЦ ВГУ, 2007. – 205с.

2. Фаулер, М. NoSQL: новая методология разработки нереляционных баз данных / М. Фаулер, Прамодкумар Дж. С. – М.: ООО «И.Д. Вильямс», 2013. – 192с.

3. Фаулер, М. Шаблоны корпоративных приложений / М. Фаулер. – М.: ООО «И.Д. Вильямс», 2016. – 544с.

4. Документация nginx [Электронный ресурс]. – Режим доступа : https://docs.nginx.com/nginx/admin-guide/web-server/serving-static-content/

5. Wiki страница модуля Upload для nginx [Электронный ресурс]. – Режим доступа : https://www.nginx.com/resources/wiki/modules/upload/

6. Разработчики Chrome и Firefox хотят отказаться от FTP [Электронный ресурс]. – Режим доступа : https://3dnews.ru/978828

7. Документация облачного хранилища в Firebase [Электронный ресурс]. – Режим доступа : https://firebase.google.com/docs/storage

8. Федеральный закон Российской Федерации № 149-ФЗ от 27.07.2006 (ред. от 30.12.2020) «Об информации, информационных технологиях и о защите информации»

9. Access Tokens [Электронный ресурс] : Guide to building OAuth 2.0 Server. – Режим доступа : https://www.oauth.com/oauth2-servers/access-tokens/

10. Документация [Электронный ресурс]: Официальный сайт Redis. – Режим доступа : https://redis.io/documentation

11. Приёмы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма [и др.]. – СПб.: Питер, 2015. – 368с.

12. Martin, R. C. Clean Architecture: A Craftman's Guid to Software Structure and Design / R. C. Martin. – 1st edition. – New Jersey: Prentice Hall, 2017. – 432p.

13. JSON Web Token [Электронный ресурс]: RFC7519. – Режим доступа : https://tools.ietf.org/html/rfc7519