Требования на разработку сервер-эмулятора спецификации Swordfish на языке Go

Оглавление

Swordfish API Emulator

Swordfish — это стандарт управления хранилищами данных, разработанный SNIA (Storage Networking Industry Association). Эта организация объединяет производителей и пользователей СХД с целью разработки и поддержки отраслевых стандартов и технологий хранения данных. Swordfish предоставляет богатый функционал для управления ресурсами хранилищ данных, конфигурации и мониторинга систем, поддерживает механизмы аутентификации и авторизации для обеспечения безопасности. Спецификация Swordfish призвана решить большое количество проблем с универсальностью управления СХД в современной инфраструктуре, а за развитием и улучшением стандарта стоит активное сообщество.

В случае отсутствия рабочей СХД, которая реализует Swordfish API, для тестирования ПО или просто для изучения Swordfish требуется сервер-эмулятор, который имитирует поведение реальной СХД.

Существует официальный эмулятор Swordfish API от компании SNIA, но он обладает рядом недостатков:

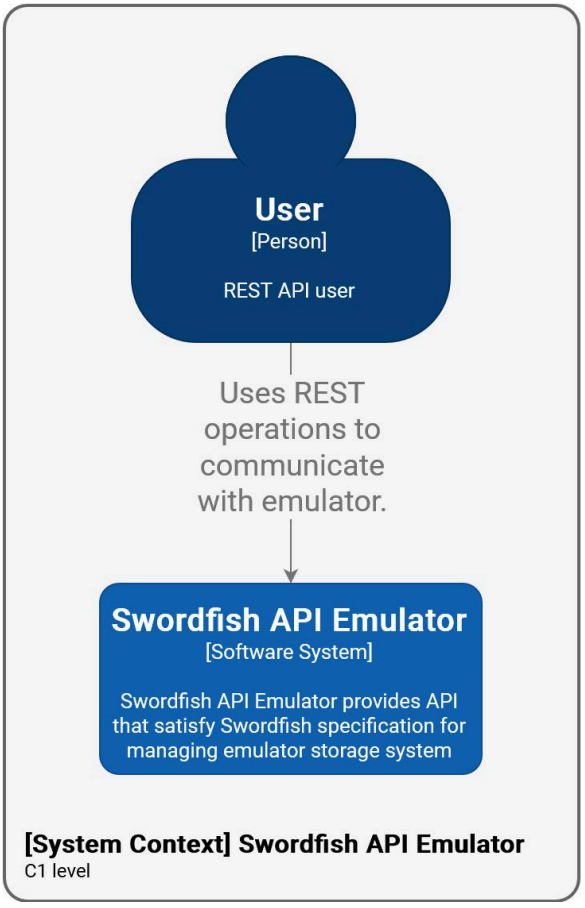
- запутанная иерархия файлов
- неполное покрытие API
- зависимость от Redfish Interface Emulator
- неудобная архитектура кода
- не реализована система ролей
- другие

Исходя из вышеперечисленных проблем, актуальной является задача по разработке сервер-эмулятора свободного от этих недостатков. Для разработки нового сервер-эмулятора был выбран язык Go.

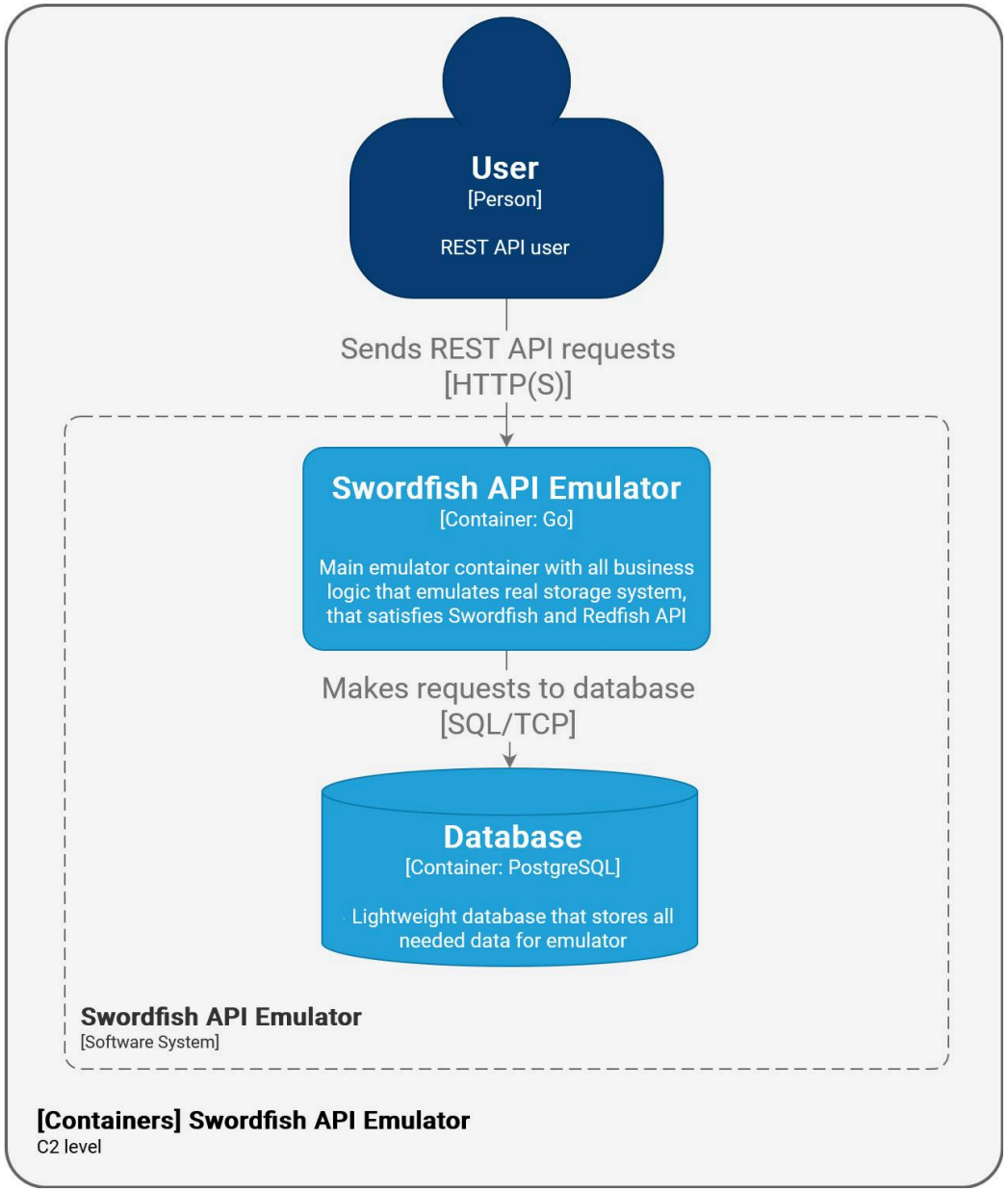
Высокоуровневый дизайн сервер-эмулятора

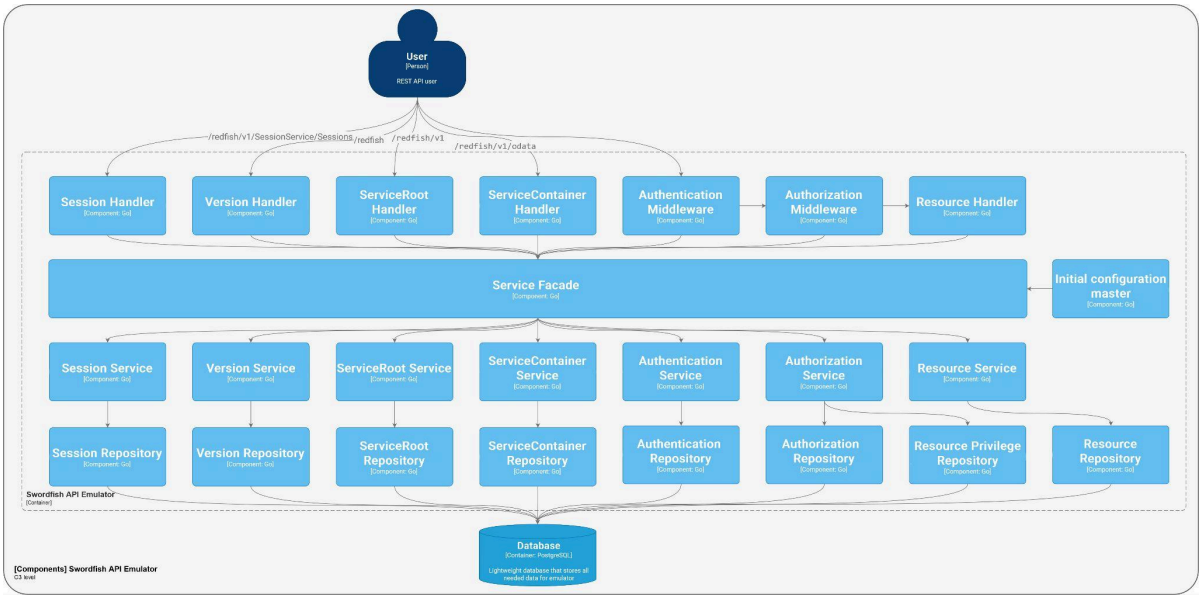
Эмулятор должен быть представлен легковесным приложением, которое будет не трудоемко в поддержке и использовании. Поэтому, дизайн должен быть выбран соответствующим образом. В будущем, при изменении спецификации Swordfish, появиться необходимость добавлять компоненты и дополнять набор ресурсов, поэтому необходимо продумать систему так, чтобы ее было легко расширить.

C4 диаграммы



[System Context] Swordfish API Emulator
C1 level


User
[Person]
REST API user
Sends REST API requests
[HTTP(S)]
Swordfish API Emulator
[Container: Go]
Main emulator container with all business logic that emulates real storage system, that satisfies Swordfish and Redfish API
Makes requests to database
[SQL/TCP]
Database
[Container: PostgreSQL]
Lightweight database that stores all needed data for emulator
Swordfish API Emulator
[Software System]
[Containers] Swordfish API Emulator
C2 level


User
[Person]
REST API user
/redfish/v1/SessionService/Sessions
/redfish
/redfish/v1
/redfish/v1/odata
Session Handler
Version Handler
ServiceRoot Handler
ServiceContainer Handler
Authentication Middleware
Authorization Middleware
Resource Handler
Service Facade
Initial configuration master
Session Service
Version Service
ServiceRoot Service
ServiceContainer Service
Authentication Service
Authorization Service
Resource Service
Session Repository
Version Repository
ServiceRoot Repository
ServiceContainer Repository
Authentication Repository
Authorization Repository
Resource Privilege Repository
Resource Repository
Swordfish API Emulator
Database
[Components] Swordfish API Emulator
C3 level

Swordfish API Emulator C4.drawio.png

В качестве основного архитектурного паттерна выбрана Onion архитектура, которая удовлетворяет основным принципам хорошего дизайна приложений:

- Принципы *SOLID* и в частности ярко выраженная *Inversion of Control*, реализованная посредством *Dependency Injection*, как средства для уменьшения связности модулей
- Масштабируемость
- Заменяемость модулей
- Поддерживаемость (*Maintainability***)**
- Переиспользование и тд.

Стек разработки

- В качестве языка программирования выбран язык **Go**, так как предоставляет удобные и необходимые инструменты для быстрого введения веб приложения в использование, а также поддерживает необходимые принципы и паттерны программирования для написания поддерживаемого кода
- В качестве редактора кода предлагается использовать **VSCode**
- Для запуска и тестирования выбран **Docker** как удобный и популярный вариант контейнеризации
- В качестве **VCS** выбран **Git** (**Gitlab** для облачного хранения) как популярный и удобный вариант

Информация для разработки

Основные пункты по организации кода и процессу разработки:

- Код должен быть структурирован согласно стандартному шаблону
- Код должен проходить проверки *Checkstyle*, настроенные в проекте
- Настройки форматирования для редактора кода или среды разработки должны удовлетворять Editor Config, настроенному в проекте
- Процесс разработки (*Workflow*) должен вестись с использованием функциональных веток (фича-веток) и их вливания в основную (*master*) ветку с помощью механизма *pull-request*'ов
    - Каждая ветка должна быть привязана к *Issue*, без её создания работать над функциональностью нельзя
    - *Issue* может быть закрыта тогда и только тогда, когда привязанный к ней *pull-request* был залит в *master*
    - Каждый *pull-request* должен проходить проверку другим членом команды для выявления проблемных моментов в коде

    - Каждый *pull-request* должен проходить проверку с помощью автоматического тестирования, настроенного в проекте
    - Каждый *pull-request* должен помимо функциональности содержать соответствующие юнит тесты
    - Каждая ветка должна иметь соответствующее название в формате *[dev|bug]/<фамилия>/<номер issue>-короткое-описание-функциональности*
    - Коммиты должны быть написаны в формате, описанном здесь
    - Коммиты должны содержать целостную функциональную единицу (целостное изменение). При наличие связанных коммитов, описывающих данную функциональную единицу рекомендуется сворачивать их в один (делать *git rebase*) для поддержания читаемой истории разработки
- Проект должен иметь покрытие *unit* и интеграционными тестами с использованием средств *Go* и механизма контейнеризации *Docker*
    - Покрытие кода должно быть минимум 80%

Функциональные требования

Детальное описание требований на API, который должен быть реализован СХД, представлено в спецификации Swordfish API и Redfish API. Задача сервер-эмулятора — предоставить полное покрытие этих требований, но без детальной функциональной реализации поведения. То есть сервер-эмулятор должен поддерживать все API операции, описанные в спецификации, но не должен реализовывать бизнес-логику СХД.

1. Общие требования на эмулятор

1.1. Эмулятор должен иметь дистрибутив для распространения

1.1.1. Дистрибутив должен быть в формате *zip*

1.1.2. Дистрибутив должен включать в себя:

1.1.2.1. скрипты установки сервер-эмулятора *(scripts/install.sh, scripts/install.bat)*

1.1.2.2. скрипты запуска сервер-эмулятора *(scripts/launch.sh, scripts/launch_emulator_server.bat)*

1.1.2.3. скрипты запуска БД *(scripts/launch_database_server.sh, scripts/launch_database_server.bat)*

1.1.2.4. скрипты установки наборов данных *(scripts/install_datasets.sh, scripts/install_datasets.bat)*

1.1.2.5. скрипты деинсталляции установленного приложения *(scripts/uninstall.sh, scripts/uninstall.bat)*

1.1.2.6. директории с наборами данных для установки *(datasets/)*
1.1.2.7. директорию с системными логам поведения системы *(/logs/errors.log, /logs/info.log)*
1.1.2.8. файлы конфигурации *(configs/)*
1.1.2.9. документацию *(docs/)*
1.1.2.10. *README* с инструкцией по использованию

1.2. Эмулятор должен иметь *docker-image* как для тестирования, так и для возможности запуска пользователем
1.2.1. Файлы для сборки *docker-image* должны лежать в репозитории в папке */deployments*
1.2.2. *Docker-image* должен быть опубликован на *docker-hub*
1.3. Эмулятор должен иметь *Makefile* для основных процессов во время разработки (запуск тестов, генерация моков и тд.)

2. Модуль управления ресурсами

2.1. Эмулятор должен поддерживать базовые ресурсы
2.1.1. Эмулятор должен реализовывать один из базовых ресурсов: *Storage* или *StorageService*, так как все схемы основаны на одном из них
2.1.2. Эмулятор должен гарантировать, что каждый класс (*ClassOfService*) включает в себя хотя бы один поддерживаемый сервис (не должно быть класса сервиса без единого сервиса - экземпляра данного класса
2.1.3. Эмулятор должен удовлетворять ограничениям на *StorageSystem*, которые требует минимального количества управляемых систем (не может быть сервера без управляемых систем)
2.1.4. Данная поддержка фактически реализуется путем инициализации базового датасета, в котором продумано наличие базовых статических ресурсов

2.2. Система должна удовлетворять основным ограничениям, описанным в спецификации
2.2.1. Должна быть поддержка функциональности *Redfish* версии 1.18.0 (согласно спецификации Swordfish версии 1.2.6)
2.2.2. Должна быть создана система проверки состояний ресурсов (ресурс должен включать в себя объект *Status* с полями *Health* - собственное состояние ресурса, и *HealthRollup* - общее состояние всех под-ресурсов ресурсов вместе с текущим ресурсом). Внутри эмулятора нет динамического изменения состояния ресурсов, поэтому данные поля могут меняться только от действий пользователя

2.3. Эмулятор должен возвращать правильные *HTTP* коды на соответствующие операции с ресурсами согласно спецификации

2.3.1. Системой должны быть определены соответствия *HTTP* кодов и ошибок, связанных с доступом/изменением/удалением ресурсов. При этом должна быть возможность корректировать это соответствие при изменении спецификации без изменения основной логики эмулятора (должна быть реализована внутренняя “библиотека” ошибок)

2.4. Эмулятор должен поддерживать *Actions*, являющимися расширением операций над ресурсами, которые не реализовать через *REST* запросы

2.5. Эмулятор должен поддерживать типы описанные в спецификации, так как они являются непосредственным *JSON* представлением ресурсов

2.6. Эмулятор должен удовлетворять требованиям при передаче “чувствительных” данных:

2.6.1. Системой должен использоваться только *HTTPS* протокол

2.6.2. Данные должны быть заменены на *null* при передаче в ответе

2.6.3. Может использоваться код *404 Not Found*, вместо *401 Unauthorized* или *403 Forbidden*

2.7. Управление ресурсами должно реализовываться непосредственно логикой “сервисов” (*Service*) и их операциями над “репозиториями” (*Repository*), которые в свою очередь отправляют запросы к базе данных

3. Модуль хранения ресурсов

3.1. Все ресурсы Swordfish и необходимые эмулятору данные для авторизации, аутентификации, системы ролей и тд должны храниться в базе данных PostgreSQL

3.1.1. Пользователю должен быть предоставлен выбор запускать встроенную (*embedded*) *PostgreSQL* или поднятую локально (в инфраструктуре пользователя или контейнере *Docker*)

3.1.2. Для хранения *JSON* данных в *PostgreSQL* должен использоваться эффективный *JSON(b)* формат хранения

3.2. Эмулятор должен реализовывать хранение инициализированных ресурсов

3.2.1. Данные ресурсов должны храниться на диске (базой данных) между сессиями работы с эмулятором

3.2.2. Эмулятор должен сохранять состояние системы (набор ресурсов и их данные), полученное пользователем в результате работы с эмулятором. То есть, все действия

совершенные пользователем над ресурсами не должны сбрасываться между сессиями работы с эмулятором, без его требования

3.3. Эмулятор должен предоставлять несколько вариантов базовых датасетов для инициализации системы

3.3.1. Датасеты должны храниться на диске в подкаталогах *datasets/*

3.3.2. Датасеты должны из себя представлять набор ресурсов, разделенных на подкаталоги, в *JSON* формате

3.3.3. Эмулятор должен производить загрузку определенного пользователем датасета в базу данных перед началом использования эмулятора

3.4. Эмулятор должен хранить данные для аутентификации

3.4.1. Эмулятор должен предоставлять базовые (неизменяемые) данные для аутентификации. То есть, эмулятор должен перед началом работы загружать определенные заранее логины и пароли в базу данных

3.4.2. Эмулятор должен хранить пользовательские данные для аутентификации между сессиями работы. То есть, произведенные пользователем действия над *AccountService* не должны сбрасываться между сессиями работы с эмулятором

3.5. Эмулятор должен хранить данные для авторизации

3.5.1. В базе данных должно храниться соответствие пользователя (логин) и его ролей

3.5.2. Эмулятор должен хранить необходимые роли для доступа к сконфигурированным ресурсам

3.5.2.1. Эмулятор должен хранить роли между сессиями работы

3.5.3. Эмулятор должен иметь “регистр привилегий” для всех сконфигурированных ресурсов в требуемом пунктом 4.4 формате

3.5.3.1. Эмулятор должен хранить “регистр привилегий” между сессиями работы

3.5.3.2. В базе данных “регистр привилегий” должен быть представлен в JSON(b) формате

4. Модуль конфигурации

4.1. Все файлы конфигурации должны храниться в директории *configs/*

4.2. Система должна поддерживать базовую конфигурацию (*emulator* в *configs/emulator-config.yaml)*

4.2.1. Эмулятор из конфигурации должен получать настройки порта (поле *port*) для запуска

4.2.2. Эмулятор из конфигурации должен получать настройки для HTTP сервера Go: таймауты ожидания, обработки… (поля *write-timeout, read-timeout*)

4.3. Эмулятор должен получать настройки логирования из конфигурационного файла (*logger* в *configs/emulator-config.yaml)*

4.3.1. Эмулятор должен получать названия файлов (в каталоге *logs/)*, в которые сохранять логи разного уровня (поля *info*, *error*)

4.4. Эмулятор должен получать настройки функционала хранения ресурсов (*db* в *configs/emulator-config.yaml)*

4.4.1. Эмулятор должен получать настройки базы данных из файла конфигурации (поля *host, name, user, password)*

4.5. Эмулятор должен получать настройки функционала аутентификации (*authentication* в *configs/emulator-config.yaml)*

4.5.1. Эмулятор должен получать настройки режима аутентификации (поле *type)*

4.5.2. Эмулятор должен получать настройки включения/выключения аутентификации (поле *enable)*

4.6. Эмулятор должен получать настройки функционала авторизации (*authorization* в *configs/emulator-config.yaml)*

4.6.1. Эмулятор должен получать настройки включения/выключения авторизации (поле *enable)*

4.7. Эмулятор должен получать настройки базовой системы для инициализации (*dataset* в *configs/emulator-config.yaml)*

4.7.1. Эмулятор должен получать путь до каталога с набором базовых системы для инициализации из файла конфигурации (поле *path*)

4.7.2. Эмулятор должен получать название базовой системы (из соответствующего каталога) для инициализации из файла конфигурации (поле *name*)

4.8. Эмулятор должен предоставлять пользователю возможность создать собственный *dataset*

4.8.1. Эмулятор должен иметь строго определенный формат для описания базовой системы (*dataset*) для инициализации

4.8.2. Эмулятор должен предоставлять пользователю возможность добавлять собственные системы в каталог. То есть у пользователя должна быть возможность создать собственный dataset в соответствующем каталоге и использовать при запуске

5. Модуль аутентификации

5.1. Реализация должна удовлетворять *TLS* (TLS Specification for Storage Systems, ISO/IEC 20648)

5.2. При использовании в реализации сертификатов, они должны удовлетворять X.509-v3

5.3. Эмулятор должен поддерживать базовую HTTP аутентификацию

5.4. Эмулятор должен поддерживать Redfish session login authentication

5.5. Эмулятором не должно требоваться создание сессии для пользователя, который использует базовую *HTTP* аутентификацию

5.6. Все “пишущие” операции (*POST*, *PUT*, *PATCH*, *DELETE*, кроме ресурса *Sessions*) должны требовать аутентификацию

5.7. Эмулятор должен удовлетворять требованиям на аутентификацию из спецификации. Эти требования включает подробное описание требований на реализацию механизмов аутентификации

5.8. Реализация аутентификации должна производиться компонентом *Authentication Service*, который сверяет переданные пользователем данные с данными (логин, пароль, ключ сессии) в базе данных или в другом внешнем хранилище

6. Модуль авторизации

6.1. Система должна поддерживать авторизацию, которая осуществляется с помощью модели ролей и привилегий (в том числе может быть поддержка *OAuth 2.0*)

6.2. В системе должны быть определены неизменяемые базовые роли

6.3. В системе должны быть заданы соответствия роль-операция для отслеживания доступности ресурса конкретному пользователю

6.4. В системе должен быть реализован “регистр привилегий” в виде *JSON* документа, состоящего из списка поддерживаемых ресурсов и написанный в заданном формате

6.5. Эмулятор должен реализовать Account Service — сервис выполняющий хранение паролей пользователей

6.6. Реализация авторизации должна производиться компонентом *Authorization Service*, который сверяет переданные пользователем данные с данными в базе данных (регистр привилегий и роли пользователя) или в другом внешнем хранилище

7. Модуль REST API

7.1. Эмулятор должен поддерживать *API*, описанное в спецификации *Swordfish*

7.2. Swordfish является *RESTful*, соответственно эмулятор тоже должен являтся *RESTful*. Фактически, это означает, что эмулятор должен удовлетворять 3 уровню *Richardson Maturity Model*

7.3. Эмулятор, должен поддерживать *HATEOAS*. Это требование реализуется наличием ссылок на связанные ресурсы (поля *Links{}* и *RelatedItem[]*) и наличие корневого ресурса *ServiceRoot*, из которого можно получить доступ к любому ресурсу

7.4. Эмулятор должен поддерживать все ресурсы, описанные в *Swordfish*

7.5. Эмулятор должен поддерживать все операции над ресурсами, описанными в *Swordfish*

7.6. Эмулятор помимо *REST* операций должен поддерживать *Actions. Actions* являются определенными функциями над ресурсом, к которому они относятся. Со стороны реализации никак не отличаются от операции над ресурсом

7.7. Должна быть поддержка эндпойнтов Redfish

7.7.1. */redfish* должен возвращать доступные версии (ресурс *Version*)

7.7.2. */redfish/v1/* должен возвращать ресурс *ServiceRoot*, который является корневым для всех остальных

7.7.3. */redfish/v1/odata* должен возвращать служебный документ OData

7.7.4. */redfish/v1/$metadata* должен возвращать метаданные Redfish

Нефункциональные требования

1. Общие требования на эмулятор

1.1. Эмулятор должен быть легковесным с минимумом потребления ресурсов

1.2. Эмулятор не должен требовать установки дополнительных зависимостей

1.3. Эмулятор должен быть спроектирован с возможностью расширять набор ресурсов по мере развития и изменения спецификации

2. Модуль хранения ресурсов

2.1. Хранение ресурсов не должно требовать от пользователя администрирования хранилища

3. Модуль конфигурации

3.1. Конфигурация должна иметь читаемый формат, понятный для пользователя, и не требовать от него погружения в исходный код

3.2. Конфигурация должна быть максимально гибкой, чтобы пользователь определял нужное ему поведение