**URL Shortener Prototype**

**Цель:** предоставить пользователям сервис для сокращения ссылок, ведущий учет переходов.

1. **Функциональные требования:**

GATEWAY API должен выглядеть следующим образом:

1. GET-запрос на *gateway-api/* должен предоставить пользователям справочную информацию о сервисе. Не важно, будет ли это UI, использующий дальнейшие endpoints, либо же простой текст.
2. Чтобы создать укороченную ссылку, пользователь должен послать POST-запрос на *gateway-api/* со следующими параметрами:

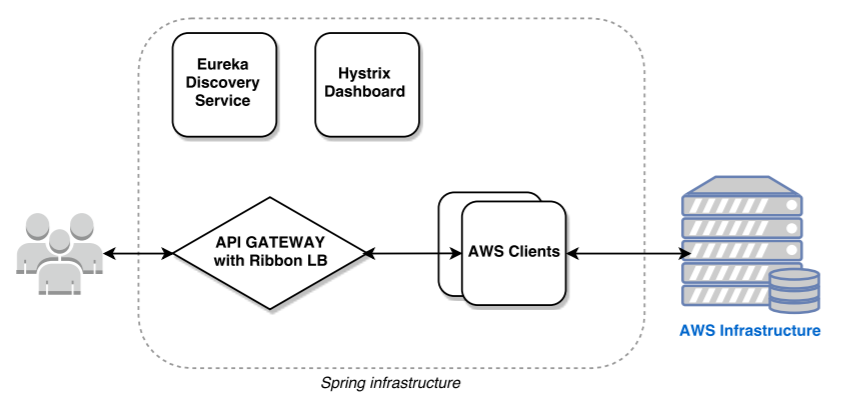
* Ссылку (*link*), которую нужно сократить
* Опционально, предоопределенный *key*, по которому ссылка будет доступна.
* Опционально, *expiration\_date* – дата удаления ссылки из системы, он не должен быть больше одного месяца.

Если *key* не указан – сервисы должны предоставить рандомный и уникальный ключ автоматически. Если *expiration\_date*, не задан, то выставлять дефолтный – одна неделя.

Ответом на запрос должна быть укороченная ссылка (адрес сервера должен предоставлять discovery service).

1. Чтобы перейти по укороченной ссылке, пользователь должен сделать GET-запрос на *gateway-api/{key}* (необходим редирект в браузере).
2. **Нефункциональные требования:**
3. Данные должны храниться в AWS, но т.к. их сервера расположены не во всех странах, в целях понижения latency, предлагается развернуть экосистему микросервисов spring на локальном(ых) сервере(ах). И максимально редко обращаться к AWS, используя техники кэширования. Например, в заголовках ответов на GET-запросы Cache-Control = public.
4. Система является прототипом, боевую нагрузку принимать не будет, тем не менее, нужно организовать все возможные оптимизации, для ознакомления с технологиями.
5. Инициализация инфраструктуры должна быть максимально простой, желательно в один клик. Первоначальная настройка AWS-аккаунта должна быть также в один клик.
6. Необходимо наличие discovery сервиса и мониторинга ошибок, последний должен быть доступен исключительно для администраторов.
7. Все взаимодействие по сети должно проходить через REST.
8. **Технические рекомендации**

К рассмотрению предлагается следующая архитектура:



**Технологический стэк:**

* AWS (DynamoDB, Lambda ?)
* Spring Boot 1.5.7
* Netflix Eureka (discovery service)
* Netflix Hystrix (мониторинг)
* Netflix Ribbon (с WeightedResponseTimeRule)
* Spring Data DynamoDB
* Spring Security (ограничение видимости dashboard'ов)
* Lombok
* Docker Compose

Вся спринговая инфраструктура должна быть описана в Docker Compose для моментального развертывания и независимого функционирования.

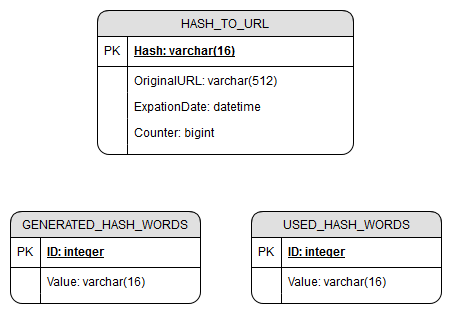
**Отдельные блоки в схеме:**

1. **API GATEWAY**

П.I.1 должен обрабатывать самостоятельно, остальное делегировать на один из AWS Client, согласно правилу WeightedResponseTimeRule риббона (Load Balancer). Взаимодействовать с сервисами через Rest Template.

1. **AWS Infrastructure**

Состоит из базы DynamoDB со схемой далее, актуальной из соображений наглядности для SQL-модели (схему нужно конвертировать и реализовать в NOSQL).



Здесь, в таблице HASH\_TO\_URL, хэшу ссылки соответствует набор из оригинального URL, срок актуальности и счетчик переходов (пока не учитываем последний).

GENERATED\_HASH\_WORDS – должен содержать все возможные комбинации латинских букв и чисел заданной длины (3 символа, после масштабирования необходимо будет 6).

USED\_HASH\_WORDS – должен содержать все уже *использованные* комбинации из предыдущей таблицы, наша задача – поддерживать ситуацию, когда каждая комбинация содержится лишь в одной из двух таблиц.

Здесь вырисовываются следующие задачи:

* При первоначальной инициализации AWS, необходимо запустить скрипт, заполняющий таблицу GENERATED\_HASH\_WORDS всеми возможными комбинациями.
* Периодически необходимо очищать в HASH\_TO\_URL просроченные ссылки (согласно ExpirationDate), поэтому нужно найти возможность создать джобу по расписанию на стороне AWS, которая по крону будет делать это. Возможно, это достигается с помощью AWS Lambda.

1. **AWS Client**

**Режим создания ссылки:**

Должен регистрировать в DynamoDB новую запись HASH\_TO\_URL. Для этого потребуется сгенерированная комбинация символов из GENERATED\_HASH\_WORDS.

**Режим получения ссылки:**

Должен подключаться к HASH\_TO\_URL, и возвращать все имеющиеся данные по соответствующему хэшу для API GATEWAY (который в свою очередь сам разберется, что именно из этого ему нужно, в зависимости от исходного запроса).

1. **Eureka**

С помощью стандартной конфигурации Spring Cloud Netflix Eureka обеспечить базовую отказоустойчивость сервисов с помощью схемы active-passive hot standby (когда один ломается – второй уже на подхвате).

1. **Hystrix Dashboard**

Необходимо предоставить администраторам (авторизация с помощью in-memory конфигурации Spring Security) мониторинг, визуализацию ошибок микросервисов. Подключить Spring Cloud Netflix Hystrix.

1. **Общая информация:**

* Ссылка на репозиторий: <https://github.com/Generalus/URL-Shortener-Prototype> (присылать туда PR)
* Над проектом работает несколько человек (2-3), если вы не успеваете, ваши задачи могут быть переданы.
* Большая часть идей взята с <https://www.educative.io/collection/page/5668639101419520/5649050225344512/5668600916475904> (доступно после регистрации, входит в бесплатную часть курса), в дальнейшем планируется реализовывать более сложные дизайны (с модификациями).

1. **Возможные варианты доработки сервиса в следующих версиях:**

* Прикрутить API Rate Limiter к API GATEWAY для ограничения по IP количества запросов на запись в час.
* Организовать кэш-кластер Redis между слоями Spring и AWS.
* Перенести вообще все на AWS и воспользоваться штатными средствами, пренебрегая повышенным latency.
* Добавить возможность просмотра количества переходов по укороченной ссылке.