Contents

1 URL Shortener Design Document

1.1 Context and scope

Team: Alexander, Yaroslav. Scope: University assignment. Project: URL shortener service

The service will allow users to create shorter links and use them instead of original ones. After link was created, anyone can use it through this service. Because this is a university assignment, service will not exhibit any complex behaviour and instead will do the bare minimum, namely:

- registration
- {un}authentication
- creation of short links
- deletion of short links
- redirection to the original links
- own KV(?)-storage

1.1.1 Optimisations and restrictions

- Taking into account the absence of e-mail verification, it was decided to simplify it a bit and allow to use any valid [a-Z0-9]+ string as a username instead of e-mail.
- Length of username and password is limited to MAX_{UNAME}$ MAX_{PW}$ chars. Length of link is limited to MAX_{LINK}$.

```
# size in bytes
MAX_UNAME=32
MAX_PW   =32
MAX_LINK =1024
```

- Max number of users - 255, sorry not sorry :(This derives from 'UUID' datatype.

1.2 Goals and non-goals

The following goals (in somewhat of modified state) derive from this specification.

1. Snapshot specification.pdf downloaded at *[2021-02-18 Thu 21:40]*

1.2.1 Goals

- User registration
- User authentication
- Shortened URLs' storage
- Registered users storage
- Creation and deletion of shortened URLs
- Url validation before
- Redirection to the original URL using shortened version

1.2.2 Non-goals

- E-mail verification / providing valid e-mail
- Complex user actions (user data management, password changing, etc...)
- UI/GUI/TUI/CLI
- ...

1.3 Architecture and Design

1.3.1 Structs

```
type UUID = Word8

data User = User
  { id       :: UUID -- ^ unique ID       -- PK
  , username :: Text -- ^ user's username -- Unique
  , hash     :: Text -- ^ password hash
  }

data Url = Url
  { id    :: UUID -- ^ unique ID                -- PK
  , orig  :: Text -- ^ original url
  , short :: Text -- ^ shortened version of url -- Unique
  , user  :: User -- ^ user that created a link -- FK user(id)
  }
```

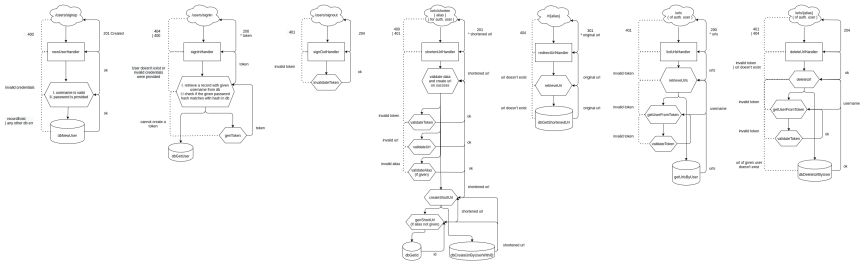
1.3.2 JWT tokens

```
data Token = Token
  { secret   :: Text
  , username :: Text -- ^ user's username
  }
```

1.3.3 Dependencies

- servant `web-server`
- colog `logger`

1.3.4 System-context-diagram



1.3.5 API

Main API specification: openapi.yaml

- Would be nice to add 409 Conflict (or any other more suitable code) in /users/signup, when username already exists.

1.3.6 Data Storage

There are gonna be two tables a.k.a. files:

- users
- shortened links

Their representation is quiet the same as datatypes in haskell.

1. **TODO** replace with their *actual* representation.
2. **TODO** describe an algorithm of traversing through the DB Probably it's gonna be a RB-tree, but need to think about the implementation part.

Common Workflows

- insert new user
- retrieve user credentials
- generate UUID for new link
- insert link
- delete link