System Design

FEATURES

- Given a long URL, the service should generate a shorter and unique alias of it.
- When the user hits a short link, the service should redirect to the original link.
- Links will expire after a standard default time span.
- The system should be highly available. This is really important to consider becauseif the service goes down, all the URL redirection will start failing.
- URL redirection should happen in real-time with minimal latency.
- Shortened links should not be predictable.
- Short URL length can be up to 30 characters starting from prefix: www.habuild.in/

REQUIREMENTS

Assuming, we will have 5M new URL shortenings per month, with 100:1 read/write ratioDatabase to be used : PostgreSQL

Methods to implement

1. Shorten Url (Destination Url) → Short Url
2. Update short url (Short Url, Destination Url) → Boolean
    a. update meaning : new destination link on same short link
3. Get Destination Url (Short Url) → Destination Url
4. Update Expiry (Short Url, Days to add in expiry) →Boolean

LONG URL:
https://www.facebook.com/wbdhhoscncoovdvmsnvocjoscsoefoegg0ri0

SHORT URL: www.habuild.in/7w98QV1jD

TRAFFIC and SYSTEM CAPACITY

Traffic

Given 100:1 read/write ratio

Number of unique shortened links generated per month= 5 million

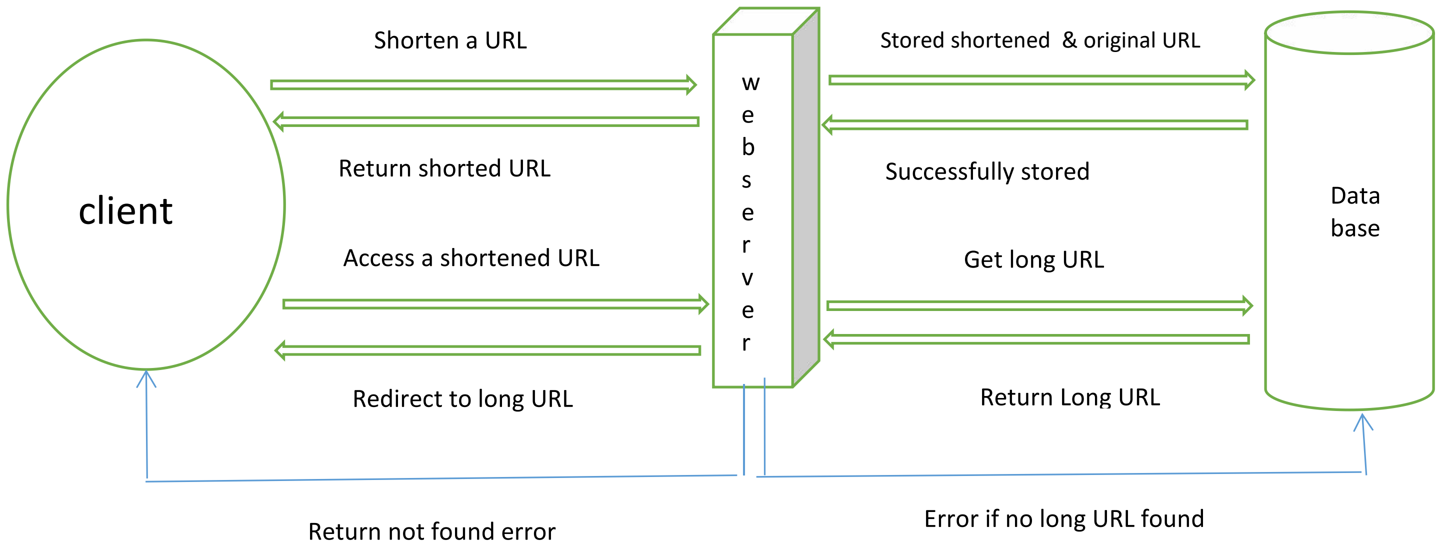
Number of unique shortened links generated per seconds= 5 million/(30days*24hrs*3600s) ~ 2 URLS/seconds

With 100:1 read/write ratio, number of redirections = 2 URLs/s * 100 = 200 URLs/s

Storage

Assuming we store every shortened link for 1 year and 5M links creation per month. For this period the service will generate (5M*12months) 60 M records.

DESIGN


Shorten a URL
Return shorted URL
Access a shortened URL
Redirect to long URL
client
w e b s e r v e r
Stored shortened & original URL
Successfully stored
Get long URL
Return Long URL
Data base
Return not found error
Error if no long URL found

DATABASE SCHEMA

ShortLink Data

1. Original URL : long Url
2. Short URL : shortened link
3. ID : unique ID for every url
4. Expiry date : date of expiry

Rest EndPoints

Create:

POST req to store the long URL along with its short alias (generated) and expiry date in the database.

Read:

GET req when we click on the shortlink it will redirect to the original link .

UPDATE:

PUT/PATCH req to update the original URL and expiration date in database.

Shortening Algorithm

1. URL encoding through base62
2. URL encoding through MD5
3. Key Generation Service

Pseudo Code

- Create a new project in VS studio
- Initialise node by npm init in the terminal
- Install dependencies like express
- In index.js file require the express framework and start your server :

```
     const express= require('express');
    const app=express();

    app.listen(3000,()=>{
          console.log("server is starting")
     });
```

- Create table in postgres according to the schema given in the earlier part of this article.
- Now connect the data base with node using pool

```
    const Pool=require('pg').Pool;
    const pool= new Pool({
   user:    ,
   password: ,
   database: ,
   host:  ,
   port:
    });
    pool.on('connect', () => {
          console.log('connected to the db');
   });
   pool.on('end', () => {
          console.log('client removed');
   });
```

- In route.js path we will handle all our requests

```
    const {Router}= require('express');
    const router=Router();
```

```
router.get('/:id', async(req,res)=>{
try{
        const result  = await pool.query(‘postgres
        query to find the data using id’,[id])
      If(result)
        {
                 Check expiry date is over or not
                If(not over)
                        Redirect to the original url
                 else
                        Send error
          }

        }
catch(err){
        Throw err
}
});

router.post('/',async(req,res)=>{
try{

        const result  = await pool.query(‘postgres
        query to insert data’,Values from req.body)
        res.send(added successfully)

}
catch(err){
        Throw err
}

});

router.patch(‘/:id’,async(req,res)=>{
 try{
        const result  = await pool.query(‘postgres
        query to find the data using id’,[id])
```

```
                If(result)
                        Update the long url taking value from
                        req.body
                Else
                        Send error
        }
        catch(err){
                Throw err
         }
});

router.patch(‘/:id’,async(req,res)=>{
         try{
                const result  = await pool.query(‘postgres
                query to find the data using id’,[id])
                If(result)
                        Update the expiry date taking value from
                                req.body
                Else
                        Send error
        }
        catch(err){
                Throw err
         }
});
```

- Now define the routes in index.js

```
        const routes= require('./routes');

        app.use(‘/’ ,routes)
```