PropTiger, Housing, Makkan

URL Shortener

January 25, 2019

-Prepared By: Darsh Vajaria

-Mentored By: Rajat Agarwal

Overview

Url shortener service is used to share shortened url via SMS(saves valuable characters in it) or link via Mail(easy to share) etc. In this project, I have created a service which primarily shortens url for a given long url and give long url for a short url. SHA-256 algorithm is used to achieve unique short URLs along with expiration time for each short url to exploit reusability and space constraints. Apart from this, analysis pertaining to clicks and total requests/API calls were also undertaken.

Goals

1. Shorten url with minimal or no repetitions.
2. Exploit reusability of short url, i.e. try using a short url for multiple long urls.
3. Analysis: Number of clicks for given short url, number of times services used.

Technical Specifications

1. Language: JAVA
2. Framework: Spring
3. ORM: Hibernate
4. Database: MySQL

5. Caching: Redis
6. Uniqueness algorithm: Bucketed SHA-256
7. Server: Tomcat

Detailed Working/Implementation

Database Design

1) url

   Columns: longURL (VARCAHAR), shortURL (VARCHAR)(Primary Key), domainName (VARCHAR), secPassed (BIGINT)

2) report

   Columns: date (Date) (Primary Key), POST requests (INT), GET requests (INT)

3) analytics

   Columns: longURL(VARCHAR), hash (VARCHAR), clicks (INT), id (INT) (Primary Key)

hash field is indexed. Hash field is deliberately included so that search process becomes time efficient. Searching on longURL is time consuming task.

Requests

1) POST Request

   https://localhost:8090/URLShortener/post

   Input JSON : {longURL, domainName(optional)}

   Output JSON: {longURL, shortURL, domainName}

Processing:

File of longURL is converted into 64 hexadecimal string using SHA-256 algorithm. Java has inbuilt library (java.security.MessageDigest) to execute this algorithm. probableShortURL is domain name + certain consecutive bits of hash. Now a probableshortURL is considered valid or the output shortURL, if it follows any of the two following conditions:

-> There is no conflict as a completely unique shortURL has been generated.

-> In case of conflict in url database, the conflicted shortURL must have expired thus the probableshortURL can be used.

What exactly does ‘certain consecutive bits’ mean?

Start with first 8 consecutive characters of hashtext. ShortURL is domain name + 8 characters. Check for validity, if conflict then consider next 8 bits. A stage will come when we will be considering last 8 bits. After that consider first 9 bits and similarly traverse 64 characters till we get a valid hash. It is guaranteed to get a unique hash because total possibilities of SHA-256 is far more than total number of current URLs.

The expiration time of link is manipulatable. secsPassed column stores the number of seconds passed from 1970 till that moment. Java has inbuilt method to calculate this. If difference between secsPassed of conflicted shortURL and probableshortURL is greater than number of seconds in a year than the conflicted shortURL is expired and probableshortURL is the output shortURL.

Once we get shortURL, we save url object in url table and return url object in JSON format.

Moreover a check method is also defined so that if longURL is already present in url database then, there is no need to follow the whole algorithm described above. Just search by longURL in url table and return the tuple updating secPassed field in url table.

2) GET Request

https://localhost:8090/URLShortener/get?surl=”<input shortURL>”

Input Request Parameter: shortURL

Output JSON: {shortURL, longURL, domainName}

Processing:

shortURL is primary key of url table. Url table is searched using the input shortURL through urlDao class. If a tuple is obtained then simply return it (ignoring the secPassed filed) else return no such url exists(no such URL has done POST request or the URL is incorrect). For any shortURL only one longURL can exist at a time. updateAnalytics() method is called to increment number of clicks for the longURL which is mapped with given shortURL.

3) Report Request

https://localhost:8090/URLShortener/report

Input : /report or /report/{reportDate} as GET request

Output JSON: {date, POST requests, GET requests}

Processing:

Report is an add on functionality. It gives number of POST requests and number of GET requests for any given day or all the days if no date is specified. Whenever a POST request is executed, update POSTs method increments number in the report table for current date and same for GET request. Java has inbuilt method to get today's date which easies the querying process in report table. Passing date as path variable will give output for the particular date.

4) Analytics Request

https://localhost:8090/URLShortener/analytics?surl="<input shortURL>"

Input: /analytics?surl="<input shortURL>"

Output JSON: {longURL, clicks}

Processing:

Analytics give number of clicks for particular url. longURL for given shortURL is fetched from URL table. The reason this needs to be done is given shortURL can have more than one longURL till date, though not simultaneously. Calculating clicks on shortURL will be meaningless.

After getting longURL, a hash is generated for it using MD5 algorithm (16 bytes output). Java has inbuilt library for MD5. Reason for hashing is searching on longURL will take more time and moreover indexing on text is bad design thus hash is generated. SHA-256 is not used because of space constraints.

If given hash is already present in analytics table the just increment the clicks else save analytics object and initialise clicks to 1.

Caching and Async

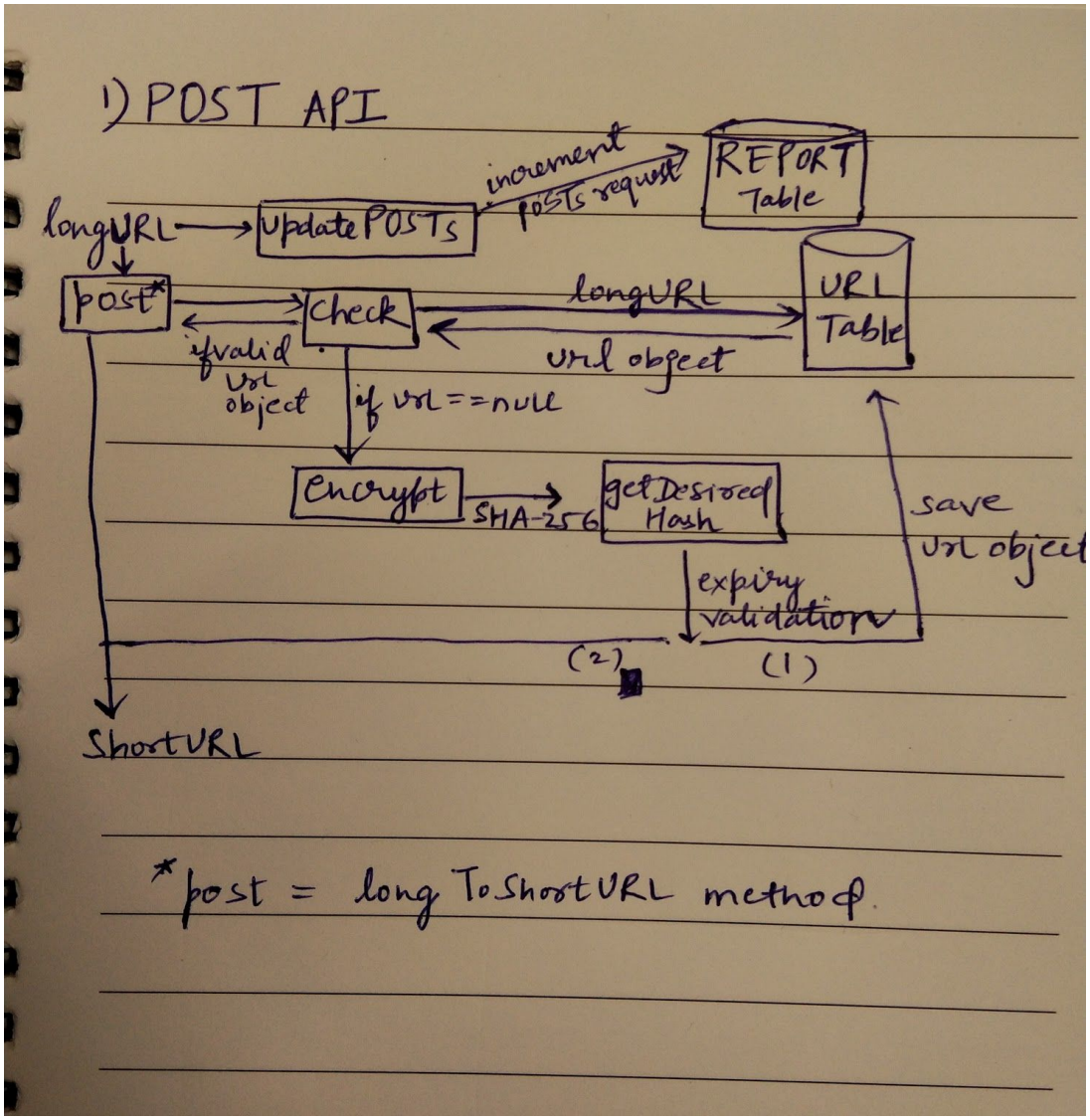
I have used Redis for caching. Redis is key-value database with ability to manipulate high level data structures. I have used <String,URL> as my key-value pair. Cache expiration time is set to 10 minutes.

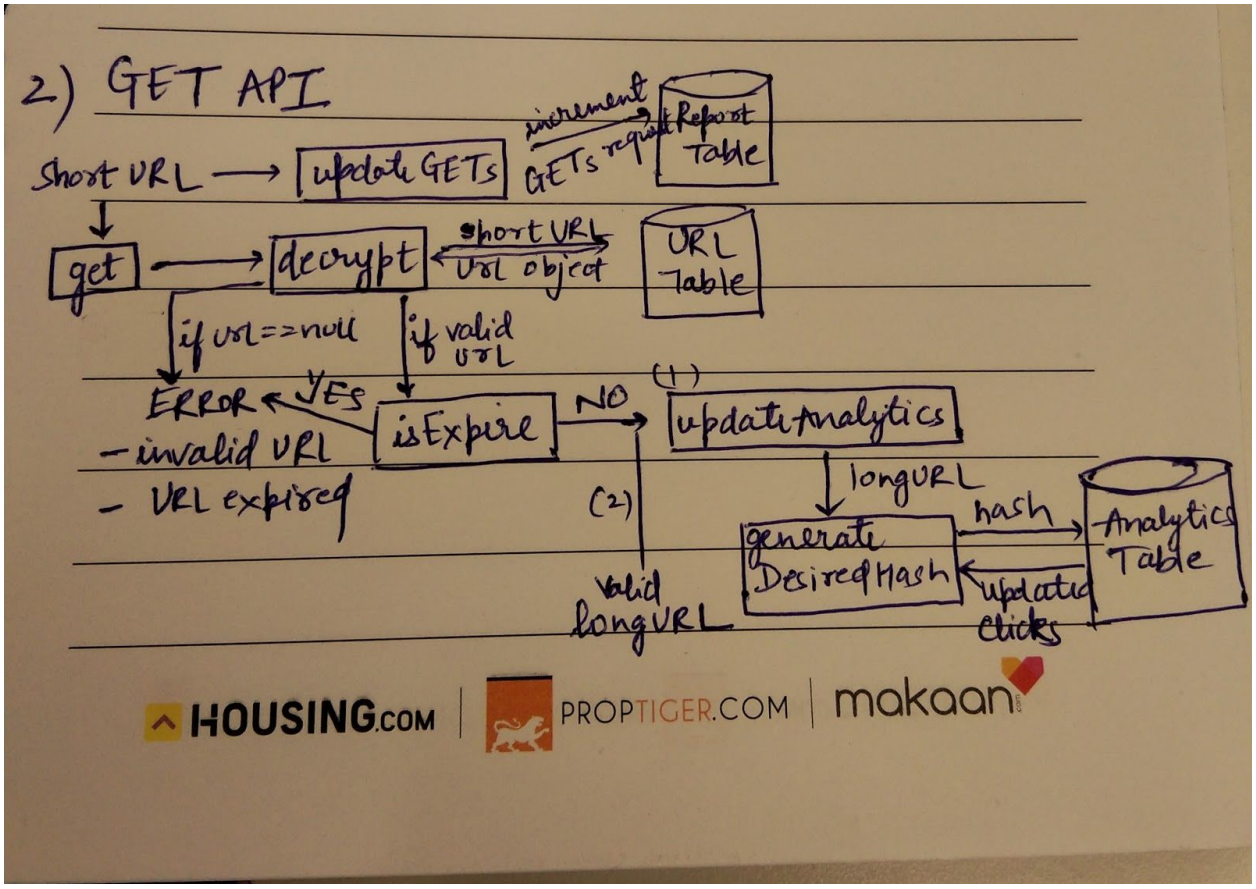
check() method of urlService class is cached. This method checks if longURL is already present in url table. To do so, I need to search longURL field through the url table which may be time consuming. Caching <String, URL> will help speed up the search process. check() method is called from longToShorturl(). Both of these methods are in urlService class and to achieve caching so that framework understands, new bean of urlService class has been created by calling getBean() method of appContext object(instance of Application Context class).

Spring has @async annotation to enable asynchronous execution for a given method. @Async will execute it in separate thread i.e. caller will not wait for completion of the caller method.

updatePOSTs and updateGETs methods of urlService class class are annoted @Async. This will not make encrypt and decrypt method of urlController to wait for the GET and POST update methods. This also help to speedup requests.

Flow Diagram





Why SHA-256?

1) Output of SHA-256 is 32 bytes, 64 characters (hexadecimal). Considering 8 characters, gives 36^8 = 3 trillion possibilities. Moreover the algorithm designed covers much more than these possibilities because of reusability and increased bucket size.
2) Reusability - Every shortURL can be attributed with expiry time. Once the shortURL expires for given longURL, we can reuse it for some another longURL.
3) Decryption of shortURL is computationally impossible task.
4) Dynamic length of shortURL possible.