Dilip Sharma

Confidential & Sensitive

URL Shortner

Design Document

Module.:	URL Shortner
Revision No.:	1.0
Date:	19th Sept 2023

DOCUMENT CONTROL

	NAME	POSITION
Author	Dilip Kumar Sharma	Tech Lead
Reviewed	TBD	TBD
Approved	TBD	TBD
Location	TBD	

REVISION HISTORY

REV	DATE	DESCRIPTION
1.0	20th Sept 2023	Document creation

TERMS & ABBREVIATIONS

TERM	DEFINITION

REFERENCED DOCUMENTS

REF NO	DOCUMENT NAME	SERVER LOCATION

Table of Contents

1 Introduction 6

1.1 Purpose 6
1.2 Scope 6
1.3 Audience 6

2 Design Overview 6

2.1 About 6
2.2 Assumptions 6
2.3 Constraints 6
2.4 Limitations 6
2.5 System Environment 7

3 Design Goals 7

3.1 Independent Component 7
3.2 Simplicity 7
3.3 Scalable 7
3.4 Flexible 7
3.5 Readable and Understandable 7
3.6 Highly Available 7
3.7 Low Latency 8
3.8 Unpredictability 8
3.9 Approach 8

4 Application Architecture 8

4.1 High Level Architecture Diagram 8
4.1.1 Client 8
4.1.2 Url Shortner Service 8
4.1.3 Encoder - Decoder 9
4.1.4 Cassandra Database 9
4.2 Complete Architecture Diagram 9
4.2.1 Data 9
4.2.1.1 Configuration 9
4.2.1.2 Database 9
4.2.2 DAL 10
4.2.2.1 DB DAOs 11
4.2.3 Core Logic 11
4.2.3.1 UrlShortner 11
4.3 Class diagram 11
4.4 Service and DB relationship 11

5 Others 11

5.1 Environment Variables 11

5.2 To Do 12
5.3 References 12

1 Introduction

1.1 Purpose

This document details the technical design and implementation of Url Shortner.

Its main purpose is to detail the functionality which will be provided by each component and show how the various components interact in the design.

1.2 Scope

The scope of this document is limited to designing of Url Shortner component.

1.3 Audience

The intended audiences for this document are, Project manager, the project development teams, technical architects, database designers, testers.

2 Design Overview

2.1 About

URL shortening service is used to generate shorter aliases for long URLs. In this kind of web service if a user gives a long URL then the service returns a short URL and if the user gives a short URL then it returns the original long URL.

Deletion: Users should be able to delete a short link generated by our system, given the rights.

2.2 Assumptions

- Users

1M users per day

Default Keep alive time – 1 Year

URL len – 7 Char

Storage

Avg URL size – 2 KB – 2048 Bytes

Short URL Size – 7 Bytes

Created At – 7 Bytes

Expiry Time – 7 Bytes

2.3 Constraints

- TBD

2.4 Limitations

- Not all edge cases, for url shortner service, have been covered in this document.

2.5 System Environment

- This design supports Docker, Docker Compose.
- This is implemented in Python 3.8.

3 Design Goals

3.1 Independent Component

- We need to bring uniformity, not only in python source code, but also in DB tables structure.

3.2 Simplicity

Source code and Database table structure should be less complex or say should be simple to understand.

- Simplifying the source code structure.
- Simplifying the database table structure.
- The short links generated by our system should be easily readable, distinguishable, and typeable.

3.3 Scalable

To be able to scale both up and down to support; -

- Varying number of users.
- Our system should be horizontally scalable with increasing demand.

3.4 Flexible

Ability of the application to adapt and evolve to accommodate new requirements without affecting the existing operations.

Flexibility in the application can be achieved by; -

- Use of Gang of Four design patterns.
- Use of SOLID principle of design patterns.
- Use of object-oriented programming principles.
- Designing database tables.

3.5 Readable and Understandable

Software is meant for modification/improvements. Fellow developers should be able to understand the code.

This could be achieved by; -

- Coding guidelines.
- UML diagrams.
- Sequence Diagrams
- Comments/Description of classes and methods used.
- Documentation (Doc string comments in Python), ReadME.

3.6 Highly Available

Our system should be highly available, because even a fraction of the second downtime would result in URL redirection failures.

Since our system's domain is in URLs, we don't have the leverage of downtime, and our design must have fault-tolerance conditions instilled in it.

3.7 Low Latency

The system should perform at low latency to provide the user with a smooth experience.

3.8 Unpredictability

From a security standpoint, the short links generated by our system should be highly unpredictable.

This ensures that the next-in-line short URL is not serially produced,

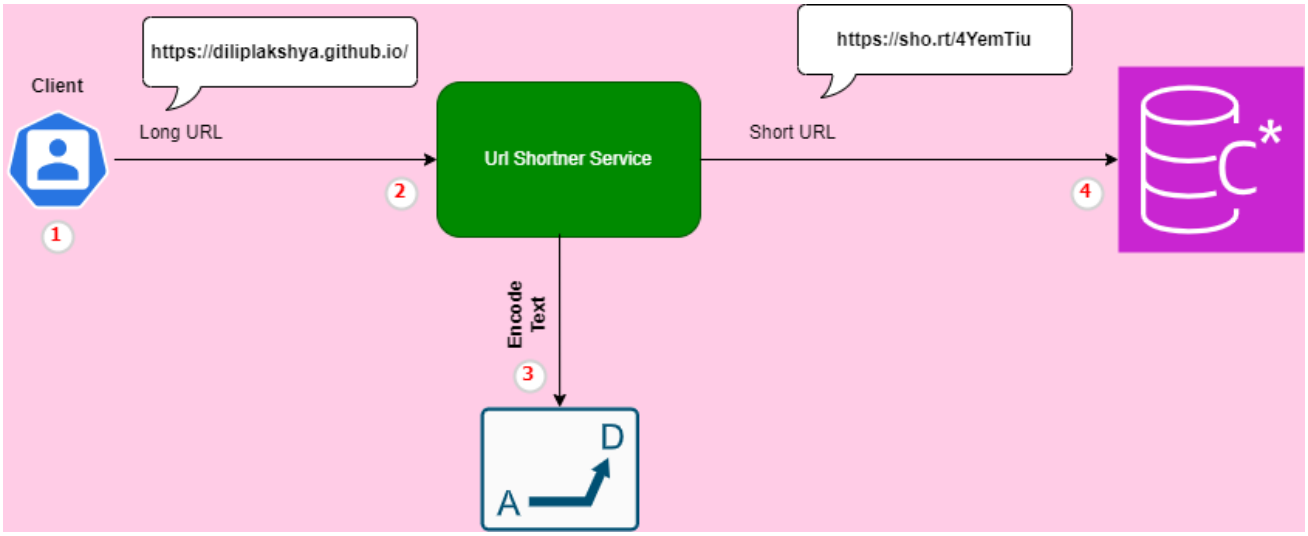
Eliminating the possibility of someone guessing all the short URLs that our system has ever produced or will produce.

3.9 Approach

- Expiry time: There must be a default expiration time for the short links, but users should be able to set the expiration time based on their requirements.
- Url should not be predictable.
- Need to design both python source code as well as DB table structure.
- Adding DAL between core logic and data.

4 Application Architecture

4.1 High Level Architecture Diagram



4.1.1 Client

- The user requesting URL Shortner service from browser or any rest api client.

4.1.2 Url Shortner Service

- Main service responsible for shortning the long url as well as returning long url when short url is passed.

4.1.3 Encoder - Decoder

Responsible for encoding the long url text into short text.

4.1.4 Cassandra Database

Database to store long and short urls.

4.2 Complete Architecture Diagram

4.2.1 Data

4.2.1.1 Configuration

- Configuration files to hold Url Shortner's static information.

.env

Environment Variable	Ex. VALUE

logging.json

Environment variable: ***US_LOG_CFG***

ATTRIBUTE	VALUE
FileName	/data/log/dbglog/url_shortner.log
Format	[%(levelname)s] %(asctime)s: [%(threadName)-17s] [%(filename)s:%(funcName)s:%(lineno)d] %(message)s
Level	10

Logging Level; -

LEVEL	NUMERIC VALUE
NOT SET	0
DEBUG	10
INFO	20
WARNING	30
ERROR	40
CRITICAL	50

4.2.1.2 Database

All ID’s to be GUIDs.

All timestamps are in UTC.

4.2.2 DAL

Data Source; -

- It could be any type of database.

DAL/DAO; -

- DAO is part of DAL which helps us to create a loose coupling between Core logic and data source (Database/Config).

Uses; -

- DAL/DAOs are used by core logic to search or update records in data source.
- For searching records, core logic will receive a Transfer Object from DAO as a result of data access.
- For updating records, core logic will construct this Transfer Object and send it to DAO.
- With the use of Transfer Object, core logic will be able to work without any need of code change in core logic even if we change the data source.
- Hence, core logic is tightly coupled with this Transfer Object.

Design Pattern; -

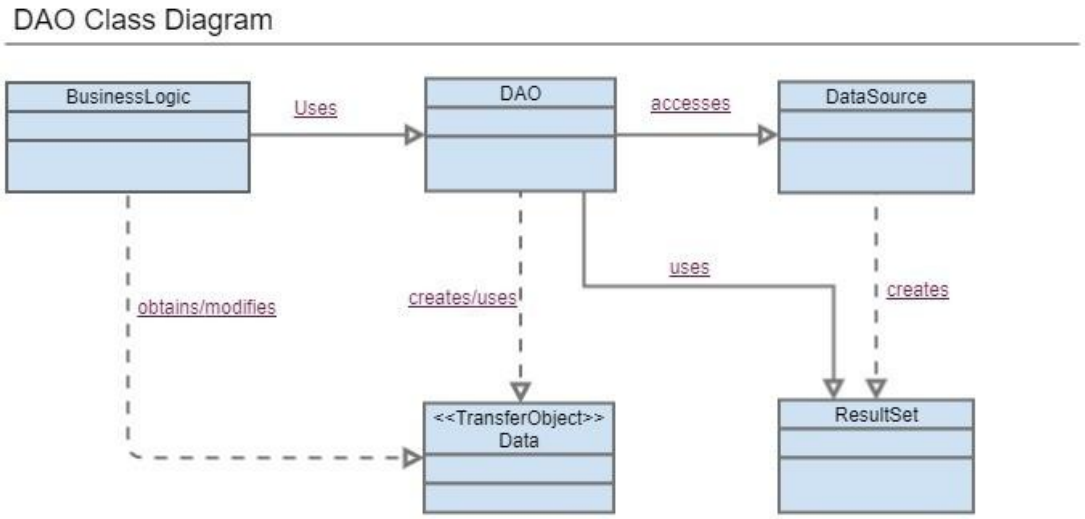
- DAOs are created by a DAO Factory.
- Both DAO factory and DAOs are part of Abstract factory design pattern.

Transfer Object; -

- Each such DAO will receive / return a Transfer object from / to core logic.
- The class structure of this Transfer object will remain same irrespective of the database used or the project in which this Matching Server is used.
- Core logic is tightly coupled with this Transfer object and fully dependent on this transfer object.

Project Directory; -

TBD



4.2.2.1 DB DAOs

- It is a bridge which provides loose coupling between Database and Url Shortner Core logic.
- For each database access there will be a separate DAO.
- This will be implemented using Abstract Factory Design Pattern.
- Type of DB DAO Factory will be decided at run time by reading environment variable to avoid code change.

DB DAO Factory; -

Sr. No.	Env Var	Value	Factory Type/DB Type
1		1	Cassandra
2		2	TBD

4.2.3 Core Logic

4.2.3.1 UrlShortner

4.3 Class diagram

TBD

4.4 Service and DB relationship

5 Others

5.1 Environment Variables

- All environment variables used in this document need to be exported permanently during deployment of the build.
- We need to make sure Url Shortner can read these environment variables on start up.

5.2 To Do

Edge cases for Url Shortner; -

- TBD.

5.3 References

TBD