High Level Architecture

Eatilink is a link shortener service which is able to convert a provided url into a fictional shortened url. In eatilink product scenarios, authentication will be handled centrally. Gateway is a good place to authenticate microservices. To get response from link shortener service, users have to authenticate through an authentication server. After validating credentials, the authentication server will return a token to invoke link shortener service.

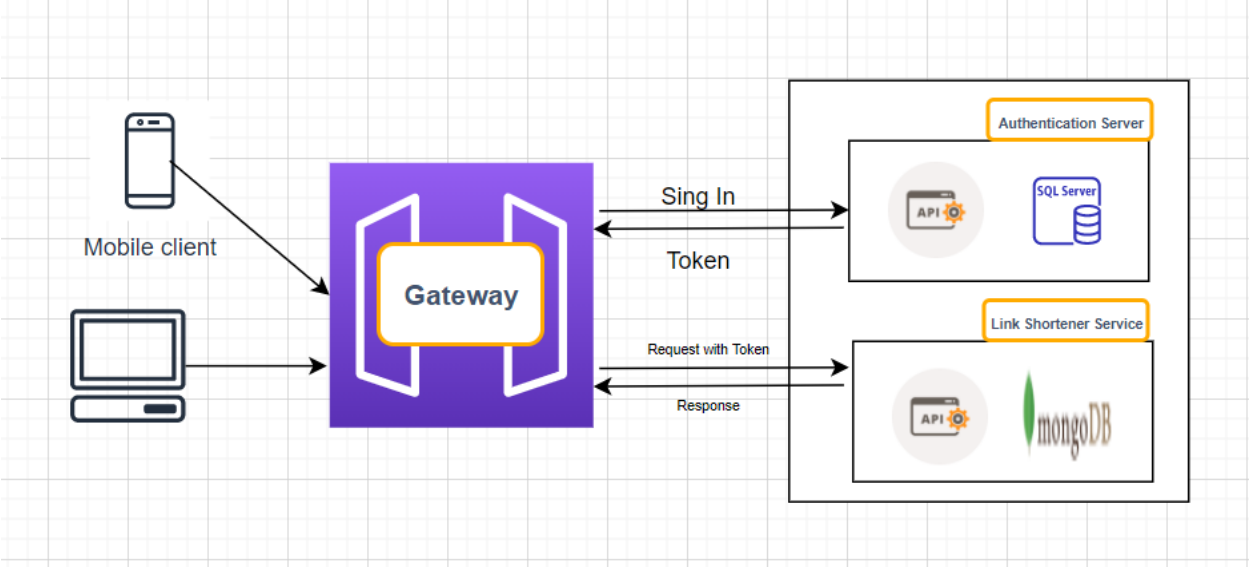
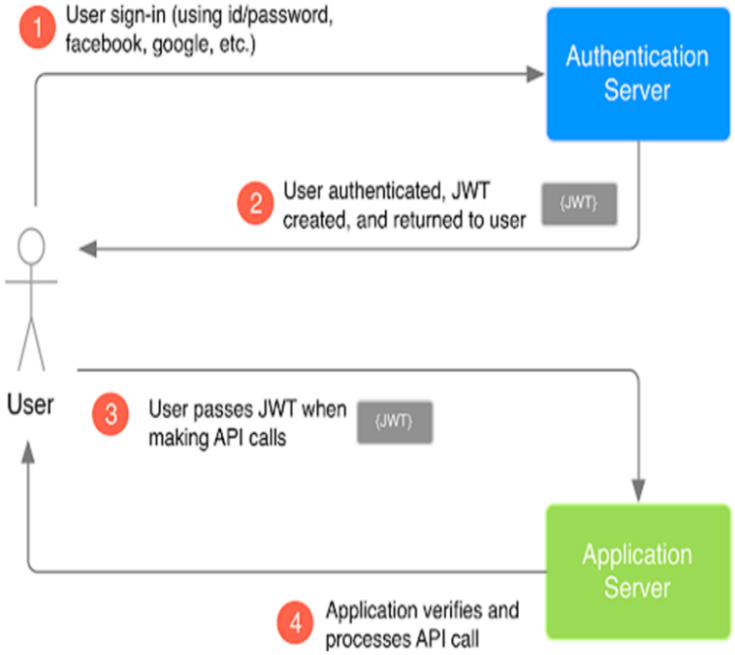


Figure 1: High level architecture of Eatilink

Low Level Design

Authentication



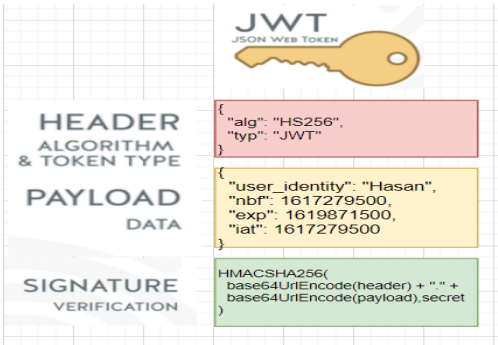
Assuming the user sends an authentication request (login request) to the authentication server with credentials (username and password).The Authentication Server takes this information and queries it from the database, combines the user name with its secret key and generates the token.When the user authenticates, the Authentication server returns **token** and the server writes this information to the related user's Token column in the database.The user passes **JWT token** when making API calls into header part of the request and the link shortener server checks each time whether the token is valid or not.

Link Shortener Service verifies and processes API calls.

JWT Structure

The token produced with JWT consists of 3 main parts encoded with Base64. These are **Header** (Header), **Payload** (Data), **Signature** parts.

Let's take a closer look at these parts. If we pay attention to the token example on the side, there are 3 fields separated by dots in the token form.

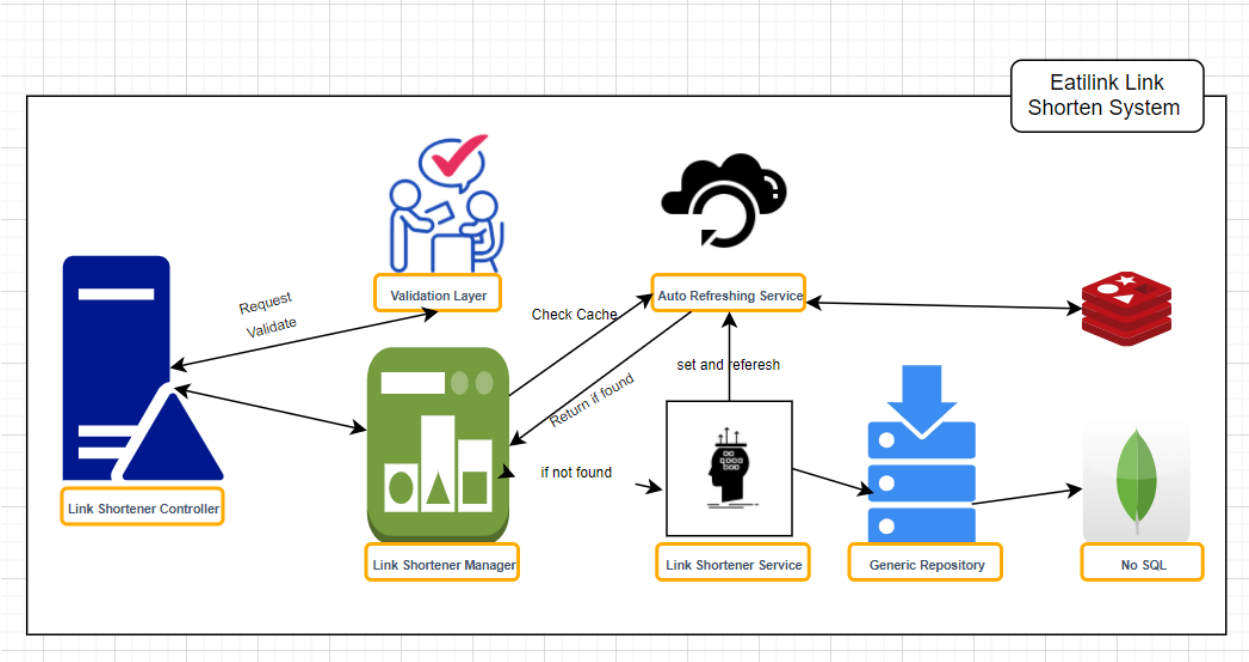


Link Shortener Service

Eatilink is a link shortener service like Bitly or TinyURL. It is able to convert a provided long url to fictional , shortened url.

Low level architecture

Eatilink data will be saved in no sql server (MongoDb) and cache will be saved in memory cache server which is frequently accessed with consideration of auto refreshing cache.



Requirement and Goals of the system

Functional Requirements

1. Given a URL, link shortener service will generate a shorter and unique alias of it. This is called a short link.
2. When users access a short link, Eatilink domain should redirect them to the original link.
3. Auto refreshing caching solution, where shortened URLs are stored longer if they've been used/accessed within its time limit.

Non-Functional Requirements

1. The system should be highly available. This is required because, if our service is down, all the URL redirections will start failing.
2. URL redirection should happen in real-time with minimal latency.
3. Shortened links should not be guessable (not predictable).

Encoding Actual URL

We can compute a unique hash (e.g., MD5 or SHA256, etc.) of the given URL. The hash can then be encoded for display. This encoding will be base62 ([A-Z, a-z, 0-9]).

Data Partitioning and Replication

In this scheme, we take a hash of the object we are storing. We then calculate which partition to use based upon the hash. In our case, we can take the hash of the 'key' or the short link to determine the partition in which we store the data object. That will be future work. Currently generating a short url to integer value by base62 algorithm to store as a unique id.

Cache

System caches URLs that are frequently accessed byMemcached, which are stored original URLs with short url by hashtable. Before hitting backend storage, the application servers can quickly check if the cache has the desired URL.

Database Design

Eatilink database requirements follows :

1. Billions of records need to be saved.
2. Each object will save as small as possible data.
3. There is no need for a relationship between records.
4. The system has a high read rate.

Database Schema

Currently Eatilink has implemented one main table to store url and expiry related information. In the future, user table consideration will be much appreciated.

URL	
Uid	***string***
OriginalUrl	***string***
ShortUrl	***string***
Domain	***string***
ShortString	***string***
CreateDate	***string***
ExpiryDate	***string***

Logging and error handling

Eatilink system is properly logged scenario basis and has handled all possible exceptions. Detailed error handling code and message will be added to the API specification error code section.

Error Codes

Http Code	Error Code	Description
200	OK	Success
400	BAD_REQUEST	Wrong API Versioning
401	INVALID_ACCESS_TOKEN	Invalid Token
		Token is Expired
		User Identity is not present inside JWT payload
		JWT algorithm is not HMAC SHA256
	UNAUTHORIZED	Operating system denies access.
404	NOT_FOUND	The requested resource does not exist on the server
422	EMPTY_ORIGINAL_URL	Original Url is empty
	INVALID_ORIGINAL_URL	Invalid Original Url
500	INTERNAL_SERVER_ERROR	Casting or divide by zero etc problem
502	DATABASE_CONFIGURATION_ERROR	MongoDB configuration error.