Shortened URL

Jan 2022

Status: In Progress | **Ready for Review** | Final

[Requirements](#_csukbxnyvgde)

[Objective](#_3hzff1f2d4l)

[Goals](#_5an20mct0jm)

[Non-Goals](#_fr2ymddzc9ju)

[Tech Design](#_1nv6m79y3y4k)

[Architecture](#_29d0mw1zocsj)

[Overview](#_93rddzwjviq0)

[User interaction sequence diagram](#_tahi866f8ip)

[Shortened URL Generation Algorithm](#_msmfm7k1q6iz)

[Database Model](#_1g8ssnwaac7r)

[Frontend - Backend API](#_qim4fss69m9i)

[Frontend UI Desi](#_8ro5f6eluuv8)

[Risks](#_b966bd5rz4jd)

[Revision History](#_af2i0stv539r)

# Requirements

Provide a UI allowing users to convert a full URL to a shortened URL and vice versa.

* Allow the user to convert a full URL to a shortened URL.
* Allow the user to convert the shortened URL to the original URL.

# Objective

## Goals

* Provide a UI for users to get/resolve a shortened URL.

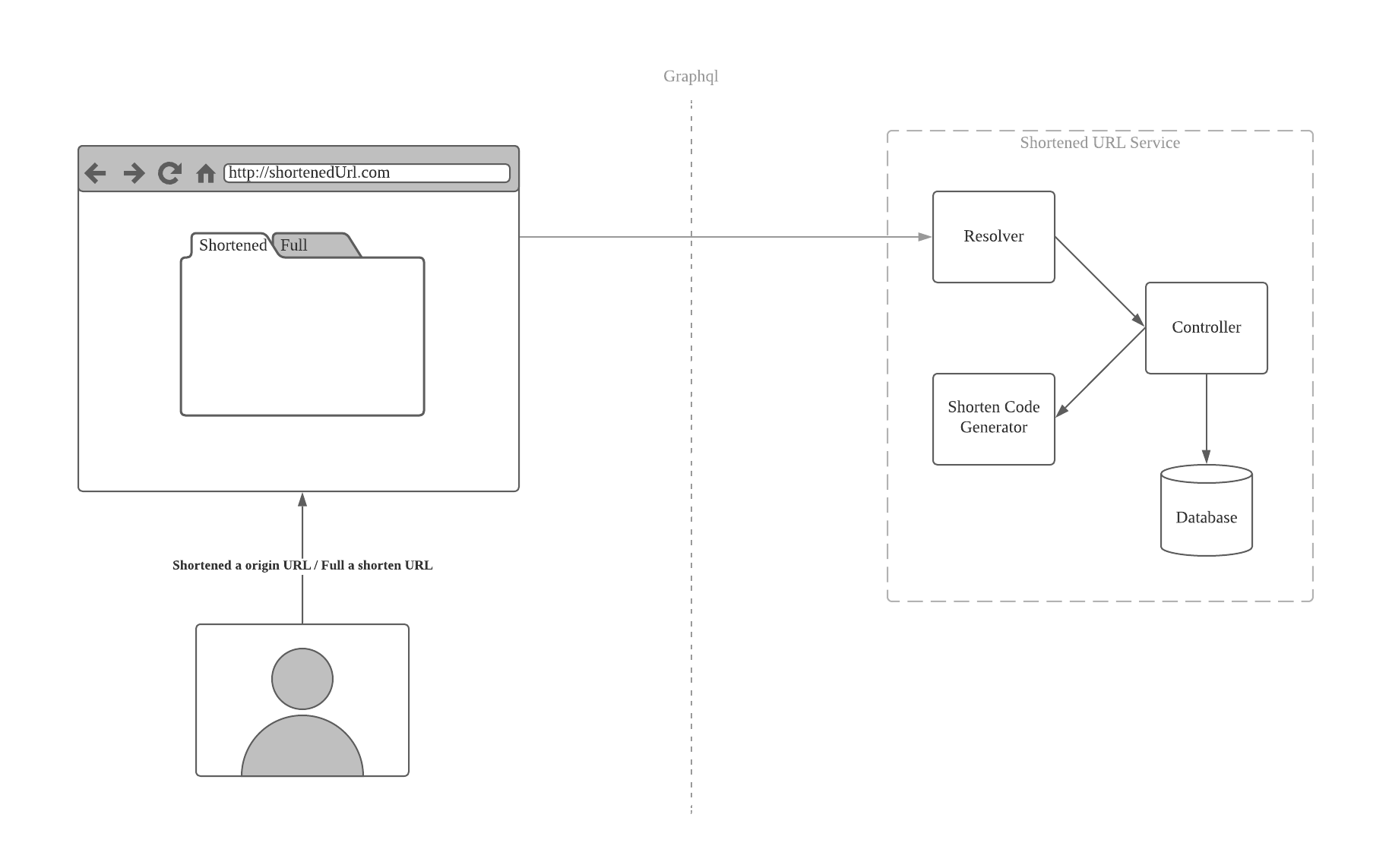
## Non-Goals

* Don’t require when the URL is input at the browser, the page will be directed.

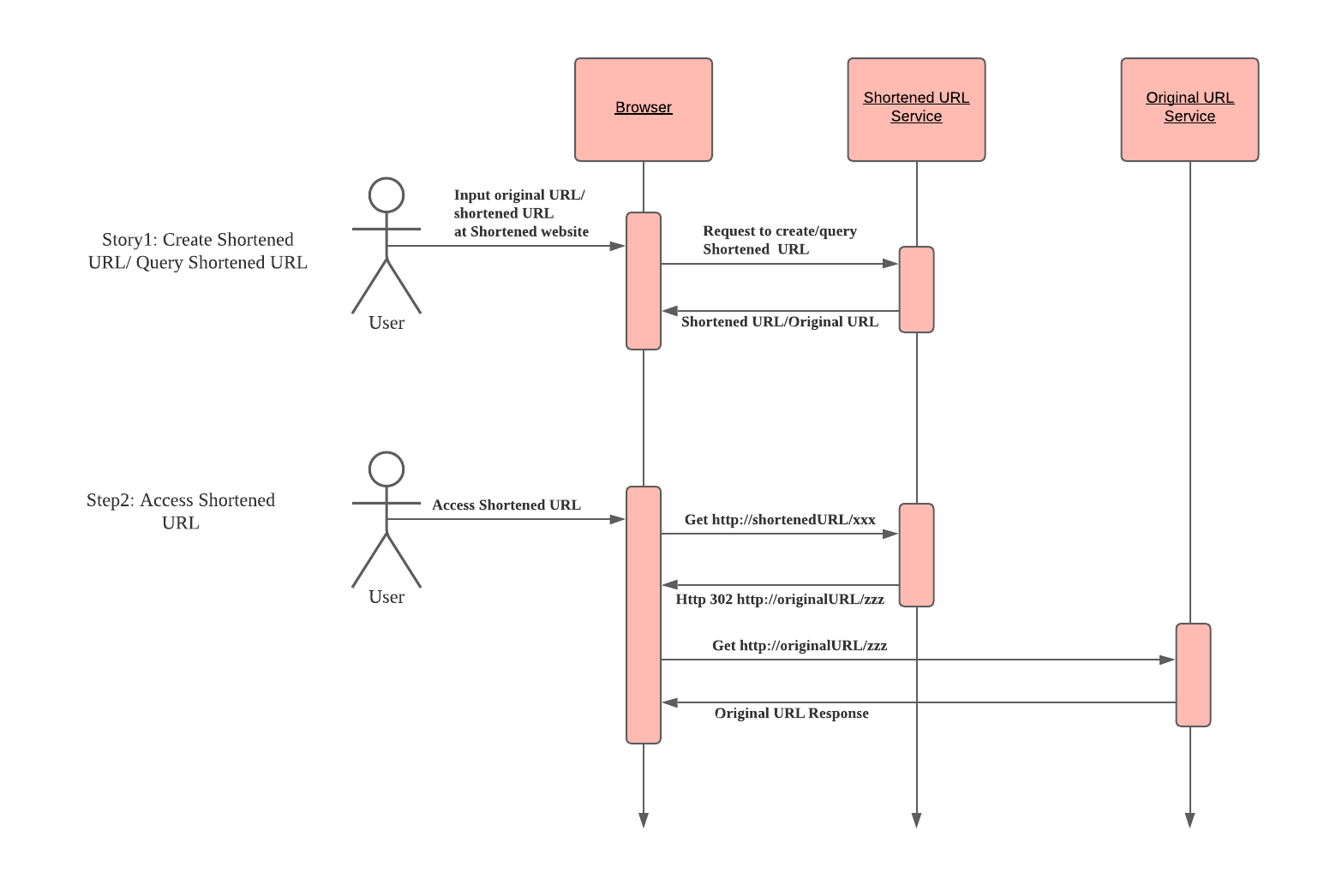
# Tech Design

## Architecture

### Overview



### User interaction sequence diagram



## Shortened URL Generation Algorithm

We shall use this algorithm to map an original URL to a shortened URL, just like the mapping:

x -> f(x)

with two constraints:

1. If x1 != x2, then f(x1) != f(x2);
2. the length of literal f(x) shall be less than x.

Based on the requirements, we can use [Base64](https://en.wikipedia.org/wiki/Base64) to achieve the goal:

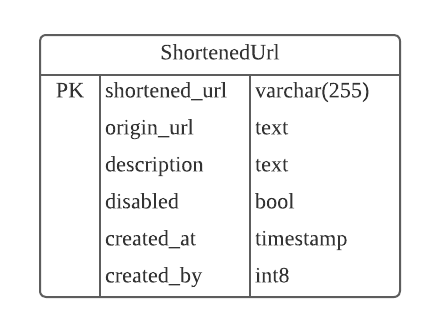
| private static final char[] base64URL = {  'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M',  'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z',  'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm',  'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z',  '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '-', '\_' }; |
| --- |

And for user experience, the f(x) shall have the same length. For convenience, we can assume the length be 6, and the possible combination have: 6^64 = 6.334 \* 10^49, which is huge enough.

Based on the above, we can put forward a algorithm like:

| public String getShortenedUrl() {  StringBuilder shortenedUrl = new StringBuilder();  for (int i = 0; i < 6; i++) {  shortenedUrl.append(base64URL[random.nextInt() % 64]);  }  return shortenedUrl.toString(); } |
| --- |

## Database Model



*Hint: No-sql is also fine.*

## Frontend - Backend API

| public String createShortenedUrl(String originalUrl) {  // Generate shortened url.  // Create Shortened Entity(mapping: original url => shortened url).  // return the shortened url.  } |
| --- |

| public String getOriginalUrl(String shortenedUrl) {  // Get original url by the Primary Key: shortened url.  // return the original url.  } |
| --- |

## Frontend UI Desi

# Risks

* The performance of the Relationship database maybe not perform well under huge flow requests, we can consider using the Non-Sql(Redis, etc.)

# Revision History

| **Date** | **Reason for changes** |
| --- | --- |
| Jan 16, 2022, | Initiative. |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |