Connect Group - Architecture/Software Design Test - URL Shortener

Questions/considerations:

The first consideration is what the possible values for our URL slug should be. This requires some estimation of usage in terms of how many URLs we expect users to create. For example:

- Using only values 0-9, the maximum possible number of unique URL slugs we can generate is 1,000,000 (producing URL slugs from /000000 to /999999).

- Using base 36 (values [0-9a-z]) the number of possible URL slug combinations is significantly higher at 2,176,782,335.

- Using base 62 (values [0-9a-zA-Z]) the number is 56,800,235,583

- It's worth noting that other special character combinations are allowed in URLs, namely $-_.,+!*'() and that using these characters increases the number of possible slug combinations still further.

It seems unlikely we would ever need more than the number of possible values offered by base 36 (that's equivalent to creating 120,000 links per day, every day for 50 years), and the benefit of choosing this over base 62+ is that it eliminates the requirement for case-sensitivity or special characters in URL slugs, meaning they are easier for users to enter. For the sake of the test, we'll assume the client wants the additional capacity of base 62.

Another consideration is how long links will persist in the system. Once created, do they remain indefinitely, or do they expire, and if they expire, under what criteria? The easiest method would be to have URLs expire after a set period of time, however this is likely to result in the least user friendly experience and will certainly result in some popular and frequently used URLs expiring. The alternative is to add a date last visited field to to the URL table in the database. We can then run a scheduled clean-up to remove any URLs not visited since X time (we may also want to track volume, e.g. URLs with below Y number of visitors since X time, to root out URLs that are likely only being visited by automated link crawlers).

The benefit of links expiring is that it reduces the number of redundant links being stored in the system and potentially frees up more URL combinations for future use. However, the disadvantages (less frequently visited links not working after a time or, worse, suddenly being repurposed to direct traffic to a different URL) likely outweigh the advantages and as such I would recommend we do not allow URLs to expire.

A big consideration is security. For the purposes of the exercise, I've assumed quite a dumb system that allows anyone to create as many URLs as they like without even checking said URLs are valid. In reality, that's wide open to abuse (a user could create a bot that fills our entire database with fake URLs, rendering the system useless). We should consider:

- Whether we have users create an account and use, at the very least, email authentication to be able to limit how many URLs they can create within a set period of time.

- Alternatively, if we want the freedom of anyone being able to create a URL, what other means are available to limit abuse (e.g. logging their IP to prevent too many new URLs within a short period of time).

- Whether we want to check URLs to ensure they point to valid, live pages. Of course this may not actually be desirable - users may want to create a short link in advance of a page/website going live (so that they can have it printed on promotional materials in advance, for instance).

- We would want to prevent users creating circular links within the system by checking they are not trying to create a link that points back to the client's service.

Another consideration is how redirects are handled. If handled incorrectly, a redirect can cause issues with search engine optimisation and usability (e.g. by writing additional links to the user's history, making it difficult for them to go "back" in the browser without hitting the redirect again). Google's recommendation is to use a server-side 301 permanent redirect.

Server setup/costs are an important consideration. This is not something I have dealt with directly as a front end developer before and therefore I have to admit this is outside my area of expertise. This is the kind of question which, given more time, I would have to liaise with the supplier directly to answer. We are fortunate that we have some idea from the client of daily visitors and peak traffic.

The final consideration is analytics. In order to understand how popular the service is, it would be useful to track how often shortened links are being followed. We could either have our application handle this (by recording that information to the database whenever a link is followed) or, since everything is based on the URL slug, we could potentially extrapolate from the server logs. The former is probably the simpler and more flexible solution, although it does require writing to/store more in the database. For the sake of the test I have assumed no analytics.

The Application

Based on the assumptions made above, our application requires classes for the following:

General classes:

- To handle database connections/errors.
- To close database connections.
- To convert base 10 values to base 62 - this generates our shortened URL slug. Note, if it has to be exactly 6 characters (rather than 6 characters maximum) we should pad the value.
- To convert base 62 values to base 10.

Classes used in the creation of new URLs:

- To use the database handler to check the long form URL entered does not already exist. If it does, return the index, pass to the base 10 to 62 converter to generate our URL slug and display to the user.
- To insert a new record to the database using database handler, returning the index for the record, which is passed to the base 10 to 62 converter to generate our URL slug.
- To display the generated URL (or error message) to the user.

Classes used to un-shorten URLs and redirect users to the original URL:

- To accept URL slug from querystring and pass to base 62 to 10 converter class (without the + if the user is just viewing the URL) to return a searchable database index.
- To use the database handler to look up our searchable database index and return the original long form URL.
- To perform a 301 redirect to the original long form URL, or;
- To display the original long form URL to the user if the user entered a slug with a + at the end (or an error if, e.g., the URI could not be found).