Assignment – URL Shortener

Introduction

I think we can all agree that there aren't enough URL shorteners in the world, so the goal of this assignment is to create a URL shortener service prototype. Now, let's describe what we're looking for in this assignment, we have two parts to it: the prototype part where you actually will write some code, and the "what if" part where you would just explain how you'd tackle this at scale.

The minimum requirements are:

- It compiles and runs
- It has a cool name
- It does what it is supposed to do
- Code should be covered by tests
- Add docker compose file to run Database instance and Redis
- It should be: java 17 / Spring boot / Spring data / Spring MVC
- It should be an HTTP REST-based service.
- If should use Mongodb and Redis
- It should be organized for easy compilation and packaging with maven
- Host it on github and provide access for review

Part 1 - Prototype (Code)

It should be a REST API web service. You can decide what endpoints in the API would be. We'll just describe in words what this thing is supposed to do and you can take it from there. If you're not sure about something, make an assumption and let us know what it is. Project source code should follow common software engineering practices. Pay attention to: software design (code structure), code formatting, test coverage, error handling, performance and logging.

- It should do two things:
    - You can submit a URL and get back a "shortened" version. E.g. submit "http://www.xplace.com" and get back "http://xpl.ac/abcdef".
        - The format of the URL returned is up to you - this is just an example.
        - The shortened URLs should resolve to only one source URL.
    - You can resolve a shortened URL to get back the original URL. E.g. submit "http://xpl.ac/abcdef" and get back "http://www.xplace.com".

Part 2 - What If

Having done the work above to create a prototype of the URL shortener service, let's think about this at a larger scale. What if this service needed to scale to 10,000 URL generation requests per second? How about 100,000 URL resolve requests per second? Describe how you'd actually architect a system like this. Specifically, how would the URL generation work at scale (and what generation method you'd use) and how the URL resolving would work at scale. How would you store the data? How long would you keep it around?

Please write up a short paragraph or two describing the above. We're not looking for a design document, but a short and sweet explanation you'd give to a team mate at the coffee machine.

Done!

Once those two are done, send over your code from part 1 and text from part 2. Thanks for taking the time to do this and showing interest!