Candidate: **Brendon Dulam**
Contact: 080 7595 0143 / brendondulam06@gmail.com
Links: https://github.com/bldulam1 https://bdulam.netlify.com/

Programming

This document and the code to the solutions of the programming questions are available in the following repository:

https://github.com/bldulam1/veltra-challenge

Question 1

Write a function to determine if a string is a "Palindrome" ignoring spaces, punctuation and

case-sensitivity.

Examples input and output would be:

Func X("Hello World") returns "Not a palindrome"

Func X("A Toyota's a Toyota") returns "A palindrome"

Solution (GO/Golang)

- Time/Space Complexity

  The optimum solution of this kind of problem is O(n) and the following solution also has time complexity of O(n). It iterates through all the characters until a falsifying case is encountered

  The space complexity is O(1), it does not use additional memory whenever n increases

- Assumptions

  Palindromes contain alphanumeric characters only

  Words containing no alphanumeric characters are not palindromes

  Words containing single alphanumeric character is palindrome

```
package problem1

func isAlphaNumeric(r rune) bool {
  return ('a' <= r && r <= 'z') || ('A' <= r && r <= 'Z') || ('0' <= r && r <= '9')
}

func toLower(r rune) rune {
  if 'A' <= r && r <= 'Z' {
    return r + ('a' - 'A')
  }
  return r
}

func isPalindrome(s string) bool {
  if len(s) == 0 {
    return false
  } else if len(s) == 1 {
    return isAlphaNumeric(rune(s[0]))
  }

  leftPtr, rightPtr := 0, len(s)-1
  isContainAlphaNumeric := false

  for leftPtr < rightPtr {
```

```
    for leftPtr < rightPtr && !isAlphaNumeric(rune(s[leftPtr])) {
      leftPtr++
    }
    for leftPtr < rightPtr && !isAlphaNumeric(rune(s[rightPtr])) {
      rightPtr--
    }
    if !isContainAlphaNumeric && (isAlphaNumeric(rune(s[leftPtr])) ||
        isAlphaNumeric(rune(s[rightPtr]))) {
      isContainAlphaNumeric = true
    }

    leftChar, rightChar := toLower(rune(s[leftPtr])), toLower(rune(s[rightPtr]))
    if leftChar != rightChar {
      return false
    }

    leftPtr++
    rightPtr--
  }

  return isContainAlphaNumeric
}

func CheckPalindrome(s string) string {
  if isPalindrome(s) {
    return "A palindrome"
  }
  return "Not a palindrome"
}
```

Driver / Test function

```
package main

import (
  "fmt"
  "veltra-challenge/problem1"
)

func main() {
  words := []string{
    "",
    "?",
    "0",
    "A",
    "AB",
    "11",
    "11a",
    "113",
    "Hello World",
    "A Toyota's a Toyota",
    "Nurses run",
  }
```

```
  for _, word := range words {
    fmt.Println(problem1.CheckPalindrome(word), ":", word)
  }
}
```

Test Results

```
Not a palindrome :
Not a palindrome : ?
A palindrome : 0
A palindrome : A
Not a palindrome : AB
A palindrome : 11
Not a palindrome : 11a
Not a palindrome : 113
Not a palindrome : Hello World
A palindrome : A Toyota's a Toyota
A palindrome : Nurses run
```

Question 2

Write a function to determine if a number representing a Gregorian Year is a leap year.

Examples input and output would be:

Func Y(2019) returns false

Func Y(2020) returns true

Solution (GO/Golang)

- Time/Space Complexity

  The time and space complexities are both O(1)

- Assumptions

  Leap Year definition from Wikipedia (https://en.wikipedia.org/wiki/Leap_year):

  - Every year that is exactly divisible by four is a leap year,
  - except for years that are exactly divisible by 100,
  - but these centurial years are leap years if they are exactly divisible by 400.

```
package problem2

/*
  leap year (definition)
  a year, occurring once every four years, which has 366 days including 29 February as an
intercalary day.

  reference: https://en.wikipedia.org/wiki/Leap_year
  Every year that is exactly divisible by four is a leap year,
  except for years that are exactly divisible by 100,
  but these centurial years are leap years if they are exactly divisible by 400.

  For example, the years 1700, 1800, and 1900 are not leap years, but the years 1600 and 2
000 are.
*/
```

```go
func isDivisible(num int, divisor int) bool {
  if num%divisor == 0 {
    return true
  }
  return false
}

func IsLeapYear(year int) bool {
  if year < 0 {
    return false
  } else if isDivisible(year, 400) {
    return true
  } else if isDivisible(year, 100) {
    return false
  } else if isDivisible(year, 4) {
    return true
  }
  return false
}
```

Driver/Test Function

```go
package main

import (
  "fmt"
  "veltra-challenge/problem2"
)

func main() {
  years := []int{
    -1, 0, 4, 10, 20, 100, 1700, 1800, 1900, 1600, 2000, 2019, 2020,
  }

  for _, year := range years {
    fmt.Println(year, ":", problem2.IsLeapYear(year))
  }
}
```

Test Results

```
-1 : false
0 : true
4 : true
10 : false
20 : true
100 : false
1700 : false
1800 : false
1900 : false
1600 : true
2000 : true
2019 : false
2020 : true
```

Design Patterns

Question 1

URL Shortening Service Application

Introduction

This is a system design for a URL shortening service. Basically, the service transforms a long url string into a unique, fixed length (normally 6 to 8 characters) URL.

The system can provide to the user the following services:

- Create a shorter URL (POST request)
    - Input
        - Long URL
        - UserID (optional, for protected URL)
        - Expiration date (optional)
    - Output
        - Short URL (URL ID)
- Retrieve the original (long) URL using a short URL, then redirect to the original URL. (GET request)
    - Input
        - Short URL (URL ID)
        - UserID (for protected URLs)

Income Generating Services

In order that this service can generate income, it would be good to include a service for analyzing the most frequently visited urls. Having this information allows the the application to inform other parties/users of the most frequently visited sites. The following is the service for fetching URL statistics

- Fetch Statistics of a Base URL
    - Input
        - Long URL
        - API_KEY for developers
    - Output
        - URL statistics

Framework Choices

String Shortening Algorithm

Most string URL shortening services rely on two popular hashing algorithms—MD5 and SHA256.

MD5 has base36, covering [a-z ,0-9], base62 ([A-Z, a-z, 0-9]), and base64 ([A-Z, a-z, 0-9, +, /]) options. The length of the short URL needs to be as short as possible and can map as many unique urls as possible. Thus, a base64/base62 is suitable for this application.

SHA256 on the other hand has slower execution time compared to MD5 based on this Stackoverflow thread: https://bit.ly/3900Bcm. All major programming language has MD5 libraries, so this system will be using MD5 hashing, with base62 and at least 6 characters in length.

Programming Language Frameworks

There are many frameworks that can be used to implement this system. I will be talking about the frameworks that I am very familiar.

Express/Serverless (NodeJS)

This framework is easy to develop. It is one of the most popular frameworks nowadays. It has a lot of ready-made packages that can be used for other smaller tasks like reading and updating a database, and other data manipulation operations. Another advantage of this framework is that it can be easily deployed to cloud services like AWS, GCP, Azure, etc.

NodeJS has some disadvantages like being not able to perform multithreading, and not suitable for long running operations. But these disadvantages does not really affect the kind of application we are building.

Echo (GO)

GO has several popular frameworks like Gin, Gorilla Mux, but I find the Echo framework/library to be more developer-friendly. Go is famous for its speed being a compiled language, and its support for concurrency operations. The execution time of applications written in GO could be faster than NodeJS. Its syntax is also easy to learn.

Go is statically typed, it has less flexibility compared to dynamically typed languages. It has fewer third party libraries than NodeJS.

Flask and Django (Python)

Python's Flask and Django are also viable frameworks for this application. Being written in Python, makes these frameworks easy to understand and makes the project fast to develop. Python is also supported in major cloud services.

The execution time of Python applications could be slower than applications written in GO.

I will be implementing the backend infrastructure using serverless or lamda services from famous cloud providers like AWS, Netlify, GCP or Azure.

Deployment and infrastructure Choices

The reason why I want to use serverless or lambda services (Function as a Service) is that it is scalable. It can easily handle instances when the usage of the application is high and automatically reduces its resources when the usage is low. Thus, we only pay for the exact computing resources that the application used. Also, the management of the servers during the fluctuation of the network traffic is taken care by the cloud provider.

One disadvantage of using lambda services is that it takes time for cold-starting. Lambda functions are good for short-running short-lived operations. Using Lambda operation requires Database as a service as its partner. These disadvantages does not hurt much the kind of application we are building.

Other solutions for the infrastructure is using traditional physical servers. These servers could be cost-saving if the application has constant high user demand and the operation of functions is heavy time/resource consuming. It also has some disadvantages like the you will be paying for the servers that has been contracted even if the user demand is low. Also the management of the servers during the user demand fluctuations should be taken care by the company.

For this application, I think the lambda services would be the most cost-effective and easier to maintain. Deployments will have to be passed through CI/CD pipeline. CI/CD is normally included inside the cloud platform (like Netlify's CI/CD).

Backend/ Preferred Communication Tools

For the client side, RESTful communication would be the suitable form of communication. RESTful APIs are easy to understand, and it already has been adapted to a lot of real life applications.

For the developers, GraphQL is also a viable option. It makes the documentation of APIs self-documenting. For the developers, there could be more information that will be fetched from the database like the statistics of the urls (number of times accessed, demography of who used the url, time of access, etc). Fetching customized set of data is easily done using GraphQL.

A mixture of GraphQL and RESTful APIs will be used as communication among the services and clients.

Issues to consider

Some major issues of the kind of application we are building are availability and scalability of the application, and security for the kind of data that is sent by users (some URLs might include API_KEY of the user for some apps).

Availability and Scalability

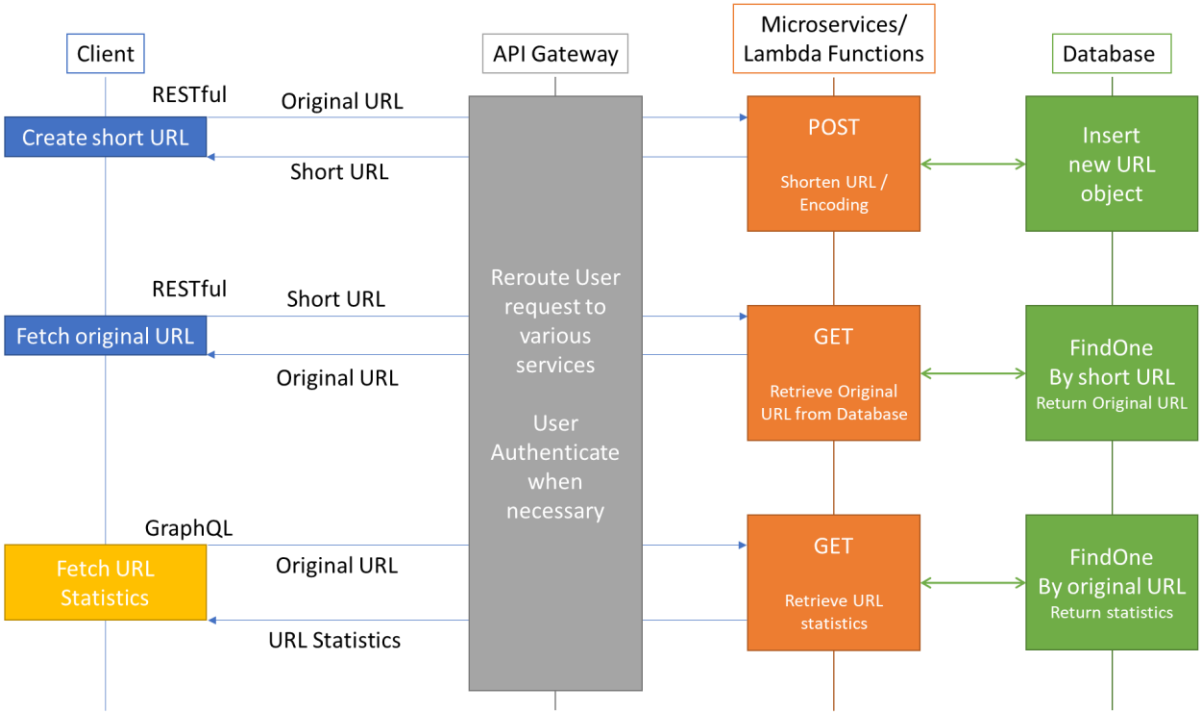
During the initial launch of the product, the system will be using lambda services. Scalability and availability will be taken care by the cloud provider.

Security

User authentication might be required for confidential URL shortening purposes. Some URLs might be restricted to some group of people or individuals. So this will be taken care by having an authentication feature. User activities can also be tracked having a separate services for url shortening-related user activities.

System Design Diagram

The following diagram shows the schematic design of the entire system. The microservices/lambda functions portion is the dynamically adjusting component of the design. It automatically increases and decreases its resources based from the demand of the clients.



Database Design

A NoSQL database would be best in implementing the database of this application.

Below are the schema of the url and user collections

URL

_id	ObjectID
originalURL	String
shortURL	String
creationDate	DateTime
expirationDate	Datetime
Owner	ObjectID
accessFrequency	Integer

User

_id	ObjectID
Email	String
Name	String
registrationDate	DateTime
lastLogin	DateTime
API_KEY	String

Question 2

Distributed System

Please describe at a simple, high level what a distributed system?

- A distributed system is a system using multiple computers/servers/microservices in dividing a big chunk of computational operation to speed up the entire operation. The smaller tasks/jobs may be sequential or parallel tasks that form the big chunk of operation.
- For example, a big task could be labelling or annotation of a video file using Machine Learning.
    - A video file can be divided into several frames/images. Then queued using a database for server assignment and status monitoring
    - The images/video frames are assigned to multiple servers for labelling, which can label the images in parallel
    - Each server may take multiple images at a time until all images from the queue are labelled
    - Using this system, the execution time of video labelling is reduced by the number of servers working in parallel

Provide some difficulties that need to be overcome when working with such systems. Provide some possible solutions for said problems.

- **Accidental failures of servers**
  This can be mitigated by having a polling mechanism from the main server to the worker servers to check the status of the worker servers. Whenever the server fails, the job taken by that server is automatically put back to the jobs queue so that other servers can take that job. A notification will be sent to the developer informing the server failure and further steps will be taken to solve the failure.
- **Scheduling for Dequeuing Queued jobs**
  When some jobs run in parallel and some jobs run sequentially, having dependencies from other jobs, a scheduling is necessary. A linked List data structure could be a solution for sequential jobs.
- **Concurrent access to the database**
  Having servers working in parallel requires handling of concurrent requests to the database. This requires ACID (atomicity, consistency, isolation, durability) properties of a database to maintain the credibility of the data from the database.