UNIVERSIDAD DE SAN BUENAVENTURA CALI

FACULTAD DE INGENIERIAS - PROGRAMA DE ING. SISTEMAS

TECNICAS AVANZADAS DE PROGRAMACION

Final Project

Presentado por: Sebastian Lopez Montenegro

Código estudiantil: 30000097500

Correo institucional:
Slopezm4@correo.usbcali.edu.co

Fecha: 03/06/2024

Problema: El proyecto de fin de curso busca poner en práctica los temas y las habilidades discutidas a lo largo del semestre. De este modo, busca incrementar las capacidades de diseño, implementación y comunicación de los estudiantes, en torno al desarrollo de un sistema de software, y la documentación de los procesos de pruebas seguidas.

La Figura 1 esboza un diagrama de la arquitectura general del sistema deseado.

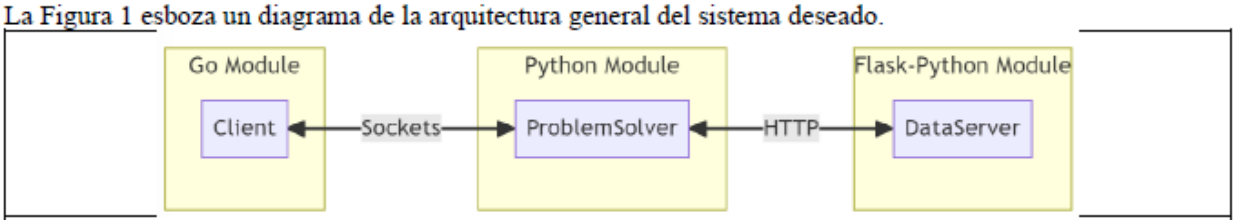


Fig. 1 Vista arquitectural de alto nivel del sistema.

Restricciones: La actividad debe ser resuelta mediante la metodología TDD, especificando cada etapa de esta metodología.

Restricciones Client:

Requerimientos funcionales

- Ofrece una interacción/interfaz en modo CLI para la selección del problema a resolver, y la cantidad de valores a procesar.
- Imprime y/o salva a disco los resultados obtenidos para el problema seleccionado, acorde con solicitud del usuario.
- Permite hacer la petición de apagado (i.e., shutdown) de todo el sistema, empleando algún mecanismo de autenticación.

Restricciones tecnológicas

- Está implementado en Golang.
- Interopera mediante sockets con ProblemSolver, empleando formato JSON para intercambio de datos.

Restricciones ProblemSolver:

Requerimientos funcionales

- Consume datos de DataServer para ser empleados en el problema a resolver.
- Resuelve el problema solicitado por Client.
- Lleva un log de las operaciones realizadas.

Restricciones tecnológicas

- Está implementado en Python.
- Interopera mediante HTTP con DataServer, empleando formato JSON para intercambio de datos.
- Emplea de manera explícita y coordinada algún mecanismo que brinde confidencialidad a los datos en tránsito con DataServer (independiente del protocolo de transmisión).
- Aplica patrones de diseño creacionales en la instanciación del problema a resolver.

Restricciones DataServer:

Requerimientos funcionales

- Genera de manera aleatoria y empleando diferentes distribuciones de probabilidad una cantidad específica de números enteros positivos en un rango también aleatorio.
- Sirve los datos generados en respuesta a una petición HTTP, empleando el formato JSON.
- Resuelve el problema solicitado por Client.
- Lleva un log de las operaciones realizadas.

Restricciones tecnológicas

- Está implementado en Flask-Python.
- Emplea de manera explícita y coordinada algún mecanismo que brinde confidencialidad a los datos en tránsito con ProblemSolver (independiente del protocolo de transmisión).
- Aplica inversión de dependencias en la generación de los números aleatorios.
- Está en la capacidad de emplear las distribuciones normal y uniforme.

Estrategias: Para la elaboración de esta actividad se necesitará la herramienta Postman, y archivos bash(scripts) de comparación como parte de la estructura de pruebas. También se necesitará la librería Flask y Numpy para el manejo de los números enviados desde DataServer generados por Numpy, y Crypto para trabajar la encriptación de mensajes en la comunicación de

DataServer y ProblemSolver. Para Client (trabajado en Go) se usará el framework Cobra para cumplir con el requisito de interfaz CLI.

Fase RED

Client:

Para el iniciar con la construcción del archivo Go, se buscó inicialmente la documentación de Cobra. Y se encontró que la implementación de la interfaz CLI se puede hacer fácilmente con el uso de “Flags”.

README Apache-2.0 license

Flags

A flag is a way to modify the behavior of a command. Cobra supports fully POSIX-compliant flags as well as the Go flag package. A Cobra command can define flags that persist through to children commands and flags that are only available to that command.

In the example above, 'port' is the flag.

Flag functionality is provided by the pflag library, a fork of the flag standard library which maintains the same interface while adding POSIX compliance.

Installing

Using Cobra is easy. First, use `go get` to install the latest version of the library.

```
go get -u github.com/spf13/cobra@latest
```

Next, include Cobra in your application:

```
import "github.com/spf13/cobra"
```

Usage

`cobra-cli` is a command line program to generate cobra applications and command files. It will bootstrap your application scaffolding to rapidly develop a Cobra-based application. It is the easiest way to incorporate Cobra into your application.

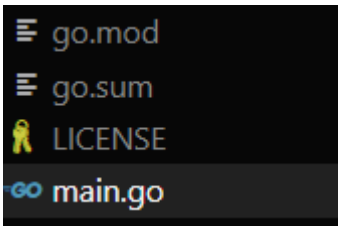
It can be installed by running:

```
go install github.com/spf13/cobra-cli@latest
```

For complete details on using the Cobra-CLI generator, please read The Cobra Generator README

For complete details on using the Cobra library, please read the The Cobra User Guide.

Al instalar Go y Cobra en el entorno virtual se generaron archivos automáticamente para el uso de la librería y se generaron dos archivos que son los necesarios para el uso de Cobra que son el main.go (que es desde donde corre el programa) y el archivo root.go que es donde se implementa el código correspondiente a Cobra.



Client > main.go > ...

```go
/*
Copyright © 2024 NAME HERE <EMAIL ADDRESS>
*/
package main

import "github.com/Sxl07/Client/cmd"

func main() {
    cmd.Execute()
}
```

```go
// rootCmd represents the base command when called without any subcommands
var rootCmd = &cobra.Command{
    Use:   "Client",
    Short: "A brief description of your application",
    Long: `A longer description that spans multiple lines and likely contains
examples and usage of using your application. For example:

Cobra is a CLI library for Go that empowers applications.
This application is a tool to generate the needed files
to quickly create a Cobra application.`,
    // Uncomment the following line if your bare application
    // has an action associated with it:
```

En el archivo root.go se explica como es el uso de las Flags incluyendo un ejemplo.

```go
// Execute adds all child commands to the root command and sets flags appropriately.
// This is called by main.main(). It only needs to happen once to the rootCmd.
func Execute() {
    err := rootCmd.Execute()
    if err ≠ nil {
        os.Exit(1)
    }
}

func init() {
    // Here you will define your flags and configuration settings.
    // Cobra supports persistent flags, which, if defined here,
    // will be global for your application.

    // rootCmd.PersistentFlags().StringVar(&cfgFile, "config", "", "config file (default is $HOME/.Client.yaml)")

    // Cobra also supports local flags, which will only run
    // when this action is called directly.
    rootCmd.Flags().BoolP("toggle", "t", false, "Help message for toggle")
```

Se crearon las flags en el init() del archivo root.go

```go
func init() {
    // Here you will define your flags and configuration settings.
    // Cobra supports persistent flags, which, if defined here,
    // will be global for your application.

    // rootCmd.PersistentFlags().StringVar(&cfgFile, "config", "", "config file (default is $HOME/.Client.yaml)")

    // Cobra also supports local flags, which will only run
    // when this action is called directly.
    rootCmd.Flags().BoolP("toggle", "t", false, "Help message for toggle")
    rootCmd.Flags().StringVarP(&problem, "problem", "p", "fizzbuzz", "Problem to be resolved fizzbuzz|fibonacci|prime")
    rootCmd.Flags().IntVarP(&cant, "quantity", "q", 25, "quantity of numbers")
    rootCmd.Flags().IntVarP(&min, "lowlimit", "l", 0, "minimum number of the range")
    rootCmd.Flags().IntVarP(&max, "suplimit", "s", 100, "maximum number of the range")
    rootCmd.Flags().StringVarP(&result, "result", "r", "", "save results in results.txt file, if you want to save result
}
```

Se añade en el comando root las variables definidas las flags. Y se creó un struct (que se convierte en JSON) para enviarlo después a ProblemSolver con los valores ingresados en el CLI.

```go
type Requests struct {
    Problem  string
    Cant     int
    Min      int
    Max      int
    Result   string
    Shutdown int
    Password string
}

var problem string
var cant int
var min int
var max int
var result string

// rootCmd represents the base command when called without any subcommands
var rootCmd = &cobra.Command{
    Use:   "Client",
    Short: "A brief description of your application",
    Long: `A longer description that spans multiple lines and likely contains
examples and usage of using your application. For example:

Cobra is a CLI library for Go that empowers applications.
This application is a tool to generate the needed files
to quickly create a Cobra application.`,
    // Uncomment the following line if your bare application
    // has an action associated with it:
    Run: func(cmd *cobra.Command, args []string) {
        request := Requests{
            Problem:  problem,
            Cant:     cant,
            Min:      min,
            Max:      max,
            Result:   result,
```

Revisando el funcionamiento de Cobra, para implementar el uso del shutdown requerido se debe usar un subcomando para el comando rootCmd. Por lo que se creará un comando shutdownCmd en el archivo root.go y se agregará como subcomando de rootCmd como se muestra a continuación.

```go
func init() {
    rootCmd.AddCommand(shutdownCmd)

    // Here you will define your flags and configuration settings.

    // Cobra supports Persistent Flags which will work for this command
    // and all subcommands, e.g.:
    // shutdownCmd.PersistentFlags().String("foo", "", "A help for foo")

    // Cobra supports local flags which will only run when this command
    // is called directly, e.g.:
    // shutdownCmd.Flags().BoolP("toggle", "t", false, "Help message for toggle")
    shutdownCmd.Flags().StringVarP(&password, "password", "p", "", "the password to authenticate shutdown,use -p 'password'")
}
```

Se creó una carpeta functions con el archivo socket.go para el uso del mismo socket, y esto conlleva el envío y recepción de datos enviados por el mismo socket. Este archivo será usado desde el archivo root.go.

```go
type Response struct {
    Result []string `json:"result"`
}

func OnServer(port string) (net.Conn, error) {
    conn, err := net.Dial("tcp", port)
    if err ≠ nil {
        return nil, fmt.Errorf("error connecting to server")
    }
    return conn, nil
}

func SendData(data []byte, conn net.Conn) error {
    _, err := conn.Write(data)
    if err ≠ nil {
        return fmt.Errorf("error sendind data")
    }
    fmt.Println("Data sent to the server:", string(data))
    return nil
}

func DataResponse(conn net.Conn) (*Response, error) {
    var response Response
    err := json.NewDecoder(conn).Decode(&response)
    if err ≠ nil {
        return nil, fmt.Errorf("error decoding the JSON response: %v", err)
    }
    return &response, nil
}
```

En la captura anterior se muestran las funciones de socket.go. OnServer() sirve para realizar la conexión con el socket que se defina en ProblemSolver. SendData() sirve para enviar los

datos mediante el socket. DataResponse() sirve para recibir los datos (en formato JSON) que responda ProblemSolver por el socket.

```go
var rootCmd = &cobra.Command{
    Run: func(cmd *cobra.Command, args []string) {
        request := Requests{
            Problem:  problem,
            Cant:     cant,
            Min:      min,
            Max:      max,
            Result:   result,
            Shutdown: 0,
            Password: "",
        }

        port := "localhost:8080"
        conn, err := functions.OnServer(port)
        if err ≠ nil {
            fmt.Println("Error:", err)
        }

        jsonData, err := json.Marshal(request)
        if err ≠ nil {
            fmt.Println("error:", err)
        }

        bad := false
        if problem == "" {
            fmt.Println("Problem cannot be empty")
            bad = true
        }
        if cant ≤ 1 {
            fmt.Println("Quantity cannot be less than 1")
            bad = true
        }
        if min < 0 {
            fmt.Println("Low limit cannot be negative")
            bad = true
        }
        if max < min {
            fmt.Println("Superior limit cannot be less than low limit")
            bad = true
        }
```

Desde el archivo root.go se empezó a definir el flujo del sistema con respecto a los valores ingresados desde los Flags, iniciando la conexión con el socket por el puerto local 8080 usando OnServer(). Luego se convierte el struct request a JSON para enviarlo mediante el socket los datos. Sin embargo, antes de enviarlo se controlan los posibles errores que se ingresen en las variables de las Flags, por ejemplo, que el problema esté vacío, que la cantidad de datos a enviar no sean menores a cero, y que los límites de del rango sean

válidos. Si alguno de estos errores se presenta, se define a la variable bad como true por lo que el envío con los datos no se efectúa. Cabe recalcar que, si el problema no existe, este se verificará en ProblemSolver y se manejará el problema desde problemSolver.

```go
if !bad {
    err = functions.SendData(jsonData, conn)
    if err ≠ nil {
        fmt.Println("Error:", err)
    }

    response, err := functions.DataResponse(conn)
    if err ≠ nil {
        fmt.Println(err)
    }
    fmt.Println(response.Result)

    if result == "y" || result == "Y" {
        fileName := "results.txt"

        content := strings.Join(response.Result, "\n")

        file, err := os.OpenFile(fileName, os.O_WRONLY|os.O_CREATE|os.O_TRUNC, 0644)
        if err ≠ nil {
            fmt.Println("Error opening the file:", err)
        }
        defer file.Close()

        _, err = file.WriteString(content)
        if err ≠ nil {
            fmt.Println("Error writing in the file:", err)
        }
        fmt.Println("File results.txt Created.")
        fmt.Println("Your results are on the results.txt file.")
    }
}
conn.Close()
```

Si todo está bien, se enviarán los datos mediante el socket y se esperará la respuesta del socket, como ya se calculan los datos, ahora se verifica si el usuario quiere que se guarden los datos en un archivo (el archivo se dejó predeterminado como results.txt) y, si no los datos se imprimen por consola.

```go
var shutdownCmd = &cobra.Command{
    Run: func(cmd *cobra.Command, args []string) {
        if password == "" {
            fmt.Println("a password is require use -p 'password'")
        }
        filename := "key.txt"

        secretKey, err := ioutil.ReadFile(filename)
        if err ≠ nil {
            log.Fatalf("Error reading file: %v", err)
        }

        h := hmac.New(sha256.New, []byte(secretKey))
        h.Write([]byte(password))
        hash := h.Sum(nil)
        hashPassword := fmt.Sprintf("%x", hash)

        request := Requests{
            Problem:  "",
            Cant:     0,
            Min:      0,
            Max:      0,
            Result:   "",
            Shutdown: 1,
            Password: hashPassword,
        }
```

```go
    port := "localhost:8080"
    conn, err := functions.OnServer(port)
    if err ≠ nil {
        fmt.Println("Error:", err)
    }

    jsonData, err := json.Marshal(request)
    if err ≠ nil {
        log.Fatalf("error: %v", err)
    }

    err = functions.SendData(jsonData, conn)
    if err ≠ nil {
        fmt.Println("Error:", err)
    }

    response, err := functions.DataResponse(conn)
    if err ≠ nil {
        fmt.Println(err)
    }
    fmt.Println(response.Result)
}
```

En el caso del comando shutdown, se les da a las variables shutdown y password que es hasheada mediante SHA256 y una llave que está almacenada en key.txt (requerida para realizar definitivamente el shutdown) del struct request los valores que necesite para hacer el proceso de shutdown en ProblemSolver y DataServer. Este lo realiza de la misma manera que en el comando root, estableciendo la conexión con el socket y desde este enviar el struct en forma de JSON para que ProblemSolver se encargue del apagado. Ya que, según lo discutido en la retroalimentación del día 31 de mayo se aclaró que el ProblemSolver se encarga de verificar la contraseña y por ende desde ahí se va a cerrar la conexión con Data server y la conexión del socket con Client. El mensaje de la conexión terminada con el socket se enviará desde ProblemSolver a Client nates de cerrar el socket desde ProblemSolver, para mostrar en Client cual fue la respuesta de los datos enviados.

Pruebas

Para probar el correcto funcionamiento se acomodó en el archivo test.sh los parámetros necesarios para probar que se imprime todo correctamente, se usó un archivo Python momentáneo para probar que funciona el socket y que se imprime todo correctamente. Esto se hizo con un solo valor para probar bien, ya que el archivo momentáneo de Python lo puse a que generará valores aleatorios, entonces se imprimirá solo el rango de 0 a 0, es decir, solo 0.

```
ent > $ test.sh
#!/bin/bash

YELLOW='\033[0;33m'
NC='\033[0m' # No Color

# Assigning default file names
command="go run .\main.go -q 1 -r y -l 0 -s 0 " #If the code is not compiled, use here the interpreter
main_module="main.go"
expected_output="expected_output.txt"
generated_output="results.txt"
diff_output="diff_output.txt"

if [ $# -eq 1 ]; then
  main_module=$1
elif [ $# -eq 2 ]; then
  main_module=$1
```

Al ejecutar el test.sh y file_compare.sh se confirma que la impresión es correcta.

```
User@DESKTOP-390G6QC MINGW64 /d/CUARTO SEMESTRE/APT/APTFinalProject/Client
$ ./test.sh
Invoking go run .\main.go -q 1 -r y -l 0 -s 0 main.go
Error: unknown command "main.go" for "Client"
Run 'Client --help' for usage.
exit status 1
Test passed: Files expected_output.txt and results.txt are identical.
```

```
Client > results.txt
1    0 FizzBuzz
```

```
Client > expected_output.txt
1    0 FizzBuzz
```

Ya finalmente probando, se ve de buena forma la impresión y la escritura del archivo.

```
results.txt ×
Client > results.txt
68 68
67 67
66 Fizz
31 31
42 Fizz
87 Fizz
26 26
43 43
74 74
83 83
44 44
93 Fizz
33 Fizz
35 Buzz
6 Fizz
14 14
86 86
88 88
27 Fizz
67 67

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

PS D:\CUARTO SEMESTRE\APT\APTFinalProject\Client> go run .\main.go -q 20 -r y
Data sent to the server: {"Problem":"fizzbuzz","Cant":20,"Min":0,"Max":100,"Result":"y","Shutdown":0,"Password":""}
[68 68 67 67 66 Fizz 31 31 42 Fizz 87 Fizz 26 26 43 43 74 74 83 83 44 44 93 Fizz 33 Fizz 35 Buzz 6 Fizz 14 14 86 86 88 88 27 Fizz 67 67]
File results.txt Created.
Your results are on the results.txt file.
PS D:\CUARTO SEMESTRE\APT\APTFinalProject\Client>
```

También se probó momentáneamente si se envía de regreso el mensaje de sistema apagado cuando se hace uso del subcomando shutdown con la letra –p que usa el tag del password con la contraseña correcta para apagar. Cabe recalcar que la contraseña se hasheó y que cuando se vaya a confirmar en ProblemSolver se analizaran tanto la contraseña enviada como la correcta de manera hasheada.

```
PS D:\CUARTO SEMESTRE\APT\APTFinalProject\Client> go run .\main.go shutdown -p admin
Data sent to the server: {"Problem":"","Cant":0,"Min":0,"Max":0,"Result":"","Shutdown":1,"Password":"6d2df7bd5b23426af87b65f44af0a7243b37649a9fe312e29e65d21c2365f528"}
[System down]
PS D:\CUARTO SEMESTRE\APT\APTFinalProject\Client>
```

DataServer:

Para la implementación de los aleatorios en DataServer se mantiene el principio de inversión de dependencias creando la interfaz IRandomNumber que lo implementan las clases RandomNormalDistribution (donde se crea primero una lista de números aleatorios para aplicarle una distribución normal) y RandomUniformDistribution (que genera una lista de aleatorios donde todos los números tienen la misma probabilidad de ser escogidos) (ambas clases están usando la librería Numpy).

IRandomNumber:

```
class IRandomNumber(ABC):
    @abstractmethod
    def generateNumbers(self,cant,min,max):
        pass
```

RandomUniformDistribution:

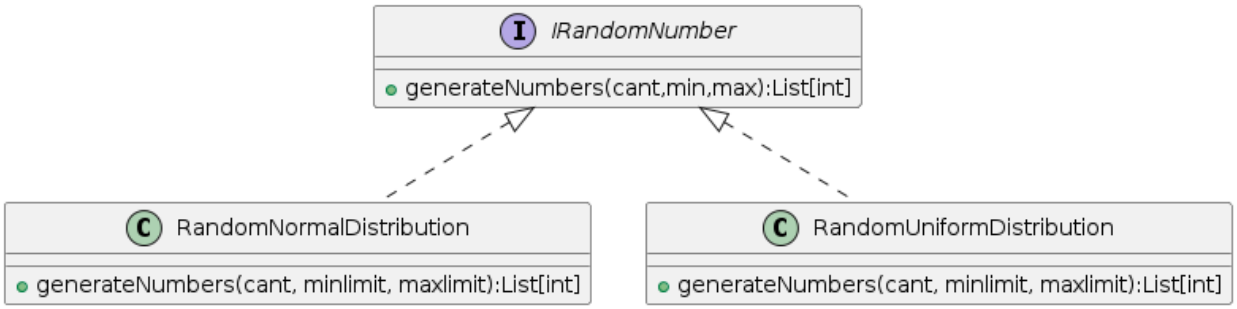
```
class RandomUniformDistribution(IRandomNumber):
    def generateNumbers(self, cant, minlimit, maxlimit):
        numbers = numpy.random.uniform(minlimit,maxlimit,cant).astype(int)
        return numbers.tolist()
```

RandomNormalDistribution:

```
class RandomNormalDistribution(IRandomNumber):
    def generateNumbers(self, cant, minlimit, maxlimit):
        numbers=numpy.random.uniform(minlimit,maxlimit,cant).astype(int)
        mean=numpy.mean(numbers)
        deviation=numpy.std(numbers)
        numbers_random = numpy.random.normal(mean,deviation,cant).astype(int)
        return numbers_random.tolist()
```

Se genera una lista mediante una distribución uniforme y luego se le saca promedio y la desviación para generar aleatoriamente con Numpy la lista de números con base a una distribución normal.

Diagrama de Clases Números aleatorios



RSAEncryption.py (Usado en DataServer y ProblemSolver):

Para la encriptación de los datos enviados por la url que conecta Problem Solver y DataServer se usó encriptación de llave asimétrica, sin embargo, se encontró que la mejor forma es hacerla hibrida entre RSA y AES, ya que al hacerse solamente con llave asimétrica no permite encriptar mensajes largos por el manejo de bytes, además de que entre más bytes use, más largo será el tiempo de encriptado y desencriptado.

```
class RSAEncryption:
    def __init__(self, key_name):
        self.private_key_path = f"{key_name}_private.pem"
        self.public_key_path = f"{key_name}_public.pem"
        self.private_key = None
        self.public_key = None

        if not os.path.exists(self.private_key_path) or not os.path.exists(self.public_key_path):
            self.generate_keys()
        else:
            self.load_keys()
```

En el constructor se llama la llave al instanciar el objeto y se crean las rutas donde se guardarán las llaves para encriptar y desencriptar (llave pública y privada, respectivamente), generando los archivos con las llaves si no se han generado.

```python
def generate_keys(self):
    key = RSA.generate(2048)
    self.private_key = key
    self.public_key = key.publickey()

    with open(self.private_key_path, 'wb') as private_file:
        private_file.write(self.private_key.export_key())

    with open(self.public_key_path, 'wb') as public_file:
        public_file.write(self.public_key.export_key())
```

Se generan las llaves RSA de 2048 y las guarda en las rutas que fueron definidas en el constructor.

```python
def load_keys(self):
    with open(self.private_key_path, 'rb') as private_file:
        self.private_key = RSA.import_key(private_file.read())

    with open(self.public_key_path, 'rb') as public_file:
        self.public_key = RSA.import_key(public_file.read())
```

Este método carga las claves RSA desde los archivos especificados.

```python
def encrypt_message(self, message, public_key_path):
    if isinstance(message, str):
        message = message.encode()

    with open(public_key_path, 'rb') as public_file:
        public_key = RSA.import_key(public_file.read())

    aes_key = get_random_bytes(16)
    cipher_aes = AES.new(aes_key, AES.MODE_EAX)
    ciphertext, tag = cipher_aes.encrypt_and_digest(message)

    cipher_rsa = PKCS1_OAEP.new(public_key)
    encrypted_aes_key = cipher_rsa.encrypt(aes_key)

    encrypted_data = {
        'aes_key': binascii.hexlify(encrypted_aes_key).decode('utf-8'),
        'nonce': binascii.hexlify(cipher_aes.nonce).decode('utf-8'),
        'tag': binascii.hexlify(tag).decode('utf-8'),
        'ciphertext': binascii.hexlify(ciphertext).decode('utf-8')
    }
    return json.dumps(encrypted_data)
```

Convierte el mensaje a bytes si es una cadena y cifra el mensaje usando AES en modo EAX, Luego cifra la clave AES usando RSA y por último, devuelve un JSON con la clave AES cifrada, el nonce, el tag y el mensaje cifrado, todos en formato hexadecimal.

```python
def decrypt_message(self, encrypted_message):
    encrypted_data = json.loads(encrypted_message)
    encrypted_aes_key = binascii.unhexlify(encrypted_data['aes_key'])
    nonce = binascii.unhexlify(encrypted_data['nonce'])
    tag = binascii.unhexlify(encrypted_data['tag'])
    ciphertext = binascii.unhexlify(encrypted_data['ciphertext'])

    cipher_rsa = PKCS1_OAEP.new(self.private_key)
    aes_key = cipher_rsa.decrypt(encrypted_aes_key)

    cipher_aes = AES.new(aes_key, AES.MODE_EAX, nonce=nonce)
    decrypted_message = cipher_aes.decrypt_and_verify(ciphertext, tag)

    return decrypted_message.decode('utf-8')
```

Se decodifica el mensaje cifrado de JSON a diccionario y convierte las partes cifradas de hexadecimal a bytes para descifrar la clave AES usando RSA. Luego se descifra el mensaje usando AES en modo EAX con la clave AES descifrada y el nonce para finalmente devolver el mensaje descifrado como una cadena.

Dataserver.py:

```
receiver=RSAEncryption('receiver')
receiver.generate_keys()
transmitter_key = None
@app.route('/key', methods=['POST'])
def key():
    request_data = request.json
    transmitter_key=request_data['public_key']
    with open("transmitter_key.pem","w") as file:
        file.write(transmitter_key)
    receiver_key=receiver.to_json()
    return receiver_key
```

Cuando se inicia el servidor de Flask se define un objeto de tipo RSA con el nombre de “receiver”, ya que dataserver será el receptor en todo momento de lo que le mande ProblemSolver. Este se instancia globalmente para usarse en la ruta /key que será usada por ProblemSolver mediante una petición POST para hacer el intercambio de llaves públicas (recordando que Problem Solver también usa RSAEncryption) y también para usarse en la ruta /numbers como se muestra a continuación.

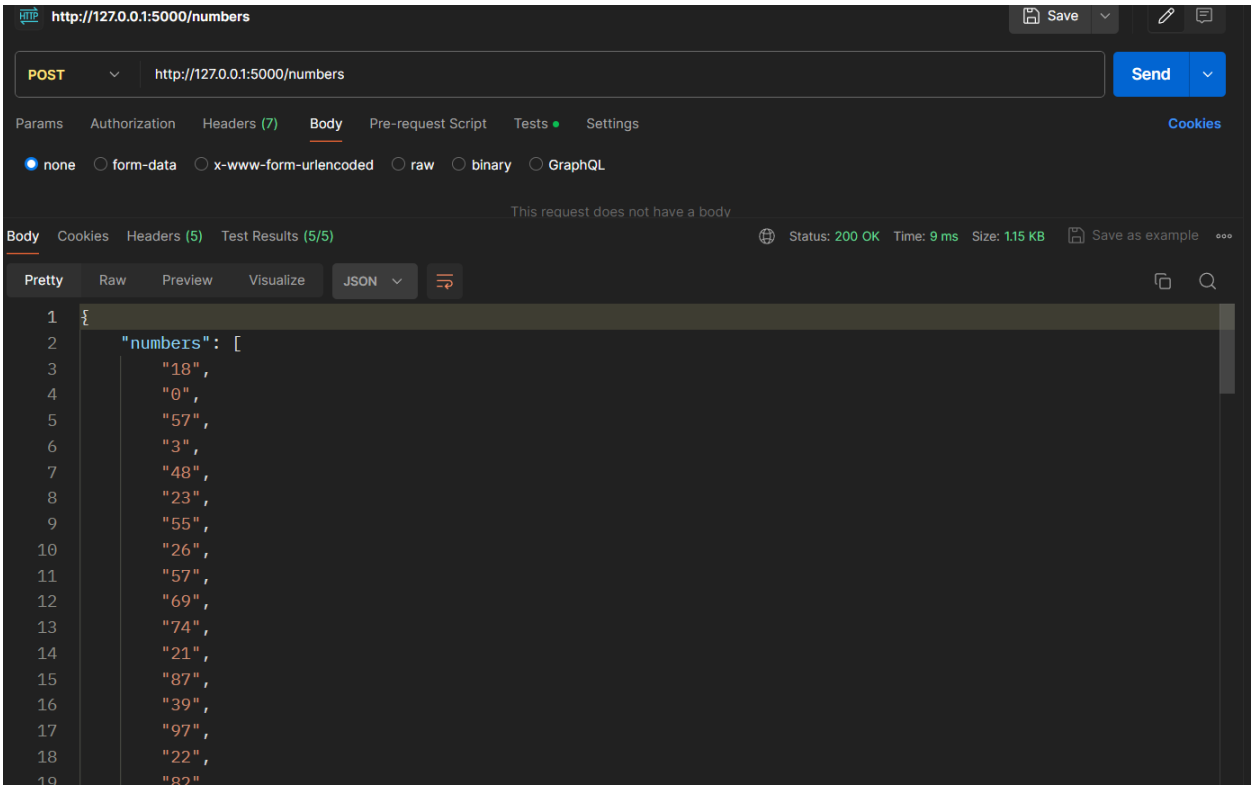
```python
@app.route("/numbers", methods=("POST",))
def numbers():
    request_data = request.json
    parameters_encrypted=request_data['parameters_encrypted']
    parameters_decrypted=receiver.decrypt_message(parameters_encrypted)
    parameters=json.loads(parameters_decrypted)

    cant=parameters['cant']
    limitmin=parameters['min']
    limitmax=parameters['max']

    flag = random.choice([True, False])
    random_numbers = None
    if flag:
        randomWay=RandomNormalDistribution()
    else:
        randomWay=RandomUniformDistribution()

    random_numbers=randomWay.generateNumbers(cant,limitmin,limitmax)
    random_numbers=[str(abs(n)) for n in random_numbers]
    response={'Numbers': random_numbers}
    response_encrypted=receiver.encrypt_message(json.dumps(response),'transmitter_key.pem')
    back={"response_encrypted":response_encrypted}
    return jsonify(back)
```

En la ruta /numbers, se recibe los parámetros encriptados (encriptados con la llave publica de receiver intercambiada anteriormente) en un JSON (cantidad de números, número mínimo, número máximo), luego se accede mediante la keyword ”parameters_encrypted” para obtener los datos encriptados. Luego se procede a hacer la desencriptación de los parámetros y con los valores ya descifrados se genera la lista de números aleatorios (el método de distribución de los datos es elegida al azar). Una vez se generen los datos, es posible que algunos valores salgan negativos si se usa la distribución normal, por lo que se busca que sean positivos siempre con la función abs() y directamente se convierte la lista de números enteros, en una lista de números pero que su tipo es string para encriptarlos y enviarlos de vuelta a ProblemSolver, encriptando ahora con la llave publica de ProblemSolver(”transmitter”). Una vez se encriptan los números son enviados de vuelta a ProblemSolver en formato JSON.

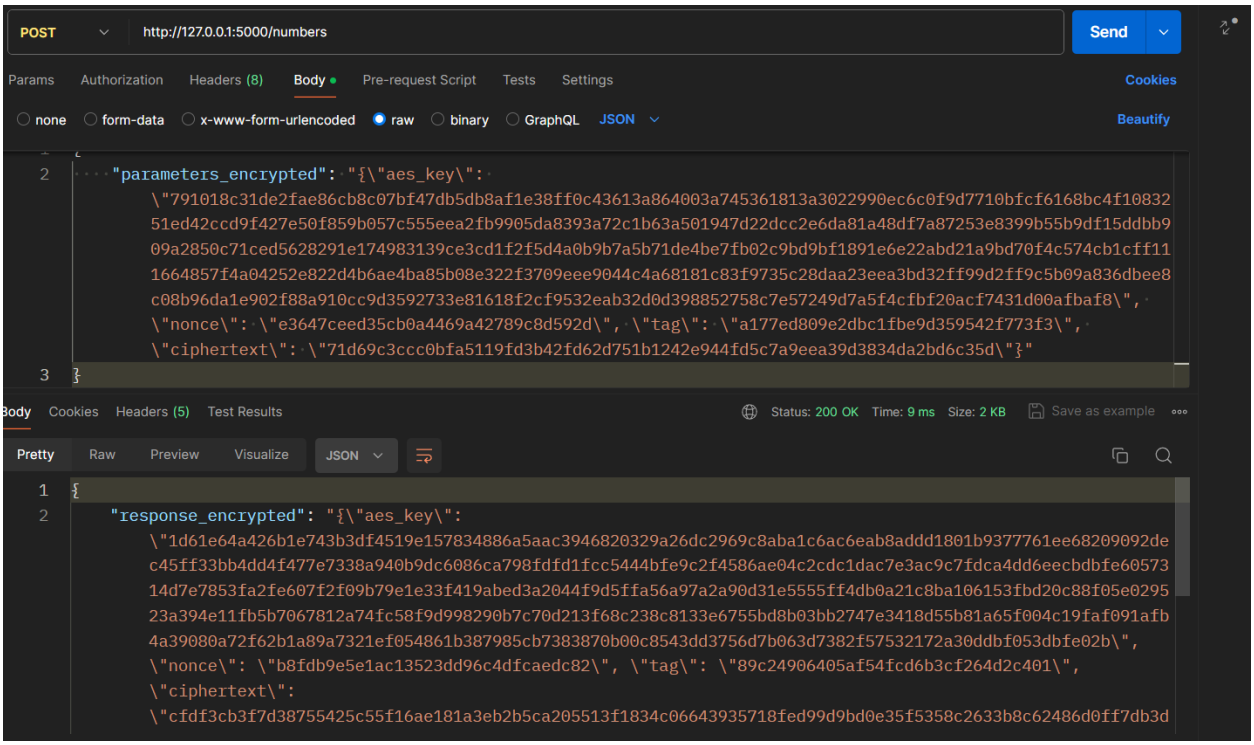
Para probar que si está retornando los valores primero se probó que se envíen los datos y luego se probará que todo se está enviando en json con las keywords explicadas en RSAEncryption. Todo esto en Postman.

```
pm.test("Response status code is 200", function () {
    pm.expect(pm.response.code).to.equal(200);
});

pm.test("Response has the required field - numbers", function () {
  const responseData = pm.response.json();
  pm.expect(responseData).to.be.an('object');
  pm.expect(responseData.numbers).to.exist.and.to.be.an('array');
});

pm.test("Numbers array should not be empty", function () {
    const responseData = pm.response.json();
    pm.expect(responseData.numbers).to.be.an('array').that.is.not.empty;
});

pm.test("Content type is application/json", function () {
    pm.expect(pm.response.headers.get("Content-Type")).to.include("application/json");
});

pm.test("Response time is less than 200ms", function () {
  pm.expect(pm.response.responseTime).to.be.below(200);
});
```

Se hace el envío de la cantidad de números en un rango y si los hace aleatorio como se puede ver en la siguiente imagen. Además, está pasando las pruebas (estas no son definitivas pues la importantes son cuando todo el producto funciona en conjunto).



Ahora se envía por el body como recibirá /numbers el diccionario (con los parámetros) con las keywords aes_key, nonce, tag y ciphertext. Y retorna los números igualmente codificados con las mismas keywords para luego en problem solver desencriptar los números.

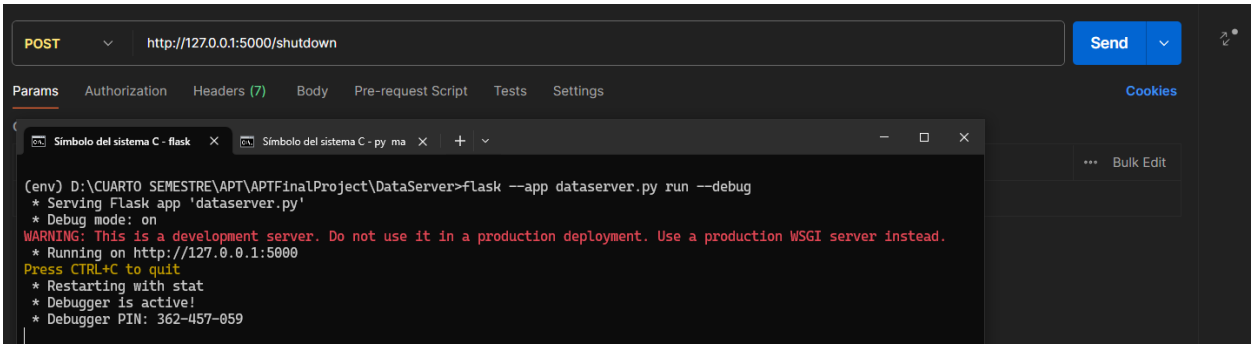



```
        @app.route('/shutdown', methods=['POST'])
        def shutdown():
            shutdown_server()
            return 'Server shutting down ...'

    return app

def shutdown_server():
    pid = os.getpid()
    os.kill(pid, signal.SIGINT)
```

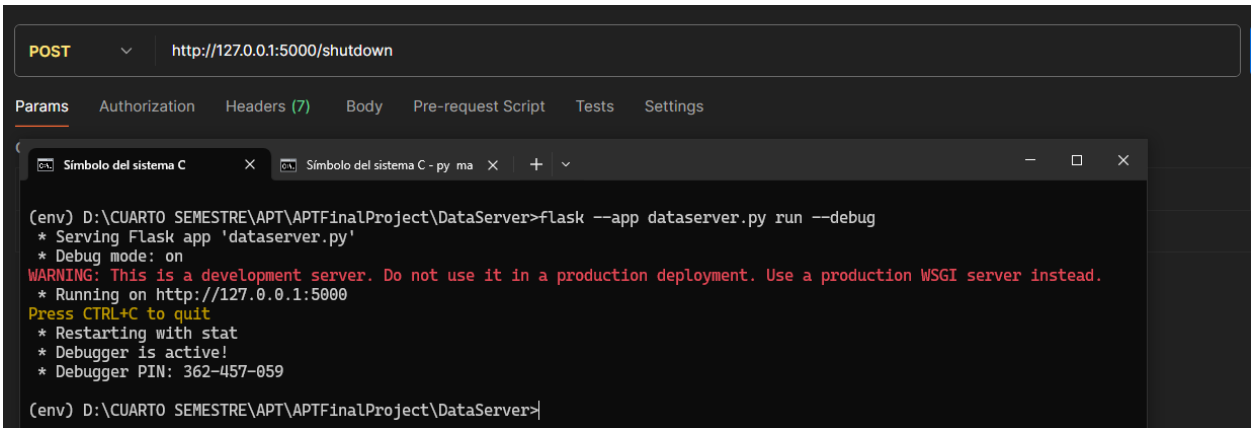
Para la ruta /shutdown se enviará la petición de apagado del sistema desde ProblemSolver para cerrar el servidor de Flask, sin embargo, esta tarea es un poco compleja ya que se si se hace directamente desde la librería SYS podría dañar el funcionamiento de la app e incluso de Flask(según lo revisado es bueno en forma de prueba pero, no es lo correcto si ya no está en etapa de producción), por lo que se buscó la forma de simular desde el código un comando de teclado CTRl + C para que deje de correr la aplicación de flask. Aunque de esta

manera se cierra la conexión bruscamente y se genera un error, este error puede ser manejado desde Problem solver como una señal de que la conexión con Flask fue cerrada exitosamente.

Se prueba desde postman que si detenga la ejecución de flask al mandar la solicitud POST a la ruta de shutdown:



Al enviar la solicitud, la aplicación de flask se detiene correctamente.

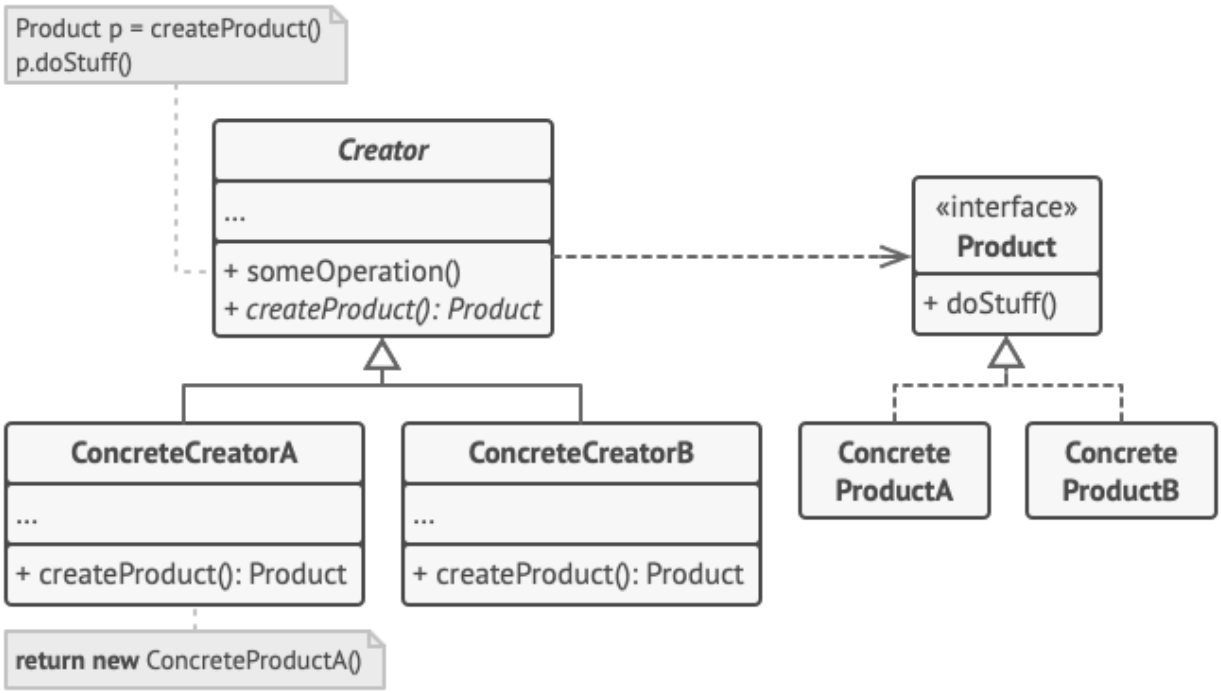


Problem Solver:

Según los requerimientos, se debe emplear un patrón de diseño creacional para instanciar los problemas a resolver. Para esto se usará el patrón de diseño Factory Method. Los problemas que resuelve ProblemSolver son los siguientes.

- FizzBuzz: especificado previamente en un taller.
- FibonacciVerifier: especificado previamente en un taller.
- PrimeClassifier: Clasifica un número, en primo, semiprimo-cuadrático, semiprimo, o ninguno de los anteriores. Un número semiprimo es aquel que tiene exactamente sólo dos divisores primos diferentes. Si los dos divisores primos son iguales, el número es un semiprimo-cuadrático.

Factory Method:



Para hacer uso del patrón de diseño se debe reconocer que papel tiene cada clase dentro del contexto y como se aplica en el patrón. Por lo tanto, se identificará el comportamiento de cada clase.

Se parte de que los problemas ya se solucionaron antes se usarán los archivos fizzbuzz.py, fibonacci.py y prime.py, que implementan la interfaz i_problem_solver.py.

Fizzbuzz.py:

```
class FizzBuzz(IProblemSolver):
  def compute_results(self, data: List[str]) -> List[str]:
    result = []
    line = ""
    for element in data:
      line = self.__fizzbuzz(int(element))
      result.append(element + " " + str(line))
    return result

  def __fizzbuzz(self, n):
    output = "Fizz" * (int(n) % 3 == 0) + "Buzz" * (int(n) % 5 == 0)
    return output or str(n)
```

Fibonacci.py:

```
class Fibonacci(IProblemSolver):
  def compute_results(self, data: List[str]) -> List[str]:
    result = []
    line = ""
    for element in data:
      line = self.__fibonacciVerifier(int(element))
      result.append(element + " " + str(line))
    return result

  def __fibonacciVerifier(self, n):
    a, b = 0, 1
    while b < n:
      a, b = b, a + b
    return b == n
```

Prime.py:

```python
class Prime(IProblemSolver):
  def compute_results(self, data: List[str]) -> List[str]:
    result = []
    line = ""
    for element in data:
      line = self.__prime_classifier(int(element))
      result.append(element + " " + str(line))
    return result

  def __prime(self,n):
    if n ≤ 1:
        return False
    if n ≤ 3:
        return True
    if n % 2 == 0 or n % 3 == 0:
        return False
    i = 5
    while i * i ≤ n:
        if n % i == 0 or n % (i + 2) == 0:
            return False
        i += 6
    return True
```

```python
def __prime_classifier(self,n):
  if self.__prime(n):
      return "prime"

  prime_divisors = []
  for i in range(2, n):
      if n % i == 0 and self.__prime(i):
          prime_divisors.append(i)
          if len(prime_divisors) > 2:
              break

  if len(prime_divisors) == 2:
      if prime_divisors[0] * prime_divisors[1] == n:
          return "semiprime"
  elif len(prime_divisors) == 1:
      if prime_divisors[0] ** 2 == n:
          return "cuadratic-semiprime"
  return "No classified"
```

I_problem_solver.py:

```python
from abc import ABC, abstractmethod
from typing import List

class IProblemSolver(ABC):
    @abstractmethod
    def compute_results(self, data: List) -> List:
        pass
```

Según el diagrama que explica el Factory Method, la interfaz Product vendría siendo la interfaz IProblemSolver y las clases que implementan Product(ConcreteProducts), vendrían siendo las clases FizzBuzz, Fibonacci y Prime.

Para la clase Creator, se tendría que crear una clase ProblemCreator y las clases que implementan de ella, teniendo así las clases creadoras de cada ConcreteProduct, entonces se crearan de la siguiente manera: FizzbuzzCreator, FibonacciCreator y PrimeCreator.

ProblemCreator.py:

```python
from .i_problem_solver import IProblemSolver

class ProblemCreator():
    def factoryMethod(self) -> IProblemSolver:
        pass

    def problemSolution(self, data):
        problem = self.factoryMethod()
        result = problem.compute_results(data)
        return result
```

FizzbuzzCreator.py:

```python
from .fizzbuzz import FizzBuzz
from .problem_creator import ProblemCreator
from .i_problem_solver import IProblemSolver

class FizzbuzzCreator(ProblemCreator):
    def factoryMethod(self) -> IProblemSolver:
        return FizzBuzz()
```

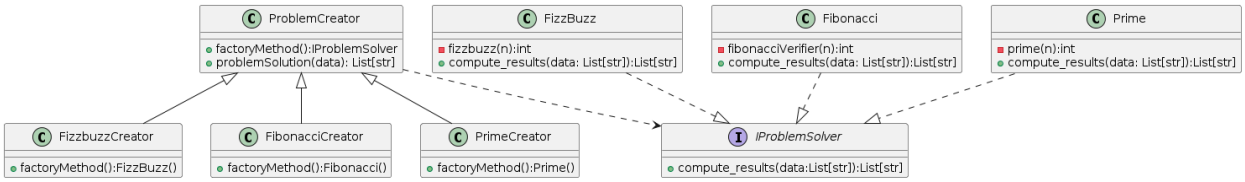
FibonacciCreator.py:

```python
from FactoryMethod.fibonacci import Fibonacci
from .problem_creator import ProblemCreator
from FactoryMethod.i_problem_solver import IProblemSolver

class FibonacciCreator(ProblemCreator):
    def factoryMethod(self) -> IProblemSolver:
        return Fibonacci()
```

PrimeCreator.py:

```python
from .problem_creator import ProblemCreator
from FactoryMethod.i_problem_solver import IProblemSolver
from FactoryMethod.prime import Prime

class PrimeCreator(ProblemCreator):
    def factoryMethod(self) -> IProblemSolver:
        return Prime()
```

Diagrama Factory Method Implementación.



Main.py:

Se pensó la manera de que se pueda conectar el Client con ProblemSolver(sabiendo que se hace con el socket definido en el main), y ProblemSolver con Dataserver(haciendo peticiones POST al servidor de flask). Se implementó el uso de Log (para cumplir con el requisito) e impresiones en consola para saber que está haciendo el main en ProblemSolver.

```
def main():
    logging.basicConfig(filename='app.log',
    filemode='a',
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
    datefmt='%Y-%m-%d %H:%M:%S',
    level=logging.INFO)

    global shutdown_flag
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind(('0.0.0.0', 8080))
    server_socket.listen(5)
    print('Server listening at the port 8080 ... ')

    while not shutdown_flag:
        client_socket, addr = server_socket.accept()
        print('Connection accepted by:', addr)
        logging.info("————————————————————————————")
        logging.info(f'Connection accepted by: {addr}')
        handle_client_connection(client_socket,addr)

    server_socket.close()
```

Se define la configuración del Log para guardarlo en un archivo app.log. Se crea el servidor socket y se configura el puerto por el que se comunicará con Client (8080, puesto definido en él). Se usó una flag global para hacer shutdown al sistema ya que se trabajará con un while pero dentro de la función handle_client_connection está la condición de que cuando se vaya a hacer shutdown cambia el valor de la flag para cerrar la conexión del socket.

```
Server listening at the port 8080...
Connection accepted by: ('127.0.0.1', 54137)
```

Se comprueba que se esté escuchando desde ProblemSolver a Client y se establece la conexión. (esto se imprimió por consola).

```
- root - INFO - ------------------------------------
- root - INFO - Connection accepted by: ('127.0.0.1', 54137)
```

Y este queda registrado en el Log de ProblemSolver.

```python
def handle_client_connection(client_socket,addr):
    global shutdown_flag
    try:
        while True:
            request_data = client_socket.recv(1024)
            if not request_data:
                break
            print(f"Request received: {request_data.decode('utf-8')}")
            logging.info("Request data received by Client")
            request = json.loads(request_data.decode('utf-8'))

            response = {
                'Result': None
            }

            shutdown = request.get('Shutdown')
            password = request.get('Password')
            if shutdown == 1:
                logging.info("Shutdown request received")
                if checkPassword(password):
                    print("Shutdown request accepted.")
                    logging.info("Shutdown request accepted")
                    response['Result'] = ["System down"]
                    response_shutdown = json.dumps(response)
                    client_socket.send(response_shutdown.encode('utf-8'))
                    shutdown_flag = True
                    logging.info("Closing client socket")
                    logging.info("Closing Flask server")
                    requests.post('http://localhost:5000/shutdown')
                    break
                else:
                    print("Shutdown request denied, incorrect password.")
                    logging.info("Shutdown request denied(Incorrect password)")
                    response['Result'] = ["Incorrect password"]
                    response_shutdown = json.dumps(response)
                    client_socket.send(response_shutdown.encode('utf-8'))
```

En la función handle_client_connection se guarda en request_data los datos que se hayan enviado desde el socket. Se decodifica a string el request_data y se convierte a diccionario. Se declara un diccionario response donde sin importar que resultado se obtenga debe estar en formato Lista[string].

Luego como ya se puede acceder al contenido del diccionario, se empieza a trabajar sobre la lógica del funcionamiento, donde se toma primero el caso de shutdown, es decir, que se haya hecho la petición de apagado (esto cuando se usa en Client el subcomando shutdown), dentro de la lógica del shutdown se verifica que la contraseña sea correcta esto se hace mediante checkPassword (función que se explicará a continuación), y si la contraseña es correcta se define el resultado como una lista de string con el mensaje [“System Down”] pasándolo a string enviándolo como respuesta a Client por el socket. En este punto se cambia el valor de la flag global para que deje de ejecutar el socket y acabe la ejecución del problemSolver. Posterior a eso, se envía una petición POST a la ruta “/shutdown” del servidor de flask para acabar con la ejecución del flask, que como se mostró antes funciona correctamente.

Por otro lado, cuando la contraseña es incorrecta, al resultado se le asigna [“Incorrect Password”] pues la contraseña es incorrecta, y finalmente se envía la respuesta por el socket.

Para probarlo, se hace la prueba con una contraseña incorrecta y la correcta.

```
(env) D:\CUARTO SEMESTRE\APT\APTFinalProject\ProblemSolver>py main.py
Server listening at the port 8080...
Connection accepted by: ('127.0.0.1', 54819)
Request received: {"Problem":"","Cant":0,"Min":0,"Max":0,"Result":"","Shutdown":1,"Password":"d33ad893
b3004742097fd0140425cfbaf15dda083db28c5385e7fa40231558b1"}
Shutdown request denied, incorrect password.
Reconnecting to client_socket...
Connection accepted by: ('127.0.0.1', 54819)
```

Se hace el intento de apagado con una contraseña incorrecta y como se puede ver en la captura, se muestra en Problemsolver que la petición de apagar el sistema fue denegada por que la contraseña no es correcta.

La conexión con el socket no debe terminar, pero surge un error (aun conectado con el socket), este se trabaja mediante una excepción de forma que se reconecte de nuevo:

```
except ConnectionResetError:
    print(f"Reconnecting to client_socket ... ")
    print('Connection accepted by:', addr)
    logging.info(f'Reconnected to {addr}')
```

Se hace el intento de apagado con la contraseña correcta. Y se apaga correctamente el sistema.

```
Connection accepted by: ('127.0.0.1', 54919)
Request received: {"Problem":"","Cant":0,"Min":0,"Max":0,"Result":"","Shutdown":1,"Password":"6d2df7bd
5b23426af87b65f44af0a7243b37649a9fe312e29e65d21c2365f528"}
Shutdown request accepted.
Flask Connection Down Success
Server socket off
```

Con respecto a flask, cuando se hace el cierre de flask, se hace de una manera muy "brusca", cerrando fuertemente el programa por lo que ProblemSolver lo toma como un error de conexión con DataServer. Este error se maneja como una excepción desde ProblemSolver pero comunicando igualmente que el servidor Flask si fue cerrado:

```
except requests.exceptions.ConnectionError:
    print("Flask Connection Down Success")
    logging.info("Flask connection closed")
```

Ahora se explicará el funcionamiento de la función checkPassword:

```
def readFileConfiguration():
    config = {}
    with open("config.txt", 'r') as file:
        for line in file:
            line = line.strip()
            if line:
                key, value = line.split(':', 1)
                config[key.strip()] = value.strip()
    return config.get('PASSWORD')

def checkPassword(password):
    correctPassword=readFileConfiguration()
    if password==correctPassword:
        return True
    return False
```

Se le envía la contraseña como parámetro y se verifica con la contraseña correcta que está almacenada en el archivo config.txt, este se lee mediante la función readFileConfiguration para traer la contraseña hasheada y guardarla en la variable correctPassword para finalmente comparar si son iguales.

Archivo config.txt:

```
ProblemSolver > config.txt
1    PASSWORD:6d2df7bd5b23426af87b65f44af0a7243b37649a9fe312e29e65d21c2365f528
```

Continuando con el funcionamiento, ahora se explicará el proceso cuando no se hace shutdown, sino que se envían los datos desde el comando rootCmd.

```
problems=["fizzbuzz","fibonacci","prime"]
if request.get('Problem') in problems:
    transmitter=RSAEncryption('transmitter')
    transmitter.generate_keys()
    transmitter_key=transmitter.to_json()
    key_response=requests.post('http://localhost:5000/key',json=transmitter_key)
    receiver_key=key_response.json()['public_key']
    logging.info("Public encryption keys exchanged")

    problem = request.get('Problem')
    values_count = int(request.get('Cant'))
    min_value = int(request.get('Min'))
    max_value = int(request.get('Max'))
    parameters={
    "cant":values_count,
    "min":min_value,
    "max":max_value,
    }

    print(f"Resolving problem: {problem} with {values_count} values between {min_value} and {max_value}")
    result = solve_problem(problem, parameters,transmitter,receiver_key)
    response['Result']=result
    response_data = json.dumps(response)
    print(f"Send response: {response_data}")
    client_socket.send(response_data.encode('utf-8'))
    logging.info("Results sent to Client")
else:
    response = {
        'Result': ["Problem not found"]
    }
    response_data=json.dumps(response)
    client_socket.send(response_data.encode('utf-8'))
    logging.info("Problem not found")
```

Se define que problemas son los disponibles para usar y si el problema enviado por el cliente no está entre fizzbuzz, fibonacci y prime, se asigna a la respuesta el valor de [“Problem Not Found”] y se envía por el socket este mensaje a Client.

Si el problema si existe, se declara el objeto de para encriptar (RSAEncryption teniendo en cuenta que es el emisor, por ende, se le puso el nombre de “transmitter”).

Una vez se generen las llaves de transmitter, se convertirán a JSON para enviarlo como petición POST a la url /key y efectuar el cambio de llaves publicas entre ProblemSolver y DataServer.

Luego se extraen los parámetros enviados desde Client (problema, cantidad, valor mínimo y valor máximo) y se le envían a la función solve_problem (Esta función se explicará a continuación). Luego de resolver el problema se guardan los resultados en un diccionario para enviarlo también de forma Lista[string], siendo los strings todos los números con su respectivo resultado. Finalmente, se envían por el socket los resultados para mostrarlos en Client.

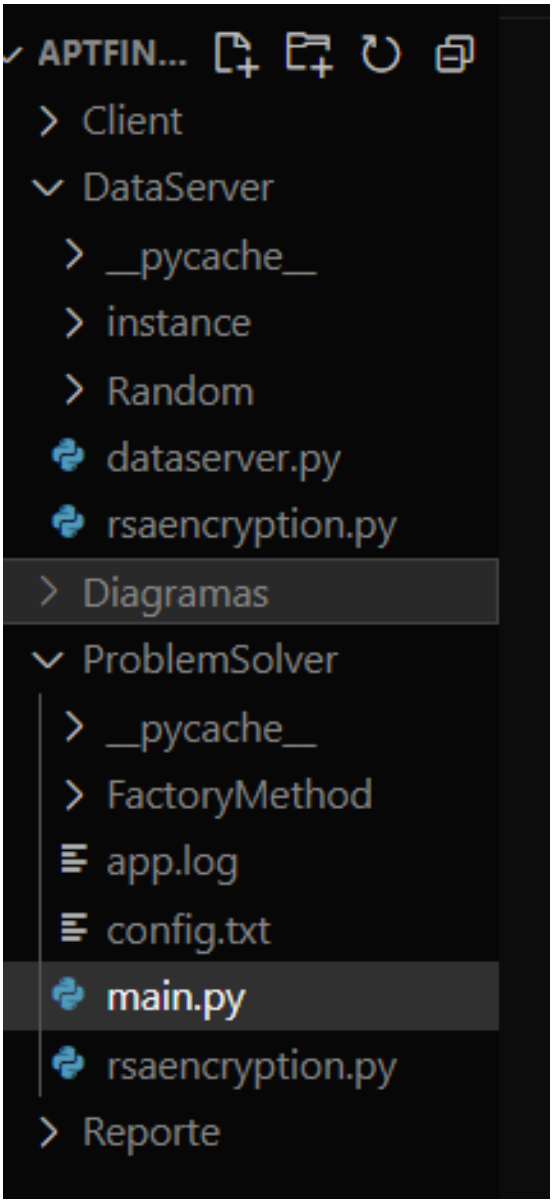
Ahora se explica la función solve_problem:

```
def solve_problem(problem, parameters, transmitter, receiver_key):
    if problem == "fizzbuzz":
        problemSolver=FizzbuzzCreator()
    if problem == "fibonacci":
        problemSolver=FibonacciCreator()
    if problem == "prime":
        problemSolver=PrimeCreator()

    with open("receiver_key.pem","w") as file:
        file.write(receiver_key)

    parameters_encrypted=transmitter.encrypt_message(json.dumps(parameters),'receiver_key.pem')
    logging.info("Data encrypted")
    print(parameters_encrypted)
    send={
        "parameters_encrypted":parameters_encrypted
    }
    logging.info("Sending data encrypt to DataServer")
    response=requests.post('http://localhost:5000/numbers',json=send)
    logging.info("Encrypted response data received by DataServer")
    back_encrypted=response.json()['response_encrypted']
    back_decrypted=transmitter.decrypt_message(back_encrypted)
    logging.info("Response data decrypted")
    numbers=json.loads(back_decrypted)['Numbers']
    logging.info(f'Resolving problem {problem}')
    result=problemSolver.problemSolution(numbers)
    logging.info('Problem solved successfully')
    return result
```

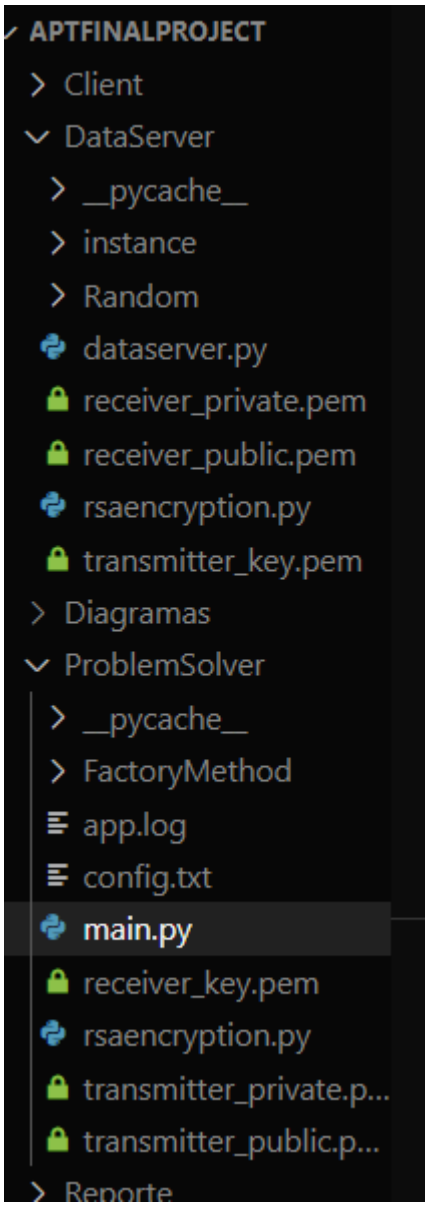
Se recibe como parámetro el problema, los parámetros, cifrador (”emisor”) y la llave pública del receptor. Primero se verifica cuál de los 3 problemas es el que se va a resolver, y en este punto se usa el Factory Method para crear un objeto dependiendo de que problema sea. Una vez se haya instanciado el tipo del problema se toma la llave pública del receptor para encriptar los parámetros con esta llave (simultáneamente en DataServer se generan los números que vuelven encriptados con la llave pública del emisor), cuando le llegan los números a ProblemSolver los desencripta para finalmente usar la solución del problema dependiendo el tipo de problema que haya instanciado. Una vez se termina el proceso se envía en forma de Lista[string] los resultados a Client.

Se probará que se crean las llaves privadas y públicas del emisor y del receptor (ProblemSolver y DataServer, respectivamente), además de que se haga el intercambio correcto de las llaves).

Carpeta DataServer y ProblemSolver antes de crear las llaves:



Carpeta DataServer y ProblemSolver después de crear las llaves:



Como se muestra a continuación se crean las llaves privada y pública del emisor, además en la parte inferior derecha está la llave que tiene el receptor, que como se puede ver, son idénticas. Esencialmente es lo que debe tener el receptor para encriptar datos que el emisor pueda desencriptar.

transmitter_private.pem

ProblemSolver > transmitter_private.pem

```
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEA2rY/u0wts3B0komS/OZQ7tT1vWI7QtemwY952f5c+LIBUduf
9jnH7di+MW35NsbZEtwD/w3tv+Lx8myIMmeIs/7Azs9j005hDTJcnKcLMVS628YB
ETS/nRxUI/TIsI649wn7Cun2nT8zWuxhn0w92Ket8k4bXqjK70EifOYa4yHGsFl9
g9A/ktxaDrEXWoMEC4r8Gs0nTOZAg2QfbEM9AFRLada0ct+1mt+Lse0rHXowiQHn
SCU3hqpYH1VJk5zEva5ww2DsDO1yISp7DxKnZqt9n0hA5sB8nLaYBqmK+2wWvnBH
6U7Kkxvgn3N9zBJ1STyETE1Vbhvblcedsady SQIDAQABAoIBACHwii9MgQMJfm2G
/9JTA1gsn/3FkqCqymvbEfOsL01Agkfrpw UUrlvQeKM/B59ieCkfD1ezF6PhWd5e
zTvVKTeVXuI4+Kq/zPu5RvngSjJbjauxLFU5NErzLHNv9vy/CSD84YIuEGESf6uq
ZvPLZwWJ78pcKI8l7mf2WZ28L4vzVmtZW9sSYfZu3ZPdQsOUTyBJ6oyxIbNCzXCB
aSsVjiXDHUeF1Bxd+9wwzRj4DXh2CugtUN1eiBiZ96yZ53irU2Bfe4LfMjYPQ+ll
j19hHPXlyw/0bXZq5jOCJd5vMpMrhZViX6Eq3bFRZ0dsERjL4Uer4bRteWn9ld/Y
uyoPLekCgYEA67pIHZFQTmi3McE+usw1wBTJ5SSXrGUPRhTQVoa/uslJ8axEdiC9
kAOF/lvVnxQ40ZB2tPBQDuhVOzivW2qVa2Uz9sNcIrPY5DkgOfdDMb/pnMoo91sk
S48wUN1UZLolOS5MtH02xSCMAld3HL/VoFvUD9EJNtE8aoQ/SM5TgmMCgYEA7YVa
V+6EPuAeT3Ov6qgAK7u5hFgi5Ks2dqX8GyV7n7vd67j2SVdeTk3hjLAHLJIaXsOT
eiq9Txpt4BIn96kvUjRPSMEr4GyggwPxqr3iLdXPZYvIm/zgFx2Gcdaw5sMC4cTm
lWWwCflOQTysxZ0afr/eH/8JksUsXvV5wbZPwmMCgYBN/LkzHwpEhpxlF5vbTQhT
Xg1Vg/+yB1VASE6M8dOoxkNycx3LPxWVkWTZPCS2logdvP+o6QBzjiX0i9JoGiEJ
ANhqQUPQL1QIXWXuB4HxiZ6OQsu+2iZt4ReJCVCMapE6E7lI1Uwzd4sakzlTB5xz
iy3jdbLosdydGNefDLPQJQKBgEEMMD28mIMPpb1klVzSPZm8g4X3DTdKnFPKCUz1
6vFH5fLUaL6VKsKPGaIJ4y0VphslvJHMeqPbPRQMZN2ye78rjDsEjBueJCc9QUXd
Ur/yuiW5LmtpyYULFPLANRo9deCWp/9qJdfVatT65OYJrGPh79cbZWfIj+pBl+Kx
Jw65AoGBAOYyi5s3ILCMP5GmqD472ftzQpt4wj6BJdUdgKyZ1DxFVtEjqk476Uh4
UuvSoOuGVDzFBu+ninM1kqx4dxCAzWCyqhSlhIqY8qiDNY1LytW/DyErCkLAp5GI
18VOlcvrBsXzkCK2uPv2zsfojDJo6DnueKscKR9vg3GcZZJwbj/+
-----END RSA PRIVATE KEY-----
```

transmitter_public.pem

ProblemSolver > transmitter_public.pem

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA2rY/u0wts3B0komS/OZQ
7tT1vWI7QtemwY952f5c+LIBUduf9jnH7di+MW35NsbZEtwD/w3tv+Lx8myIMmeI
s/7Azs9j005hDTJcnKcLMVS628YBETS/nRxUI/TIsI649wn7Cun2nT8zWuxhn0w9
2Ket8k4bXqjK70EifOYa4yHGsFl9g9A/ktxaDrEXWoMEC4r8Gs0nTOZAg2QfbEM9
AFRLada0ct+1mt+Lse0rHXowiQHnSCU3hqpYH1VJk5zEva5ww2DsDO1yISp7DxKn
Zqt9n0hA5sB8nLaYBqmK+2wWvnBH6U7Kkxvgn3N9zBJ1STyETE1Vbhvblcedsady
SQIDAQAB
-----END PUBLIC KEY-----
```

transmitter_key.pem

DataServer > transmitter_key.pem

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA2rY/u0wts3B0komS/OZQ
7tT1vWI7QtemwY952f5c+LIBUduf9jnH7di+MW35NsbZEtwD/w3tv+Lx8myIMmeI
s/7Azs9j005hDTJcnKcLMVS628YBETS/nRxUI/TIsI649wn7Cun2nT8zWuxhn0w9
2Ket8k4bXqjK70EifOYa4yHGsFl9g9A/ktxaDrEXWoMEC4r8Gs0nTOZAg2QfbEM9
AFRLada0ct+1mt+Lse0rHXowiQHnSCU3hqpYH1VJk5zEva5ww2DsDO1yISp7DxKn
Zqt9n0hA5sB8nLaYBqmK+2wWvnBH6U7Kkxvgn3N9zBJ1STyETE1Vbhvblcedsady
SQIDAQAB
-----END PUBLIC KEY-----
```

Como se muestra a continuación se crean las llaves privada y pública del receptor, además en la parte inferior derecha está la llave que tiene el emisor, que como se puede ver, son idénticas. Esencialmente es lo que debe tener el receptor para encriptar datos que el emisor pueda desencriptar.

receiver_private.pem

DataServer > receiver_private.pem

```
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAyP32S0SZ6/Bg5Kq2hkiKtdk8UAszsYjcw/mnox3YHyehsXuO
q1b9DYxUhW0CiR/kWOADX19VR0WZWq3kgscjtEhw+kYDRBBvoIsRyksx+Ug9haJx
9FWR0inMwE8k2G3TGDzv8zKJbBZwBUUzsiwtfRaFRvZpWYoltoOaton4Mw0/s8bY
vs7X0ffbPaDoMw6ORaLvXrJa/Iarxpbjkw/zW09M7VGPF0V5oriYFHVXbzqmr2nk
XdkLl/1xIx3SwEYXHhSEsKO3FOgpcjGMiFa4Jha34DmyPgEnQ5Gh8y/koETLFwIF
hG4ZTWVxC2DqI8lCMSyr40i9K2K08DxpUHMuGQIDAQABAoIBAB8fVBfrk3n5Ssed
t42rhe8KqFAsq+B9tNGTwuV7JZANbvgwjbLyUVTbueq/KnlZOt6YMW1/MZFLsadc
7dliRKER5oyacr8DXIWbgzGz2xUinoQm10ZiSaYlWtMcbjQlLV6/cNhTpvvZrZbj
VFHujUylRBXs6fs3bQeJ8Vb6njgpOnjFoL4SLUBXFMNZxOw742FaKXDuMKYxMv5t
h73LRW7mihJTeG5b00F2y7uhE1tA6DNwVEMUah4PZdJdANLtu8nOvK9q1SNXsm2I
TZ0KiCLEGvfNanUaMEUl1acqiQssFI37d5Yt8uU/yGkRLi1ZR5OkFkfoyNcuZUqc
OcNkQVECgYEAzn1bcdbVzSHtQFAzoinIUrv1kLv09cHQvz4sCS+b1SDURlLB6C92
bbIujESSxES84OWIQJEtnT7haXY99amD873fD4T4lZ2qODVkSjn5L4S7pyeTg0yr
IF31CVwzOv/ZRs8HG8FSogNUNLInnNXAcsUKg+DMVJPELeyvyJVy3gcCgYEA+S8m
2SPWcqiqt1Je1kg4N16NhCLB8mNUUT8/NOsypjIbI9hG7f+dw3TGqSpGSs8qCK5S
DqyqSna84MgmHXgRlYNCtuhGaXVNvXEApFp4cjJAYdo9yfj9F/eFHXVhXLFYDHP5
j9AV45UTApvINGsCIIcvU5n4Z2Ha9WFaq2+sit8CgYByouXBraPE3rD4o44G2Wt9
gxv4IWE9jvx1L+8Cqou5JyCyaf99B9yXyfRNq0RP8llqAgRC1FaelepOwnZjjFjV
WPQdiZT4ey+lRP8ItmbBGs+TXUVn/4iBz6faknXGq4PU69rmbc2SZIa7Oq0DRSgy
j3HnBUZrLh4b4Os2UMIUwQKBgQC2JdN98oZ1tflknB+a/5uoTcOgkhAUwKsH4PWx
FyQk/Id+QkBHBQ0rrRby3WaKnXCae+Kbu3eXiwgyP9wsunIQGq0PyuZ9FDPhgNL5
UPCueTE61i9Ijmtjs5lomT8cZL56C+SnTXJSs8v7oihMuNQLNwZPVEOSPqOxrwJ6
Qt2bfQKBgDVr8fwX4yeTE0fd054jh5t8VSFGzkzQL9Wo94FYhbFrHaBnbQ77Q90v
DtHCul4OQ/f4EQxvEBKaslojJP3Di89CvA6ADpPM8qs8msOEzHms2as8n+C9hjwq
1te9ldVeJzz2tbGrZUPMdnCGbjY156JY6BeEn2z0EOgvwGxy6pKE
-----END RSA PRIVATE KEY-----
```

receiver_public.pem

DataServer > receiver_public.pem

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAyP32S0SZ6/Bg5Kq2hkiK
tdk8UAszsYjcw/mnox3YHyehsXuOq1b9DYxUhW0CiR/kWOADX19VR0WZWq3kgscj
tEhw+kYDRBBvoIsRyksx+Ug9haJx9FWR0inMwE8k2G3TGDzv8zKJbBZwBUUzsiwt
fRaFRvZpWYoltoOaton4Mw0/s8bYvs7X0ffbPaDoMw6ORaLvXrJa/Iarxpbjkw/z
W09M7VGPF0V5oriYFHVXbzqmr2nkXdkLl/1xIx3SwEYXHhSEsKO3FOgpcjGMiFa4
Jha34DmyPgEnQ5Gh8y/koETLFwIFhG4ZTWVxC2DqI8lCMSyr40i9K2K08DxpUHMu
GQIDAQAB
-----END PUBLIC KEY-----
```

receiver_key.pem

ProblemSolver > receiver_key.pem

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAyP32S0SZ6/Bg5Kq2hkiK
tdk8UAszsYjcw/mnox3YHyehsXuOq1b9DYxUhW0CiR/kWOADX19VR0WZWq3kgscj
tEhw+kYDRBBvoIsRyksx+Ug9haJx9FWR0inMwE8k2G3TGDzv8zKJbBZwBUUzsiwt
fRaFRvZpWYoltoOaton4Mw0/s8bYvs7X0ffbPaDoMw6ORaLvXrJa/Iarxpbjkw/z
W09M7VGPF0V5oriYFHVXbzqmr2nkXdkLl/1xIx3SwEYXHhSEsKO3FOgpcjGMiFa4
Jha34DmyPgEnQ5Gh8y/koETLFwIFhG4ZTWVxC2DqI8lCMSyr40i9K2K08DxpUHMu
GQIDAQAB
-----END PUBLIC KEY-----
```

Fase GREEN

Para probar el correcto funcionamiento de todo el software, se ejecutarán todos los casos posibles para probar el funcionamiento de todo.

Ejecución predeterminada (go run main.go)

```
(env) D:\CUARTO SEMESTRE\APT\APTFinalProject\ProblemSolver>py main.py
Server listening at the port 8080...
Connection accepted by: ('127.0.0.1', 50091)
Request received: {"Problem":"fizzbuzz","Cant":25,"Min":0,"Max":100,"Result":"","Shutdown":0,"Password":""}
Resolving problem: fizzbuzz with 25 values between 0 and 100
Send response: {"Result": ["5 Buzz", "51 Fizz", "21 Fizz", "51 Fizz", "34 34", "72 Fizz", "62 62", "75 FizzBuzz", "55 Buzz", "49 49", "63
 Fizz", "36 Fizz", "68 68", "3 Fizz", "48 Fizz", "73 73", "49 49", "16 16", "33 Fizz", "106 106", "39 Fizz", "88 88", "53 53", "45 FizzBu
zz", "38 38"]}
Client socket closed
```

```
(env) D:\CUARTO SEMESTRE\APT\APTFinalProject\DataServer>flask --app
dataserver.py run --debug
 * Serving Flask app 'dataserver.py'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production
 deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 362-457-059
127.0.0.1 - - [06/Jun/2024 16:23:39] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 16:23:41] "POST /numbers HTTP/1.1" 200 -
```

```
Símbolo del sistema C

D:\CUARTO SEMESTRE\APT\APTFinalProject\Client>go run main.go
Data sent to the server: {"Problem":"fizzbuzz","Cant":25,"Min":0,"Max":100,"Result":"","Shutdown":0,"Password":""}
[5 Buzz 51 Fizz 21 Fizz 51 Fizz 34 34 72 Fizz 62 62 75 FizzBuzz 55 Buzz 49 49 63 Fizz 36 Fizz 68 68 3 Fizz 48 Fizz 73 73 49 49 16 16 33 F
izz 106 106 39 Fizz 88 88 53 53 45 FizzBuzz 38 38]

D:\CUARTO SEMESTRE\APT\APTFinalProject\Client>
```

Cada archivo va mostrando que proceso va realizando, en este caso como hay valores por defecto en las Flags en main.go se hizo el proceso con los valores por defecto.

Ejecución probando cada flag independientemente:

(go run main.go -p prime)

```
Connection accepted by: ('127.0.0.1', 50235)
Request received: {"Problem":"prime","Cant":25,"Min":0,"Max":100,"Result":"","Shutdown":0,"Password":""}
Resolving problem: prime with 25 values between 0 and 100
Send response: {"Result": ["40 No classified", "26 semiprime", "44 No classified", "94 semiprime", "61 prime", "54 No classified", "98 No
 classified", "30 No classified", "16 No classified", "22 semiprime", "60 No classified", "3 prime", "15 semiprime", "97 prime", "70 No c
lassified", "50 No classified", "25 cuadratic-semiprime", "26 semiprime", "89 prime", "44 No classified", "58 semiprime", "44 No classifi
ed", "74 semiprime", "58 semiprime", "60 No classified"]}
Client socket closed
```

```
Símbolo del sistema C

D:\CUARTO SEMESTRE\APT\APTFinalProject\Client>go run main.go
Data sent to the server: {"Problem":"fizzbuzz","Cant":25,"Min":0,"Max":100,"Result":"","Shutdown":0,"Password":""}
[5 Buzz 51 Fizz 21 Fizz 51 Fizz 34 34 72 Fizz 62 62 75 FizzBuzz 55 Buzz 49 49 63 Fizz 36 Fizz 68 68 3 Fizz 48 Fizz 73 73 49 49 16 16 33 F
izz 106 106 39 Fizz 88 88 53 53 45 FizzBuzz 38 38]

D:\CUARTO SEMESTRE\APT\APTFinalProject\Client>go run main.go -p prime
Data sent to the server: {"Problem":"prime","Cant":25,"Min":0,"Max":100,"Result":"","Shutdown":0,"Password":""}
[40 No classified 26 semiprime 44 No classified 94 semiprime 61 prime 54 No classified 98 No classified 30 No classified 16 No classified
 22 semiprime 60 No classified 3 prime 15 semiprime 97 prime 70 No classified 50 No classified 25 cuadratic-semiprime 26 semiprime 89 pri
me 44 No classified 58 semiprime 44 No classified 74 semiprime 58 semiprime 60 No classified]

D:\CUARTO SEMESTRE\APT\APTFinalProject\Client>
```

```
 * Debugger is active!
 * Debugger PIN: 362-457-059
127.0.0.1 - - [06/Jun/2024 16:23:39] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 16:23:41] "POST /numbers HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 16:38:50] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 16:38:53] "POST /numbers HTTP/1.1" 200 -
```

(go run main.go –q 10)

```
ed", "74 semiprime", "58 semiprime", "60 No classified"]}
Client socket closed
Connection accepted by: ('127.0.0.1', 50252)
Request received: {"Problem":"fizzbuzz","Cant":10,"Min":0,"Max":100,"Result":"","Shutdown":0,"Password":""}
Resolving problem: fizzbuzz with 10 values between 0 and 100
Send response: {"Result": ["88 88", "63 Fizz", "8 8", "71 71", "16 16", "60 FizzBuzz", "83 83", "43 43", "35 Buzz", "54 Fizz"]}
Client socket closed
```

```
127.0.0.1 - - [06/Jun/2024 16:41:57] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 16:41:59] "POST /numbers HTTP/1.1" 200 -
```

```
Símbolo del sistema C

D:\CUARTO SEMESTRE\APT\APTFinalProject\Client>go run main.go
Data sent to the server: {"Problem":"fizzbuzz","Cant":25,"Min":0,"Max":100,"Result":"","Shutdown":0,"Password":""}
[5 Buzz 51 Fizz 21 Fizz 51 Fizz 34 34 72 Fizz 62 62 75 FizzBuzz 55 Buzz 49 49 63 Fizz 36 Fizz 68 68 3 Fizz 48 Fizz 73 73 49 49 16 16 33 F
izz 106 106 39 Fizz 88 88 53 53 45 FizzBuzz 38 38]

D:\CUARTO SEMESTRE\APT\APTFinalProject\Client>go run main.go -p prime
Data sent to the server: {"Problem":"prime","Cant":25,"Min":0,"Max":100,"Result":"","Shutdown":0,"Password":""}
[40 No classified 26 semiprime 44 No classified 94 semiprime 61 prime 54 No classified 98 No classified 30 No classified 16 No classified
 22 semiprime 60 No classified 3 prime 15 semiprime 97 prime 70 No classified 50 No classified 25 cuadratic-semiprime 26 semiprime 89 pri
me 44 No classified 58 semiprime 44 No classified 74 semiprime 58 semiprime 60 No classified]

D:\CUARTO SEMESTRE\APT\APTFinalProject\Client>go run main.go -q 10
Data sent to the server: {"Problem":"fizzbuzz","Cant":10,"Min":0,"Max":100,"Result":"","Shutdown":0,"Password":""}
[88 88 63 Fizz 8 8 71 71 16 16 60 FizzBuzz 83 83 43 43 35 Buzz 54 Fizz]

D:\CUARTO SEMESTRE\APT\APTFinalProject\Client>
```

(go run main.go -l 90)

```
Connection accepted by: ('127.0.0.1', 50420)
Request received: {"Problem":"fizzbuzz","Cant":25,"Min":90,"Max":100,"Result":"","Shutdown":0,"Password":""}
Resolving problem: fizzbuzz with 25 values between 90 and 100
Send response: {"Result": ["90 FizzBuzz", "96 Fizz", "95 Buzz", "94 94", "93 Fizz", "91 91", "92 92", "97 97", "91 91", "94 94", "97 97",
 "99 Fizz", "93 Fizz", "99 Fizz", "97 97", "93 Fizz", "95 Buzz", "94 94", "93 Fizz", "97 97", "93 Fizz", "96 Fizz", "93 Fizz", "98 98", "
95 Buzz"]}
Client socket closed
```

```
127.0.0.1 - - [06/Jun/2024 16:41:57] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 16:41:59] "POST /numbers HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:02:32] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:02:34] "POST /numbers HTTP/1.1" 200 -
```

```
Símbolo del sistema C

D:\CUARTO SEMESTRE\APT\APTFinalProject\Client>go run main.go
Data sent to the server: {"Problem":"fizzbuzz","Cant":25,"Min":0,"Max":100,"Result":"","Shutdown":0,"Password":""}
[5 Buzz 51 Fizz 21 Fizz 51 Fizz 34 34 72 Fizz 62 62 75 FizzBuzz 55 Buzz 49 49 63 Fizz 36 Fizz 68 68 3 Fizz 48 Fizz 73 73 49 49 16 16 33 F
izz 106 106 39 Fizz 88 88 53 53 45 FizzBuzz 38 38]

D:\CUARTO SEMESTRE\APT\APTFinalProject\Client>go run main.go -p prime
Data sent to the server: {"Problem":"prime","Cant":25,"Min":0,"Max":100,"Result":"","Shutdown":0,"Password":""}
[40 No classified 26 semiprime 44 No classified 94 semiprime 61 prime 54 No classified 98 No classified 30 No classified 16 No classified
 22 semiprime 60 No classified 3 prime 15 semiprime 97 prime 70 No classified 50 No classified 25 cuadratic-semiprime 26 semiprime 89 pri
me 44 No classified 58 semiprime 44 No classified 74 semiprime 58 semiprime 60 No classified]

D:\CUARTO SEMESTRE\APT\APTFinalProject\Client>go run main.go -q 10
Data sent to the server: {"Problem":"fizzbuzz","Cant":10,"Min":0,"Max":100,"Result":"","Shutdown":0,"Password":""}
[88 88 63 Fizz 8 8 71 71 16 16 60 FizzBuzz 83 83 43 43 35 Buzz 54 Fizz]

D:\CUARTO SEMESTRE\APT\APTFinalProject\Client>go run main.go -l 90
Data sent to the server: {"Problem":"fizzbuzz","Cant":25,"Min":90,"Max":100,"Result":"","Shutdown":0,"Password":""}
[90 FizzBuzz 96 Fizz 95 Buzz 94 94 93 Fizz 91 91 92 92 97 97 91 91 94 94 97 97 99 Fizz 93 Fizz 99 Fizz 97 97 93 Fizz 95 Buzz 94 94 93 Fiz
z 97 97 93 Fizz 96 Fizz 93 Fizz 98 98 95 Buzz]
```

(go run main.go -s 10)

```
Connection accepted by: ('127.0.0.1', 50433)
Request received: {"Problem":"fizzbuzz","Cant":25,"Min":0,"Max":10,"Result":"","Shutdown":0,"Password":""}
Resolving problem: fizzbuzz with 25 values between 0 and 10
Send response: {"Result": ["7 7", "6 Fizz", "4 4", "8 8", "2 2", "11 11", "6 Fizz", "8 8", "6 Fizz", "7 7", "8 8", "0 FizzBuzz", "7 7", "
4 4", "4 4", "5 Buzz", "1 1", "7 7", "6 Fizz", "3 Fizz", "7 7", "6 Fizz", "2 2", "5 Buzz", "4 4"]}
Client socket closed
```

```
127.0.0.1 - - [06/Jun/2024 16:41:59] "POST /numbers HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:02:32] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:02:34] "POST /numbers HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:04:16] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:04:18] "POST /numbers HTTP/1.1" 200 -
```

```
Símbolo del sistema C

Data sent to the server: {"Problem":"fizzbuzz","Cant":25,"Min":0,"Max":100,"Result":"","Shutdown":0,"Password":""}
[5 Buzz 51 Fizz 21 Fizz 51 Fizz 34 34 72 Fizz 62 62 75 FizzBuzz 55 Buzz 49 49 63 Fizz 36 Fizz 68 68 3 Fizz 48 Fizz 73 73 49 49 16 16 33 F
izz 106 106 39 Fizz 88 88 53 53 45 FizzBuzz 38 38]

D:\CUARTO SEMESTRE\APT\APTFinalProject\Client>go run main.go -p prime
Data sent to the server: {"Problem":"prime","Cant":25,"Min":0,"Max":100,"Result":"","Shutdown":0,"Password":""}
[40 No classified 26 semiprime 44 No classified 94 semiprime 61 prime 54 No classified 98 No classified 30 No classified 16 No classified
 22 semiprime 60 No classified 3 prime 15 semiprime 97 prime 70 No classified 50 No classified 25 cuadratic-semiprime 26 semiprime 89 pri
me 44 No classified 58 semiprime 44 No classified 74 semiprime 58 semiprime 60 No classified]

D:\CUARTO SEMESTRE\APT\APTFinalProject\Client>go run main.go -q 10
Data sent to the server: {"Problem":"fizzbuzz","Cant":10,"Min":0,"Max":100,"Result":"","Shutdown":0,"Password":""}
[88 88 63 Fizz 8 8 71 71 16 16 60 FizzBuzz 83 83 43 43 35 Buzz 54 Fizz]

D:\CUARTO SEMESTRE\APT\APTFinalProject\Client>go run main.go -l 90
Data sent to the server: {"Problem":"fizzbuzz","Cant":25,"Min":90,"Max":100,"Result":"","Shutdown":0,"Password":""}
[90 FizzBuzz 96 Fizz 95 Buzz 94 94 93 Fizz 91 91 92 92 97 97 91 91 94 94 97 97 99 Fizz 93 Fizz 99 Fizz 97 97 93 Fizz 95 Buzz 94 94 93 Fiz
z 97 97 93 Fizz 96 Fizz 93 Fizz 98 98 95 Buzz]

D:\CUARTO SEMESTRE\APT\APTFinalProject\Client>go run main.go -s 10
Data sent to the server: {"Problem":"fizzbuzz","Cant":25,"Min":0,"Max":10,"Result":"","Shutdown":0,"Password":""}
[7 7 6 Fizz 4 4 8 8 2 2 11 11 6 Fizz 8 8 6 Fizz 7 7 8 8 0 FizzBuzz 7 7 4 4 4 4 5 Buzz 1 1 7 7 6 Fizz 3 Fizz 7 7 6 Fizz 2 2 5 Buzz 4 4]

D:\CUARTO SEMESTRE\APT\APTFinalProject\Client>
```

(go run main.go -r Y/y). Este en especifico salva los datos en results.txt

```
Connection accepted by: ('127.0.0.1', 50444)
Request received: {"Problem":"fizzbuzz","Cant":25,"Min":0,"Max":100,"Result":"y","Shutdown":0,"Password":""}
Resolving problem: fizzbuzz with 25 values between 0 and 100
Send response: {"Result": ["48 Fizz", "61 61", "114 Fizz", "49 49", "14 14", "51 Fizz", "29 29", "78 Fizz", "27 Fizz", "61 61", "88 88",
"43 43", "32 32", "84 Fizz", "83 83", "51 Fizz", "37 37", "45 FizzBuzz", "50 Buzz", "74 74", "59 59", "85 Buzz", "76 76", "35 Buzz", "118
 118"]}
Client socket closed
```

```
127.0.0.1 - - [06/Jun/2024 16:41:57] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 16:41:59] "POST /numbers HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:02:32] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:02:34] "POST /numbers HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:04:16] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:04:18] "POST /numbers HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:05:41] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:05:43] "POST /numbers HTTP/1.1" 200 -
```

Símbolo del sistema C

```
 22 semiprime 60 No classified 3 prime 15 semiprime 97 prime 70 No classified 50 No classified 25 cuadratic-semiprime 26 semiprime 89 pri
me 44 No classified 58 semiprime 44 No classified 74 semiprime 58 semiprime 60 No classified]

D:\CUARTO SEMESTRE\APT\APTFinalProject\Client>go run main.go -q 10
Data sent to the server: {"Problem":"fizzbuzz","Cant":10,"Min":0,"Max":100,"Result":"","Shutdown":0,"Password":""}
[88 88 63 Fizz 8 8 71 71 16 16 60 FizzBuzz 83 83 43 43 35 Buzz 54 Fizz]

D:\CUARTO SEMESTRE\APT\APTFinalProject\Client>go run main.go -l 90
Data sent to the server: {"Problem":"fizzbuzz","Cant":25,"Min":90,"Max":100,"Result":"","Shutdown":0,"Password":""}
[90 FizzBuzz 96 Fizz 95 Buzz 94 94 93 Fizz 91 91 92 92 97 97 91 91 94 94 97 97 99 Fizz 93 Fizz 99 Fizz 97 97 93 Fizz 95 Buzz 94 94 93 Fiz
z 97 97 93 Fizz 96 Fizz 93 Fizz 98 98 95 Buzz]

D:\CUARTO SEMESTRE\APT\APTFinalProject\Client>go run main.go -s 10
Data sent to the server: {"Problem":"fizzbuzz","Cant":25,"Min":0,"Max":10,"Result":"","Shutdown":0,"Password":""}
[7 7 6 Fizz 4 4 8 8 2 2 11 11 6 Fizz 8 8 6 Fizz 7 7 8 8 0 FizzBuzz 7 7 4 4 4 4 5 Buzz 1 1 7 7 6 Fizz 3 Fizz 7 7 6 Fizz 2 2 5 Buzz 4 4]

D:\CUARTO SEMESTRE\APT\APTFinalProject\Client>go run main.go -r y
Data sent to the server: {"Problem":"fizzbuzz","Cant":25,"Min":0,"Max":100,"Result":"y","Shutdown":0,"Password":""}
[48 Fizz 61 61 114 Fizz 49 49 14 14 51 Fizz 29 29 78 Fizz 27 Fizz 61 61 88 88 43 43 32 32 84 Fizz 83 83 51 Fizz 37 37 45 FizzBuzz 50 Buzz
 74 74 59 59 85 Buzz 76 76 35 Buzz 118 118]
File results.txt Created.
Your results are on the results.txt file.
```

Client > results.txt

```
48 Fizz
61 61
114 Fizz
49 49
14 14
51 Fizz
29 29
78 Fizz
27 Fizz
61 61
88 88
43 43
32 32
84 Fizz
83 83
51 Fizz
37 37
45 FizzBuzz
50 Buzz
74 74
59 59
85 Buzz
76 76
35 Buzz
118 118
```

Por ultimo se probarán con 10 datos todos los problemas guardando los datos en results.txt

Fizzbuzz:

```
Connection accepted by: ('127.0.0.1', 50510)
Request received: {"Problem":"fizzbuzz","Cant":10,"Min":0,"Max":100,"Result":"y","Shutdown":0,"Password":""}
Resolving problem: fizzbuzz with 10 values between 0 and 100
Send response: {"Result": ["2 2", "39 Fizz", "70 Buzz", "16 16", "37 37", "23 23", "99 Fizz", "76 76", "49 49", "35 Buzz"]}
Client socket closed
```

```
127.0.0.1 - - [06/Jun/2024 17:02:32] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:02:34] "POST /numbers HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:04:16] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:04:18] "POST /numbers HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:05:41] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:05:43] "POST /numbers HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:16:05] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:16:07] "POST /numbers HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:16:45] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:16:47] "POST /numbers HTTP/1.1" 200 -
```

```
Símbolo del sistema C
z 97 97 93 Fizz 96 Fizz 93 Fizz 98 98 95 Buzz]

D:\CUARTO SEMESTRE\APT\APTFinalProject\Client>go run main.go -s 10
Data sent to the server: {"Problem":"fizzbuzz","Cant":25,"Min":0,"Max":10,"Result":"","Shutdown":0,"Password":""}
[7 7 6 Fizz 4 4 8 8 2 2 11 11 6 Fizz 8 8 6 Fizz 7 7 8 8 0 FizzBuzz 7 7 4 4 4 4 5 Buzz 1 1 7 7 6 Fizz 3 Fizz 7 7 6 Fizz 2 2 5 Buzz 4 4]

D:\CUARTO SEMESTRE\APT\APTFinalProject\Client>go run main.go -r y
Data sent to the server: {"Problem":"fizzbuzz","Cant":25,"Min":0,"Max":100,"Result":"y","Shutdown":0,"Password":""}
[48 Fizz 61 61 114 Fizz 49 49 14 14 51 Fizz 29 29 78 Fizz 27 Fizz 61 61 88 88 43 43 32 32 84 Fizz 83 83 51 Fizz 37 37 45 FizzBuzz 50 Buzz
74 74 59 59 85 Buzz 76 76 35 Buzz 118 118]
File results.txt Created.
Your results are on the results.txt file.

D:\CUARTO SEMESTRE\APT\APTFinalProject\Client>go run main.go -p fizzbuzz -q 10
Data sent to the server: {"Problem":"fizzbuzz","Cant":10,"Min":0,"Max":100,"Result":"","Shutdown":0,"Password":""}
[40 Buzz 27 Fizz 11 11 35 Buzz 17 17 68 68 79 79 107 107 92 92 45 FizzBuzz]

D:\CUARTO SEMESTRE\APT\APTFinalProject\Client>go run main.go -p fizzbuzz -q 10 -r y
Data sent to the server: {"Problem":"fizzbuzz","Cant":10,"Min":0,"Max":100,"Result":"y","Shutdown":0,"Password":""}
[2 2 39 Fizz 70 Buzz 16 16 37 37 23 23 99 Fizz 76 76 49 49 35 Buzz]
File results.txt Created.
Your results are on the results.txt file.
```

```
1   2 2
2   39 Fizz
3   70 Buzz
4   16 16
5   37 37
6   23 23
7   99 Fizz
8   76 76
9   49 49
10  35 Buzz
```

Fibonacci:

```
Connection accepted by: ('127.0.0.1', 50525)
Request received: {"Problem":"fibonacci","Cant":10,"Min":0,"Max":100,"Result":"y","Shutdown":0,"Password":""}
Resolving problem: fibonacci with 10 values between 0 and 100
Send response: {"Result": ["96 False", "9 False", "31 False", "93 False", "76 False", "56 False", "97 False", "11 False", "7 False", "94
False"]}
Client socket closed
```

```
Símbolo del sistema C

D:\CUARTO SEMESTRE\APT\APTFinalProject\Client>go run main.go -r y
Data sent to the server: {"Problem":"fizzbuzz","Cant":25,"Min":0,"Max":100,"Result":"y","Shutdown":0,"Password":""}
[48 Fizz 61 61 114 Fizz 49 49 14 14 51 Fizz 29 29 78 Fizz 27 Fizz 61 61 88 88 43 43 32 32 84 Fizz 83 83 51 Fizz 37 37 45 FizzBuzz 50 Buzz
 74 74 59 59 85 Buzz 76 76 35 Buzz 118 118]
File results.txt Created.
Your results are on the results.txt file.

D:\CUARTO SEMESTRE\APT\APTFinalProject\Client>go run main.go -p fizzbuzz -q 10
Data sent to the server: {"Problem":"fizzbuzz","Cant":10,"Min":0,"Max":100,"Result":"","Shutdown":0,"Password":""}
[40 Buzz 27 Fizz 11 11 35 Buzz 17 17 68 68 79 79 107 107 92 92 45 FizzBuzz]

D:\CUARTO SEMESTRE\APT\APTFinalProject\Client>go run main.go -p fizzbuzz -q 10 -r y
Data sent to the server: {"Problem":"fizzbuzz","Cant":10,"Min":0,"Max":100,"Result":"y","Shutdown":0,"Password":""}
[2 2 39 Fizz 70 Buzz 16 16 37 37 23 23 99 Fizz 76 76 49 49 35 Buzz]
File results.txt Created.
Your results are on the results.txt file.

D:\CUARTO SEMESTRE\APT\APTFinalProject\Client>go run main.go -p fibonacci -q 10 -r y
Data sent to the server: {"Problem":"fibonacci","Cant":10,"Min":0,"Max":100,"Result":"y","Shutdown":0,"Password":""}
[96 False 9 False 31 False 93 False 76 False 56 False 97 False 11 False 7 False 94 False]
File results.txt Created.
Your results are on the results.txt file.

D:\CUARTO SEMESTRE\APT\APTFinalProject\Client>
```

```
127.0.0.1 - - [06/Jun/2024 16:41:59] "POST /numbers HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:02:32] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:02:34] "POST /numbers HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:04:16] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:04:18] "POST /numbers HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:05:41] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:05:43] "POST /numbers HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:16:05] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:16:07] "POST /numbers HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:16:45] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:16:47] "POST /numbers HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:18:10] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:18:13] "POST /numbers HTTP/1.1" 200 -
```

```
96 False
9 False
31 False
93 False
76 False
56 False
97 False
11 False
7 False
94 False
```

Prime:

```
Connection accepted by: ('127.0.0.1', 50538)
Request received: {"Problem":"prime","Cant":10,"Min":0,"Max":100,"Result":"y","Shutdown":0,"Password":""}
Resolving problem: prime with 10 values between 0 and 100
Send response: {"Result": ["14 semiprime", "11 prime", "66 No classified", "75 No classified", "62 semiprime", "87 semiprime", "33 semipr
ime", "5 prime", "88 No classified", "51 semiprime"]}
Client socket closed
```

```
Símbolo del sistema C

D:\CUARTO SEMESTRE\APT\APTFinalProject\Client>go run main.go -p fizzbuzz -q 10
Data sent to the server: {"Problem":"fizzbuzz","Cant":10,"Min":0,"Max":100,"Result":"","Shutdown":0,"Password":""}
[40 Buzz 27 Fizz 11 11 35 Buzz 17 17 68 68 79 79 107 107 92 92 45 FizzBuzz]

D:\CUARTO SEMESTRE\APT\APTFinalProject\Client>go run main.go -p fizzbuzz -q 10 -r y
Data sent to the server: {"Problem":"fizzbuzz","Cant":10,"Min":0,"Max":100,"Result":"y","Shutdown":0,"Password":""}
[2 2 39 Fizz 70 Buzz 16 16 37 37 23 23 99 Fizz 76 76 49 49 35 Buzz]
File results.txt Created.
Your results are on the results.txt file.

D:\CUARTO SEMESTRE\APT\APTFinalProject\Client>go run main.go -p fibonacci -q 10 -r y
Data sent to the server: {"Problem":"fibonacci","Cant":10,"Min":0,"Max":100,"Result":"y","Shutdown":0,"Password":""}
[96 False 9 False 31 False 93 False 76 False 56 False 97 False 11 False 7 False 94 False]
File results.txt Created.
Your results are on the results.txt file.

D:\CUARTO SEMESTRE\APT\APTFinalProject\Client>go run main.go -p prime -q 10 -r y
Data sent to the server: {"Problem":"prime","Cant":10,"Min":0,"Max":100,"Result":"y","Shutdown":0,"Password":""}
[14 semiprime 11 prime 66 No classified 75 No classified 62 semiprime 87 semiprime 33 semiprime 5 prime 88 No classified 51 semiprime]
File results.txt Created.
Your results are on the results.txt file.
```

```
127.0.0.1 - - [06/Jun/2024 16:41:59] "POST /numbers HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:02:32] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:02:34] "POST /numbers HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:04:16] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:04:18] "POST /numbers HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:05:41] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:05:43] "POST /numbers HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:16:05] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:16:07] "POST /numbers HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:16:45] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:16:47] "POST /numbers HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:18:10] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:18:13] "POST /numbers HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:19:29] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:19:31] "POST /numbers HTTP/1.1" 200 -
```

```
1   14 semiprime
2   11 prime
3   66 No classified
4   75 No classified
5   62 semiprime
6   87 semiprime
7   33 semiprime
8   5 prime
9   88 No classified
10  51 semiprime
```

Ahora se probará con respecto a shutdown

(go run main.go shutdown)

```
Connection accepted by: ('127.0.0.1', 50551)
Request received: {"Problem":"","Cant":0,"Min":0,"Max":0,"Result":"","Shutdown":1,"Password":"95ccedba61b3d42254e24e5e651e5141ac06a071479
296924e6e25d0e7882859"}
Shutdown request denied, incorrect password.
Reconnecting to client_socket...
Connection accepted by: ('127.0.0.1', 50551)
Client socket closed
```

```
Símbolo del sistema C

Data sent to the server: {"Problem":"fizzbuzz","Cant":10,"Min":0,"Max":100,"Result":"y","Shutdown":0,"Password":""}
[2 2 39 Fizz 70 Buzz 16 16 37 37 23 23 99 Fizz 76 76 49 49 35 Buzz]
File results.txt Created.
Your results are on the results.txt file.

D:\CUARTO SEMESTRE\APT\APTFinalProject\Client>go run main.go -p fibonacci -q 10 -r y
Data sent to the server: {"Problem":"fibonacci","Cant":10,"Min":0,"Max":100,"Result":"y","Shutdown":0,"Password":""}
[96 False 9 False 31 False 93 False 76 False 56 False 97 False 11 False 7 False 94 False]
File results.txt Created.
Your results are on the results.txt file.

D:\CUARTO SEMESTRE\APT\APTFinalProject\Client>go run main.go -p prime -q 10 -r y
Data sent to the server: {"Problem":"prime","Cant":10,"Min":0,"Max":100,"Result":"y","Shutdown":0,"Password":""}
[14 semiprime 11 prime 66 No classified 75 No classified 62 semiprime 87 semiprime 33 semiprime 5 prime 88 No classified 51 semiprime]
File results.txt Created.
Your results are on the results.txt file.

D:\CUARTO SEMESTRE\APT\APTFinalProject\Client>go run main.go shutdown
a password is require use -p 'password'
Data sent to the server: {"Problem":"","Cant":0,"Min":0,"Max":0,"Result":"","Shutdown":1,"Password":"95ccedba61b3d42254e24e5e651e5141ac06
a071479296924e6e25d0e7882859"}
[Incorrect password]
```

```
127.0.0.1 - - [06/Jun/2024 16:41:57] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 16:41:59] "POST /numbers HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:02:32] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:02:34] "POST /numbers HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:04:16] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:04:18] "POST /numbers HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:05:41] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:05:43] "POST /numbers HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:16:05] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:16:07] "POST /numbers HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:16:45] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:16:47] "POST /numbers HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:18:10] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:18:13] "POST /numbers HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:19:29] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:19:31] "POST /numbers HTTP/1.1" 200 -
```

(go run main.go shutdown –p buendia)

```
Connection accepted by: ('127.0.0.1', 50558)
Request received: {"Problem":"","Cant":0,"Min":0,"Max":0,"Result":"","Shutdown":1,"Password":"d44aa45204568f1065b576afb8e7e912340b256cdce
8abcddef2e6b202153bea"}
Shutdown request denied, incorrect password.
Reconnecting to client_socket...
Connection accepted by: ('127.0.0.1', 50558)
Client socket closed
```

```
Símbolo del sistema C

D:\CUARTO SEMESTRE\APT\APTFinalProject\Client>go run main.go -p fibonacci -q 10 -r y
Data sent to the server: {"Problem":"fibonacci","Cant":10,"Min":0,"Max":100,"Result":"y","Shutdown":0,"Password":""}
[96 False 9 False 31 False 93 False 76 False 56 False 97 False 11 False 7 False 94 False]
File results.txt Created.
Your results are on the results.txt file.

D:\CUARTO SEMESTRE\APT\APTFinalProject\Client>go run main.go -p prime -q 10 -r y
Data sent to the server: {"Problem":"prime","Cant":10,"Min":0,"Max":100,"Result":"y","Shutdown":0,"Password":""}
[14 semiprime 11 prime 66 No classified 75 No classified 62 semiprime 87 semiprime 33 semiprime 5 prime 88 No classified 51 semiprime]
File results.txt Created.
Your results are on the results.txt file.

D:\CUARTO SEMESTRE\APT\APTFinalProject\Client>go run main.go shutdown
a password is require use -p 'password'
Data sent to the server: {"Problem":"","Cant":0,"Min":0,"Max":0,"Result":"","Shutdown":1,"Password":"95ccedba61b3d42254e24e5e651e5141ac06
a071479296924e6e25d0e7882859"}
[Incorrect password]

D:\CUARTO SEMESTRE\APT\APTFinalProject\Client>go run main.go shutdown -p buendia
Data sent to the server: {"Problem":"","Cant":0,"Min":0,"Max":0,"Result":"","Shutdown":1,"Password":"d44aa45204568f1065b576afb8e7e912340b
256cdce8abcddef2e6b202153bea"}
[Incorrect password]
```

```
127.0.0.1 - - [06/Jun/2024 16:41:57] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 16:41:59] "POST /numbers HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:02:32] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:02:34] "POST /numbers HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:04:16] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:04:18] "POST /numbers HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:05:41] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:05:43] "POST /numbers HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:16:05] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:16:07] "POST /numbers HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:16:45] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:16:47] "POST /numbers HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:18:10] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:18:13] "POST /numbers HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:19:29] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:19:31] "POST /numbers HTTP/1.1" 200 -
```

(go run main.go shutdown –p admin). (Por un momento me asusté porque no validaba la contraseña pero era porque la habia escrito mal). Y como se puede ver a continuacion, se dejó de ejecutar cada archivo de todo el proyecto.

```
(env) D:\CUARTO SEMESTRE\APT\APTFinalProject\ProblemSolver>py main.py
Server listening at the port 8080...
Connection accepted by: ('127.0.0.1', 50610)
Request received: {"Problem":"","Cant":0,"Min":0,"Max":0,"Result":"","Shutdown":1,"Password":"6d2df7bd5b23426af87b65f44af0a7243b37649a9fe
312e29e65d21c2365f528"}
Shutdown request accepted.
Flask Connection Down Success
Client socket closed
Server socket off

(env) D:\CUARTO SEMESTRE\APT\APTFinalProject\ProblemSolver>
```

```
Símbolo del sistema C

649a9fe312e29e65d21c2365f528"}
[Incorrect password]

D:\CUARTO SEMESTRE\APT\APTFinalProject\Client>go run main.go shutdown -p admin
Data sent to the server: {"Problem":"","Cant":0,"Min":0,"Max":0,"Result":"","Shutdown":1,"Password":"6d2df7bd5b23426af87b65f44af0a7243b37
649a9fe312e29e65d21c2365f528"}
[Incorrect password]

D:\CUARTO SEMESTRE\APT\APTFinalProject\Client>go run main.go shutdown -p admi
Data sent to the server: {"Problem":"","Cant":0,"Min":0,"Max":0,"Result":"","Shutdown":1,"Password":"151ec987be8e1f8807b04239fbb7eea6be13
386b2a3f629cc50ad6dbe595359e"}
[Incorrect password]

D:\CUARTO SEMESTRE\APT\APTFinalProject\Client>go run main.go shutdown -p admin
Data sent to the server: {"Problem":"","Cant":0,"Min":0,"Max":0,"Result":"","Shutdown":1,"Password":"6d2df7bd5b23426af87b65f44af0a7243b37
649a9fe312e29e65d21c2365f528"}
[Incorrect password]

D:\CUARTO SEMESTRE\APT\APTFinalProject\Client>go run main.go shutdown -p admin
Data sent to the server: {"Problem":"","Cant":0,"Min":0,"Max":0,"Result":"","Shutdown":1,"Password":"6d2df7bd5b23426af87b65f44af0a7243b37
649a9fe312e29e65d21c2365f528"}
[System down]

D:\CUARTO SEMESTRE\APT\APTFinalProject\Client>
```

```
127.0.0.1 - - [06/Jun/2024 16:23:41] "POST /numbers HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 16:38:50] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 16:38:53] "POST /numbers HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 16:41:57] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 16:41:59] "POST /numbers HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:02:32] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:02:34] "POST /numbers HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:04:16] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:04:18] "POST /numbers HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:05:41] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:05:43] "POST /numbers HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:16:05] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:16:07] "POST /numbers HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:16:45] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:16:47] "POST /numbers HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:18:10] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:18:13] "POST /numbers HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:19:29] "POST /key HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 17:19:31] "POST /numbers HTTP/1.1" 200 -

(env) D:\CUARTO SEMESTRE\APT\APTFinalProject\DataServer>
```

Fase BLUE

Para hacer refactoring al código se usará la herramienta pylint para mejorar la calidad del producto FinalProject. Posterior a la primera calificación se modificará cada carpeta con archivos .py por separado para mejorar esta calificación y se mostrará esta calificación final del producto.

Como calificación inicial del producto se tiene 3.42

```
------------------------------------------------------------------
Your code has been rated at 3.42/10 (previous run: 3.42/10, +0.00)
```

Calificacion despues de mejorar la carpeta DataServer: 5.30

```
------------------------------------------------------------------
Your code has been rated at 5.30/10 (previous run: 3.42/10, +1.88)
```

Calificacicion despues de mejorar la carpeta ProblemSolver: 9.11.

```
------------------------------------------------------------------
Your code has been rated at 9.11/10 (previous run: 5.30/10, +3.81)
```

Metacognición

Aprendí RSA y AES, claramente lo basico, pero es interesante para iniciar con algo de ciberseguridad.

Mejoré mi manejo del lenguaje Golang, es realmente muy buen lenguaje.

Aprendí a leer mejor la documentación oficial de las librerías mas que todo cobra.

Me gustó el manejo de las excepciones para siempre saber a manera de prueba, que está pasando con el código en diferentes situaciones.

Por ultimo, me gusta que de manera implicita al programar (probablemente quedé traumado), ya pienso en como darle mas que todo, mantenibilidad y seguridad al código.