📖 Report

Zocket Golang Assignment

Name: Mukund Deepak

Note:

Hi there! Please note that I have no experience writing testing reports but this pretty much covers all the functionality and gives you an understanding of the Application.

I have also encapsulated everything into a monolith which is not a good practice, but i assure you, in production, I will use modular programming and microservice architecture to decouple and make a clean codebase.

Problem Statement:

I have created as per request, an API which has functionality to add products to the DB and then sends the product_id to the Rabbit MQ message queue initialized from which the consumer who is always listening for new messages in the queue, recieves the PID, get's the image_urls of that particular product from the DB and then downloads each of the image and adds the compressed equivalent to the local directory and adds the path to the compressed image path array in the DB for that particular product. This application is a very basic working model, therefore it only works with jpegs as of now.

Code:

app.go (This is the API):

```
package main

import (
        "fmt"
        "context"
        "strconv"
        "gorm.io/driver/postgres"
```

```
  "gorm.io/gorm"
        "github.com/gin-gonic/gin"
        "github.com/lib/pq"
        amqp "github.com/rabbitmq/amqp091-go"
        "time"
)
type Product struct{
        Product_ID int `gorm:"primaryKey" gorm:"column:product_id"`
        Product_Name string `gorm:"column:product_name"`
        Product_Description string `gorm:"column:product_description"`
        Product_Images pq.StringArray `gorm:"type:text[]" gorm:"column:pr
        Product_Price float64 `gorm:"column:product_price"`
        Compressed_Product_Images pq.StringArray `gorm:"type:text[]" gorr
        Created_At time.Time `gorm:"column:created_at"`
        Updated_At time.Time `gorm:"column:updated_at"`
}
type User struct{
        ID int `gorm:"column:id" gorm:"primaryKey"`
        Name string `gorm:"column:name"`
        Mobile int64 `gorm:"column:mobile"`
        Latitude float64 `gorm:"column:latitude"`
        Longitude float64 `gorm:"column:latitude"`
        Created_At time.Time `gorm:"column:created_at"`
        Updated_At time.Time `gorm:"column:updated_at"`
}
//Input JSON structure
type Input struct{
  User_ID   int `json:"user_id"`
  Product_Name  string `json:"product_name"`
  ProductDescription  string `json:"product_description"`
  ProductImages []string `json:"product_images"`
  ProductPrice  float64 `json:"product_price"`
}
func main(){
        //Database config
        dsn := "host=localhost user=postgres dbname=prod_db port=5432 ss
        db, err := gorm.Open(postgres.Open(dsn), &gorm.Config{})
        if err!=nil{
                fmt.Println(err)
                panic(err)
        }else{
                fmt.Println("Database Connected Successfully!")
        }
        //RabbitMQ config
        conn,err := amqp.Dial("amqp://guest:guest@localhost:5672")
        if err!=nil{
                fmt.Println(err)
```

```
                panic(err)
        }
        defer conn.Close()
        //fmt.Println(conn)
        fmt.Println("Successfully connected to RabbitMQ!")

        ch,err := conn.Channel()
        if err!=nil{
                fmt.Println(err)
                panic(err)
        }
        defer ch.Close()

        q, err := ch.QueueDeclare(
                "product",
                false,
                false,
                false,
                false,
                nil,
        )
        if err!=nil{
                fmt.Println(err)
                panic(err)
        }

        ctx, cancel := context.WithTimeout(context.Background(), 5*time.S
        defer cancel()

        db.AutoMigrate(&Product{}, &User{})
        fmt.Println("All schemas, tables, et cetera have been migrated")
r := gin.Default()
r.GET("/status", func(c *gin.Context){
  c.JSON(200, gin.H{"status":"Up and Running!",})
})
r.POST("/addProduct", func(c *gin.Context){
  var i Input
  err := c.BindJSON(&i);
  if err!=nil{
    c.JSON(400, gin.H{"error":err,})
  }
  //fmt.Printf("%T\n", i.ProductImages)
  prod := Product{Product_Name:i.Product_Name, Product_Description:i.Pr
  db.Save(&prod)
  c.JSON(200, "Product Added!")
  fmt.Println("Product ID:",prod.Product_ID)
  //fmt.Printf("%T\n", prod.Product_Images)
```

```
    //For testing purposes, i used the below code.
    /*fmt.Printf("User ID:%d\nProduct Name:%s\nProduct Description:%s\nP
    for _,value := range i.ProductImages{
      fmt.Printf("%s\n",value)
    }*/
    body := prod.Product_ID
                err = ch.PublishWithContext(
                        ctx,
                        "",
                        q.Name,
                        false,
                        false,
                        amqp.Publishing {
                                ContentType: "text/plain",
                                Body: []byte(strconv.Itoa(body)),
                        },
                )
                if err!=nil{
                        fmt.Println(err)
                        panic(err)
                }
                fmt.Printf("[x] Sent %d\n", body)
  })
  r.Run(":5000")
}
```

consumer.go:

```
package main

import(
  "fmt"
  "log"
  "strconv"
  "strings"
  "os"
  "image"
  "image/jpeg"
  "time"
  "net/http"
  "gorm.io/driver/postgres"
  "gorm.io/gorm"
  "github.com/lib/pq"
  "github.com/nfnt/resize"
  amqp "github.com/rabbitmq/amqp091-go"
)
```

```go
func extractImageName(url string) string {
        parts := strings.Split(url, "/")
        filename := parts[len(parts)-1]
        return strings.TrimSuffix(filename, ".jpg")
}
func DownloadAndCompress(urls []string) ([]string){
  var final_arr []string
  for _, url := range urls{
    response, err := http.Get(url)
    if err!=nil{
      fmt.Println("Failed to retrieve image!")
    }
    defer response.Body.Close()
    img_name := extractImageName(url)
    _, err = os.Stat("compressed_images")
    if os.IsNotExist(err){
      err:=os.Mkdir("compressed_images", 0755)
      if err!=nil{
        fmt.Println(err)
        panic(err)
      }
    }
    file, err := os.Create("compressed_images/"+img_name+"_compressed.jpg
    if err!=nil{
      fmt.Println(err)
      panic(err)
    }
    defer file.Close()
    var img image.Image
    img, err = jpeg.Decode(response.Body)
    if err!=nil{
      fmt.Println(err)
      panic(err)
    }
    newImg := resize.Thumbnail(200,200,img,resize.Lanczos3)
    err = jpeg.Encode(file, newImg, nil)
    if err!=nil{
      fmt.Println(err)
      panic(err)
    }

    final_arr = append(final_arr, file.Name())
  }
  return final_arr
}
//Product Model for Retrieval
type Product struct{
```

```
        Product_ID int `gorm:"primaryKey" gorm:"column:product_id"`
        Product_Name string `gorm:"column:product_name"`
        Product_Description string `gorm:"column:product_description"`
        Product_Images pq.StringArray `gorm:"type:text[]" gorm:"column:pr
        Product_Price float64 `gorm:"column:product_price"`
        Compressed_Product_Images pq.StringArray `gorm:"type:text[]" gorm
        Created_At time.Time `gorm:"column:created_at"`
        Updated_At time.Time `gorm:"column:updated_at"`
}
func main(){
  dsn:="host=localhost user=postgres dbname=prod_db port=5432 sslmode=dis
  db, err := gorm.Open(postgres.Open(dsn))
  if err!=nil{
    fmt.Println(err)
    panic(err)
  }else{
    fmt.Println("Successfully connected to Postgres DB")
  }

  //RabbitMQ connection
  conn, err := amqp.Dial("amqp://guest:guest@localhost:5672")
  defer conn.Close()
  if err!=nil{
    fmt.Println(err)
    panic(err)
  }else{
    fmt.Println("RabbitMQ connection Successful!")
  }
  ch,err := conn.Channel()
  if err!=nil{
    fmt.Println(err)
    panic(err)
  }
  defer ch.Close()

  q, err := ch.QueueDeclare(
    "product",
    false,
    false,
    false,
    false,
    nil,
  )
  if err!=nil{
    fmt.Println(err)
    panic(err)
  }
```

```
  msgs, err := ch.Consume(
    q.Name,
    "",
    true,
    false,
    false,
    false,
    nil,
  )
  if err!=nil{
    fmt.Println(err)
    panic(err)
  }

  var forever chan struct{}

  go func(){
    for d:=range msgs{
      log.Printf("Recieved a PID: %s\n", d.Body);
      var prod Product
      id, err := strconv.Atoi(string(d.Body))
      if err!=nil{
        fmt.Println(err)
        panic(err)
      }
      db.Where("product_id = ?", id).First(&prod)
      compressed_li := DownloadAndCompress([]string(prod.Product_Images))
      prod.Compressed_Product_Images = pq.StringArray(compressed_li)
      db.Save(&prod)

      fmt.Println(prod)
    }
  }()

  fmt.Println("[*] Waiting for Messages from queue... To exit press Ctrl-
  <-forever
}
```

Testing:

Unit Testing:

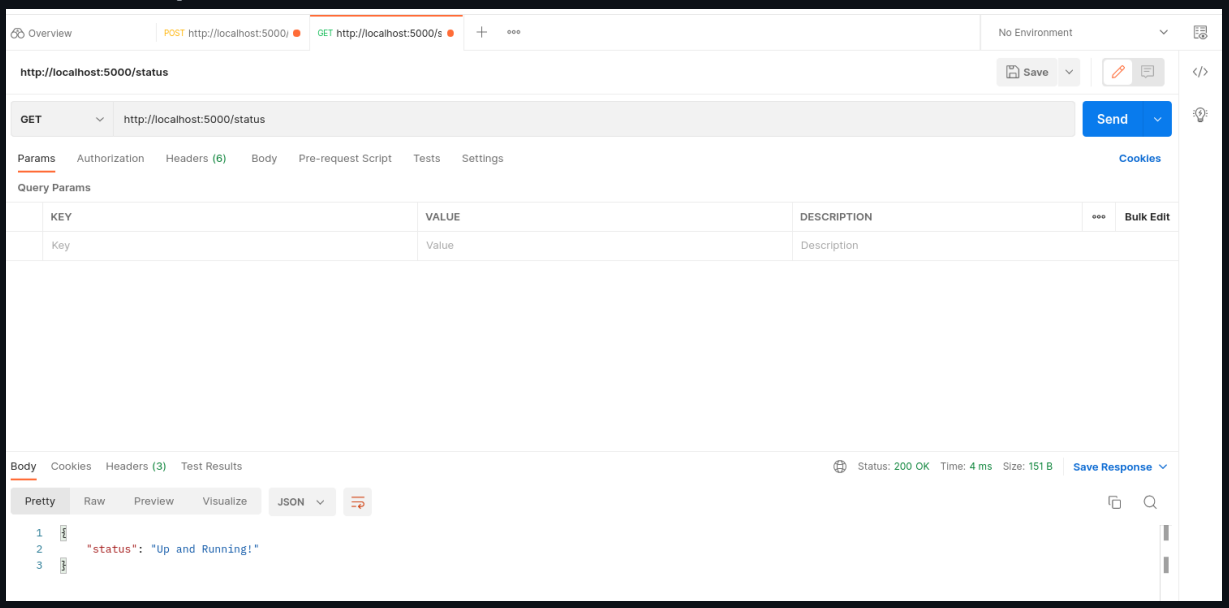
API:

API requirements: Design an API where it should receive a product data and store in the database, below are the parameters that should be passed in the API

- user_id (create users table and primary key of that table)
- product_name
- product_description (text)
- product_images (array of image urls)
- product_price (Number)

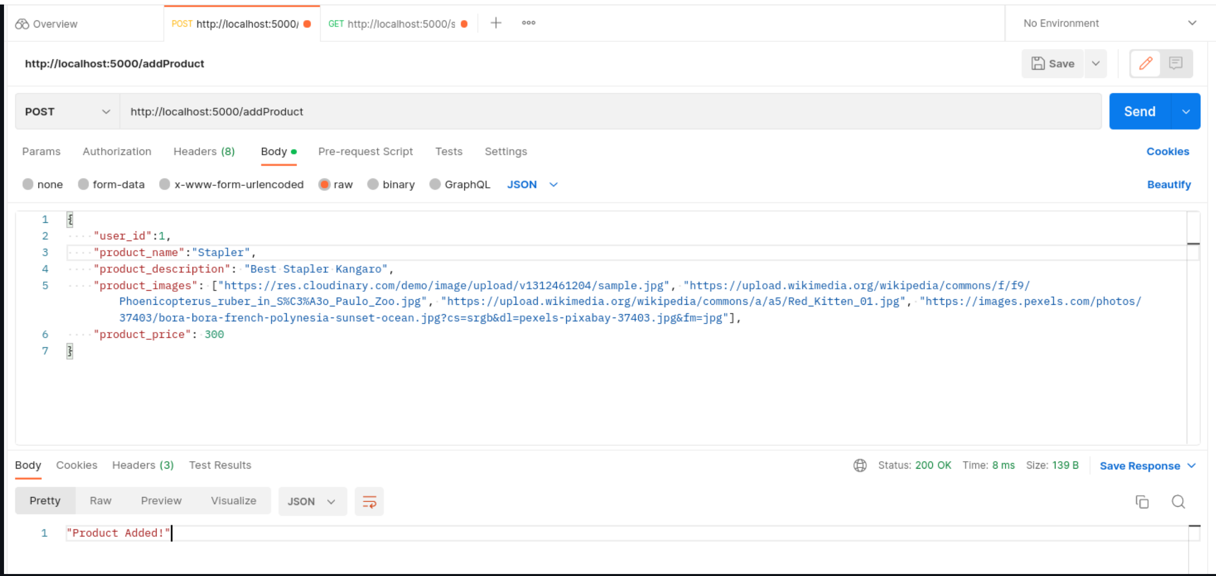
The API has two endpoints as of now: /addProduct (Used for adding the products to DB) /status (Used for checking running status of API)

/status endpoint:



/addProduct endpoint: The addProduct endpoint according to API Requirements need to add product details to DB aka populate it. But in this aspect, product also acts as a **producer**, so it should add the product ID to the queue.

Populating the DB:



DB Verification:

```
 product_id | product_name | product_description  |                                                                                                           product_i
mages                                                                                                        | product_price | compressed_product_images |
created_at                | updated_at
------------+--------------+----------------------+-----------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------+---------------+---------------------------+----------
-------------------+---------------------------
          1 | Stapler      | Best Stapler Kangaro | {https://res.cloudinary.com/demo/image/upload/v1312461204/sample.j
pg,https://upload.wikimedia.org/wikipedia/commons/f/f9/Phoenicopterus_ruber_in_S%C3%A3o_Paulo_Zoo.jpg,https://upload.w
ikimedia.org/wikipedia/commons/a/a5/Red_Kitten_01.jpg,https://images.pexels.com/photos/37403/bora-bora-french-polynesi
a-sunset-ocean.jpg?cs=srgb&dl=pexels-pixabay-37403.jpg&fm=jpg} |           300 |                           | 0001-01-0
1 05:53:28+05:53:28 | 0001-01-01 05:53:28+05:53:28
(1 row)
```

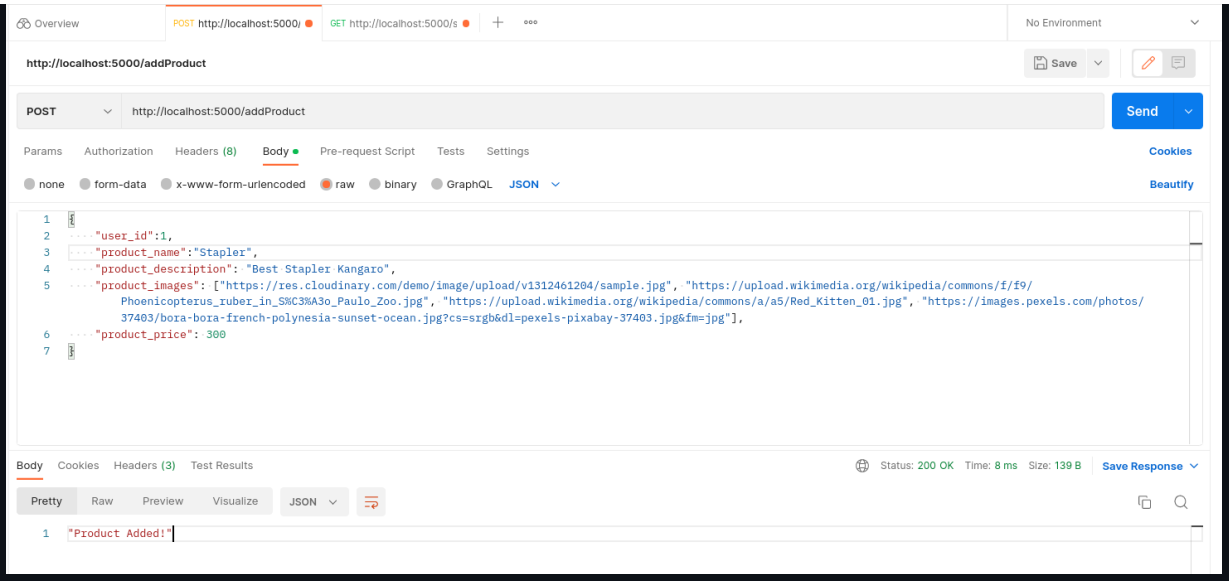
If table does not exists before, as soon as app.go is ran, it takes care of creation and migration of models i.e. creating the tables.

Producer:

As mentioned earlier, API acts as producer and adds the newly created product's id into the queue.

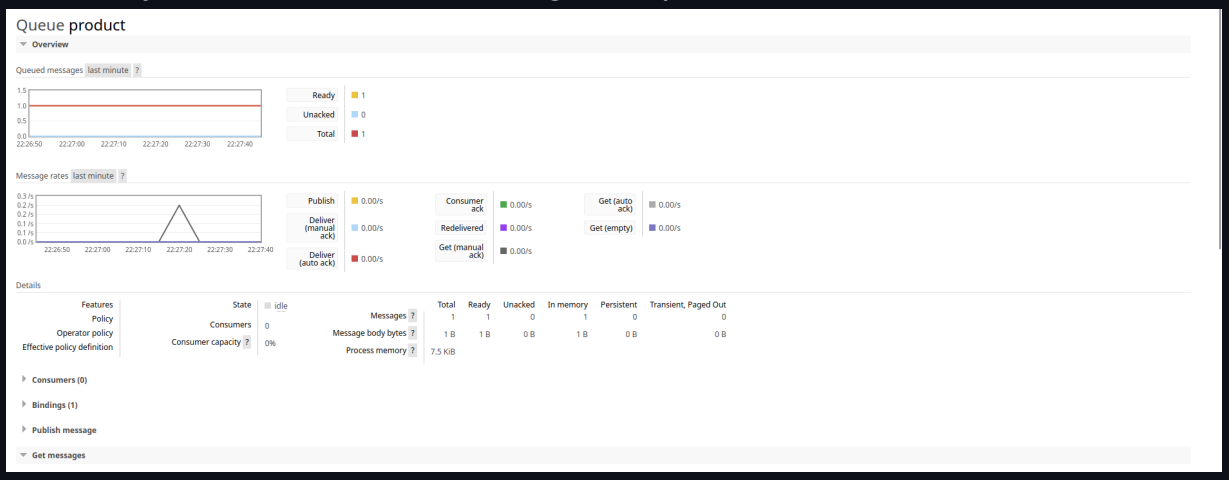
Adding product id to products queue: A queue called products is created under a channel which can be seen in the RabbitMQ Management Portal.

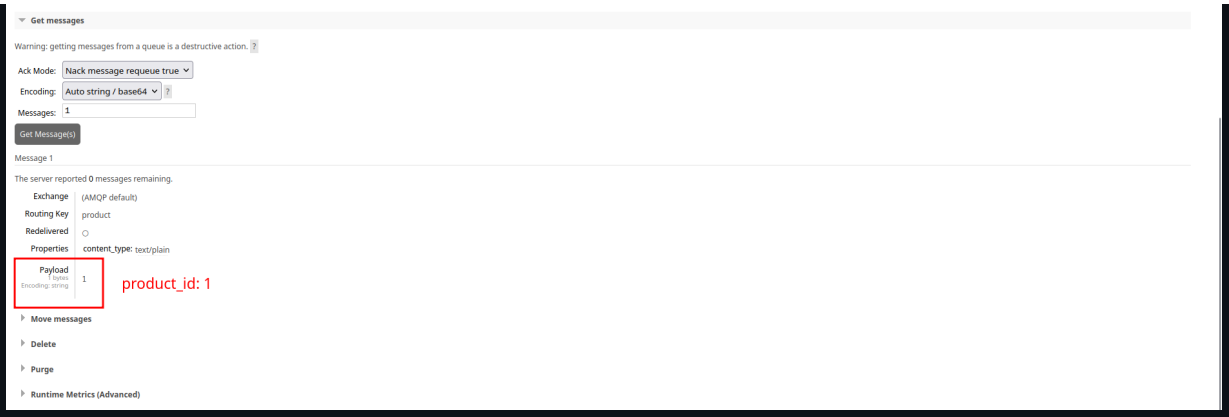
Adding the product to DB and pushing product id to message queue:




```
mukund@mukund-bravo15a4ddr ~/assignment  main ± go run ./app.go
Database Connected Successfully!
Successfully connected to RabbitMQ!
All schemas, tables, et cetera have been migrated
[GIN-debug] [WARNING] Creating an Engine instance with the Logger and Recovery middleware already attached.

[GIN-debug] [WARNING] Running in "debug" mode. Switch to "release" mode in production.
 - using env:   export GIN_MODE=release
 - using code:  gin.SetMode(gin.ReleaseMode)

[GIN-debug] GET    /status                   --> main.main.func1 (3 handlers)
[GIN-debug] POST   /addProduct               --> main.main.func2 (3 handlers)
[GIN-debug] [WARNING] You trusted all proxies, this is NOT safe. We recommend you to set a value.
Please check https://pkg.go.dev/github.com/gin-gonic/gin#readme-don-t-trust-all-proxies for details.
[GIN-debug] Listening and serving HTTP on :5000
Product ID: 1
[x] Sent 1
[GIN] 2023/05/24 - 22:25:37 | 200 |    40.76558ms |             ::1 | POST     "/addProduct"
```

Let’s verify this in the RabbitMQ management portal:



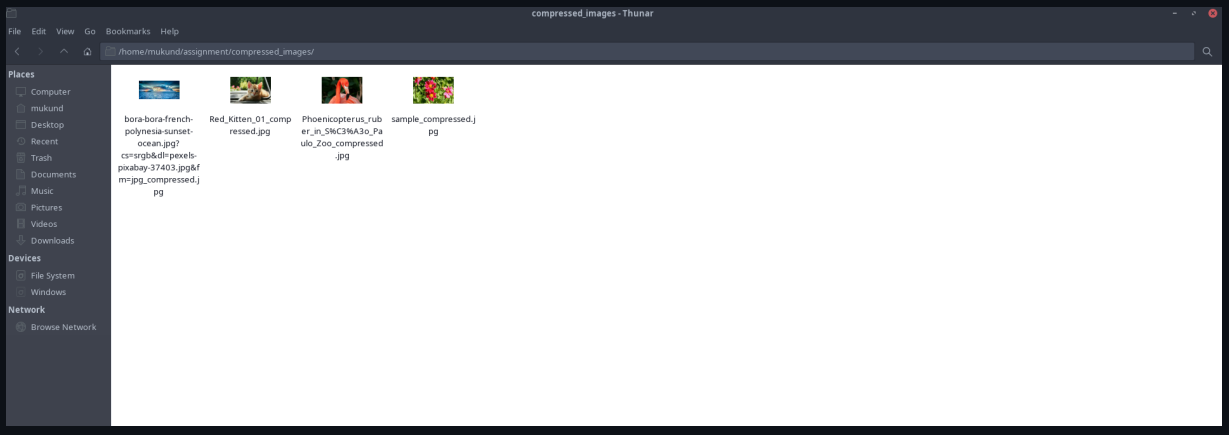


Consumer:

The consumer keeps listening for new messages in the products queue, retrieves the product id and obtains the image url array and passes it to the DownloadAndCompress function which retrieves all the images, compresses it and saves it in a compressed jpeg file locally inside the compressed_images folder and adds the path to the compressed_product_images array which is updated to the DB at the end.

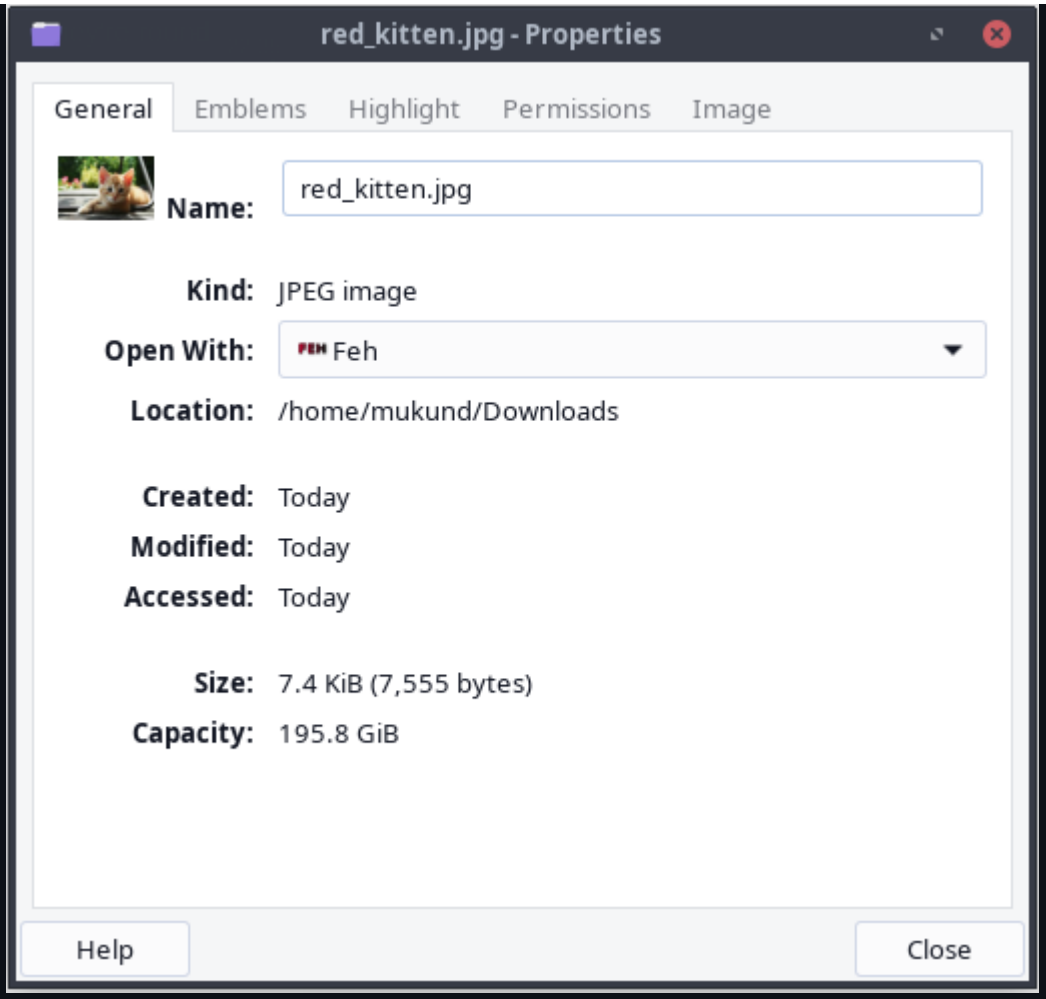
Consumer pulling a product_id and printing the array of compressed_product_images after saving in local:

Saved in compressed_images folder (Verification):

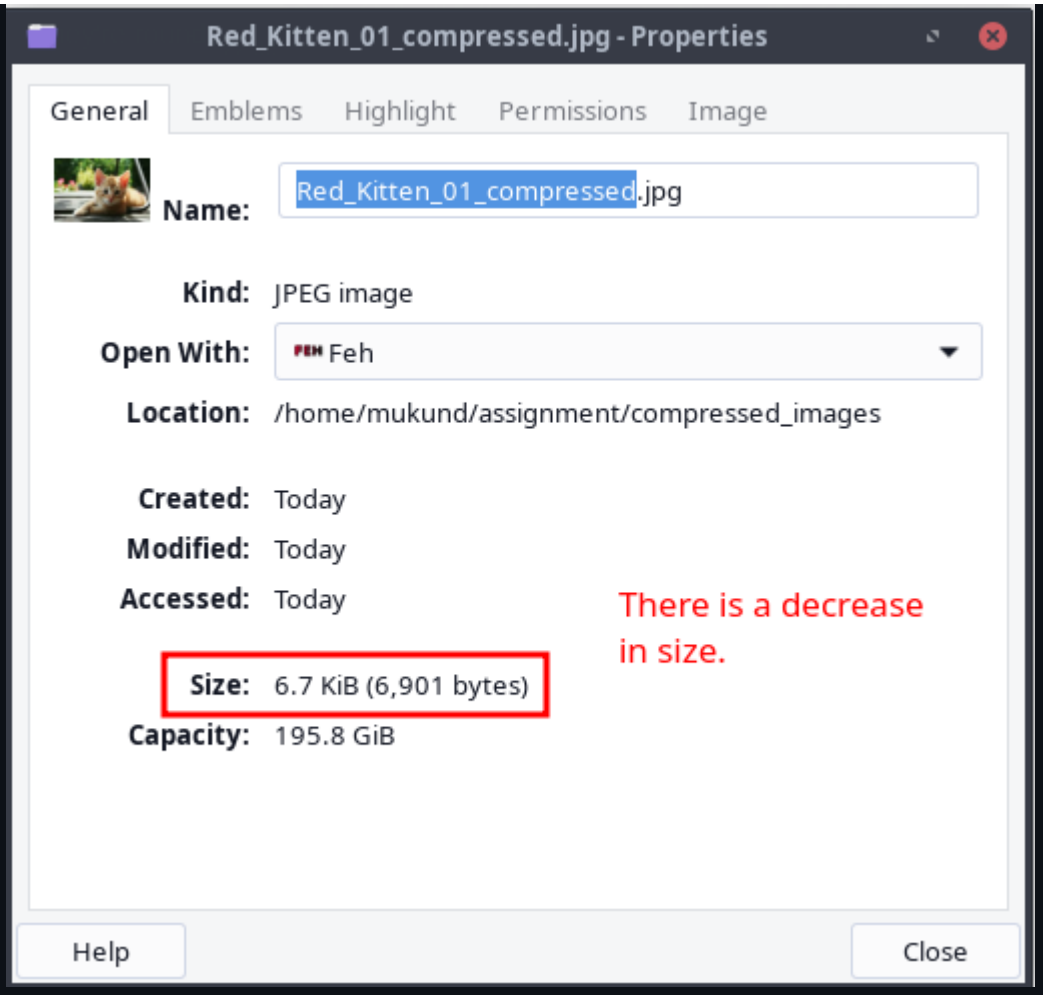


Size Comparision: We will take the red kitten picture to compare.

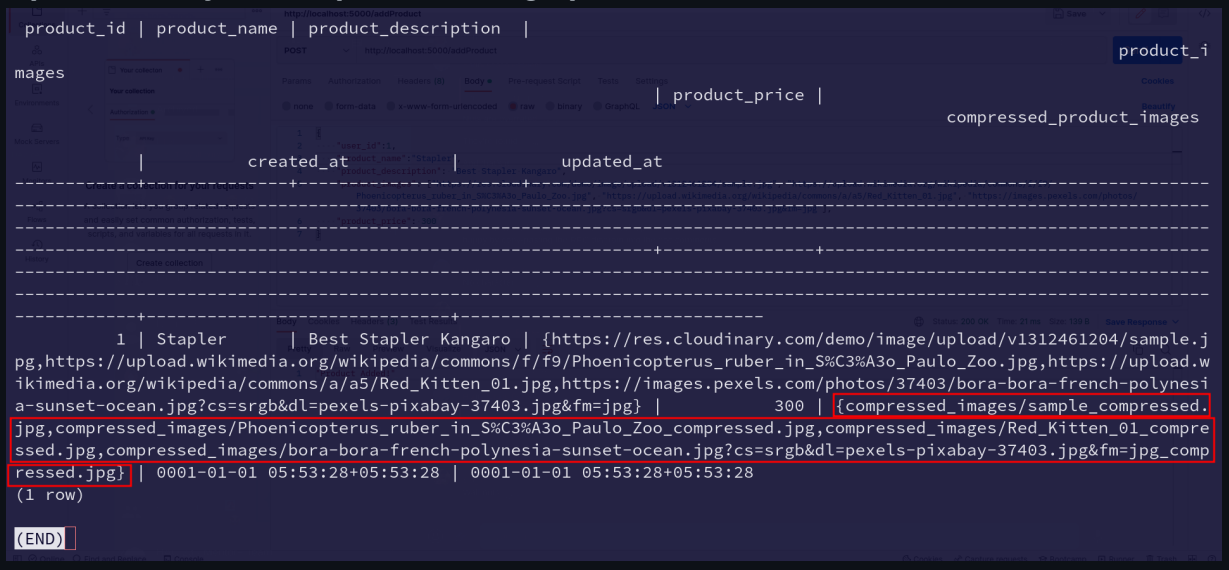
Before compression:



After compression:



Updated array of compressed image path in DB:



Populating User table manually in psql:

Inserting values into table users:

```
prod_db=# INSERT INTO users (name,mobile,latitude,longitude,created_at,updated_at) VALUES('Mukund',6366072930,4.8011,1
30.335,current_timestamp,current_timestamp);
INSERT 0 1
prod_db=# INSERT INTO users (name,mobile,latitude,longitude,created_at,updated_at) VALUES('Meghana',9663779956,3.120,1
92.123,current_timestamp,current_timestamp);
INSERT 0 1
prod_db=# INSERT INTO users (name,mobile,latitude,longitude,created_at,updated_at) VALUES('Rema',9880055663,100.12,2.1
13,current_timestamp,current_timestamp);
INSERT 0 1
```

Verification of Addition of records into table users:

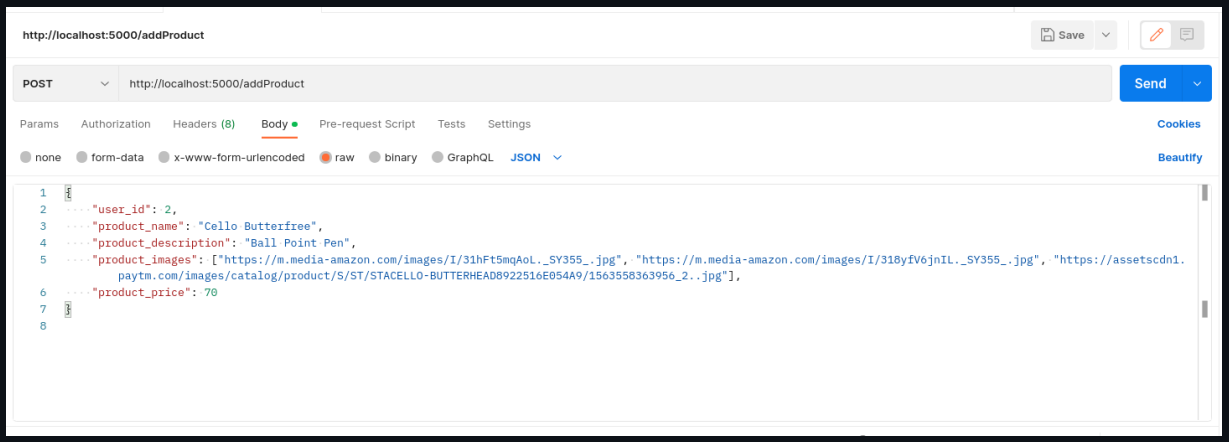
```
 id |  name   |   mobile   | latitude | longitude |          created_at           |          updated_at
----+---------+------------+----------+-----------+-------------------------------+-------------------------------
--
  1 | Mukund  | 6366072930 |   4.8011 |   130.335 | 2023-05-25 09:20:53.805853+05:30 | 2023-05-25 09:20:53.805853+05:3
0
  2 | Meghana | 9663779956 |    3.120 |   192.123 | 2023-05-25 09:21:41.917721+05:30 | 2023-05-25 09:21:41.917721+05:3
0
  3 | Rema    | 9880055663 |   100.12 |     2.113 | 2023-05-25 09:22:14.424545+05:30 | 2023-05-25 09:22:14.424545+05:3
0
(3 rows)

(END)
```

Integration Testing:

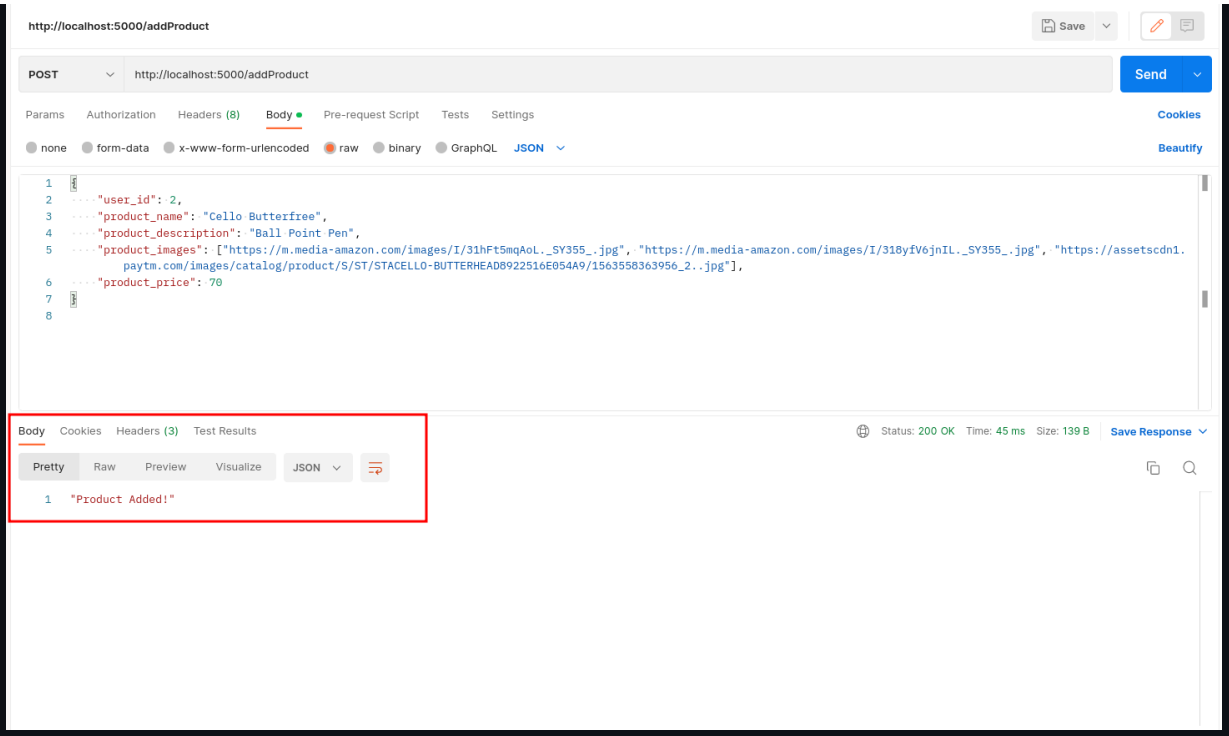
Ok so we shall add the following data to the Products DB through the API:

```
{
    "user_id": 2,
    "product_name": "Cello Butterfree",
    "product_description": "Ball Point Pen",
    "product_images": ["https://m.media-amazon.com/images/I/31hFt5mqAoL._
    "product_price": 70
}
```

Postman Request:



We successfully send the postman request and get the response as follows:



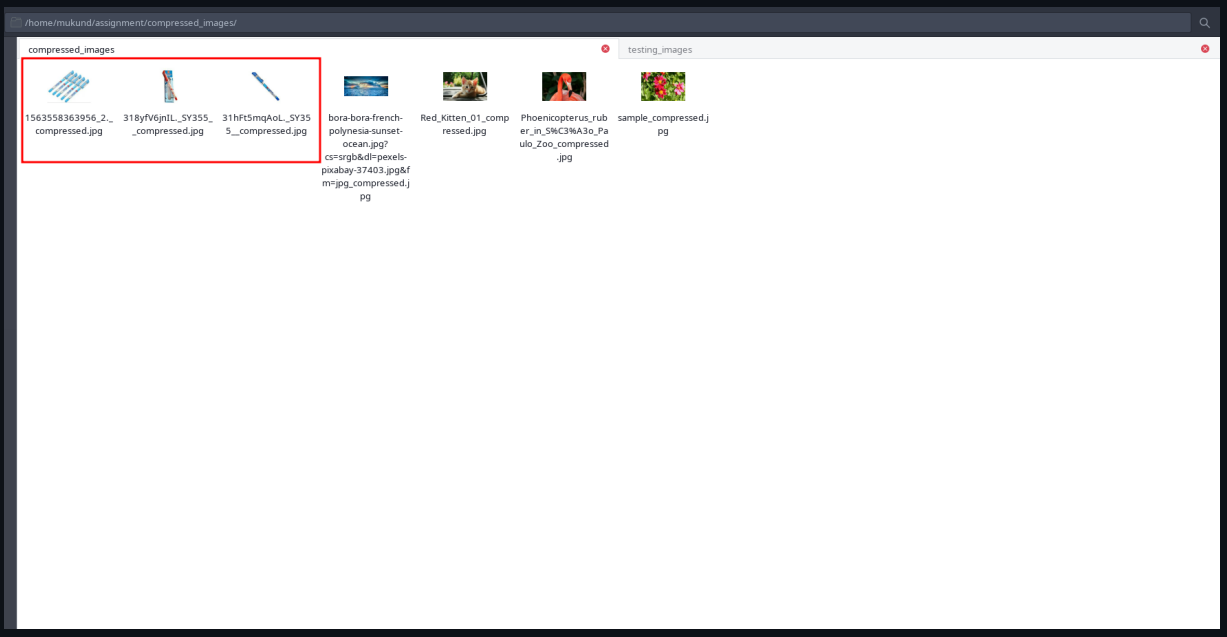
The terminal output confirms a successful request and addition of product ID to queue.

```
mukund@mukund-bravo15a4ddr ~/assignment  main  go run ./app.go
Database Connected Successfully!
Successfully connected to RabbitMQ!
All schemas, tables, et cetera have been migrated
[GIN-debug] [WARNING] Creating an Engine instance with the Logger and Recovery middleware already attached.

[GIN-debug] [WARNING] Running in "debug" mode. Switch to "release" mode in production.
 - using env:   export GIN_MODE=release
 - using code:  gin.SetMode(gin.ReleaseMode)

[GIN-debug] GET    /status                   --> main.main.func1 (3 handlers)
[GIN-debug] POST   /addProduct               --> main.main.func2 (3 handlers)
[GIN-debug] [WARNING] You trusted all proxies, this is NOT safe. We recommend you to set a value.
Please check https://pkg.go.dev/github.com/gin-gonic/gin#readme-don-t-trust-all-proxies for details.
[GIN-debug] Listening and serving HTTP on :5000
Product ID: 1
[x] Sent 1
[GIN] 2023/05/24 - 22:46:46 | 200 |     13.05841ms |             ::1 | POST     "/addProduct"
Product ID: 2
[x] Sent 2
[GIN] 2023/05/25 - 06:31:07 | 200 |     39.63995ms |             ::1 | POST     "/addProduct"
```

Parallely, our consumer who has always been listening should have recieved the same PID and for verification printed out the DB entry after compressing all images and placing it in it's respective path and updating the DB with the array of compressed image paths.

```
mukund@mukund-bravo15a4ddr ~/assignment  main  go run ./consumer.go
Successfully connected to Postgres DB
RabbitMQ connection Successful!
[*] Waiting for Messages from queue... To exit press Ctrl+C
2023/05/24 22:47:01 Recieved a PID: 1
{1 Stapler Best Stapler Kangaro [https://res.cloudinary.com/demo/image/upload/v1312461204/sample.jpg https://upload.wikimedia.org/wikipedia/commons/f/f9/Phoenicopterus_ruber_in_S%C3%A3o_Paulo_Zoo.jpg https://upload.wikimedia.org/wikipedia/commons/a/a5/Red_Kitten_01.jpg https://images.pexels.com/photos/37403/bora-bora-french-polynesia-sunset-ocean.jpg?cs=srgb&dl=pexels-pixabay-37403.jpg&fm=jpg] 300 [compressed_images/sample_compressed.jpg compressed_images/Phoenicopterus_ruber_in_S%C3%A3o_Paulo_Zoo_compressed.jpg compressed_images/Red_Kitten_01_compressed.jpg compressed_images/bora-bora-french-polynesia-sunset-ocean.jpg?cs=srgb&dl=pexels-pixabay-37403.jpg&fm=jpg_compressed.jpg] 0001-01-01 05:53:28 +0553 LMT 0001-01-01 05:53:28 +0553 LMT}
2023/05/25 06:31:07 Recieved a PID: 2
{2 Cello Butterfree Ball Point Pen [https://m.media-amazon.com/images/I/31hFt5mqAoL._SY355_.jpg https://m.media-amazon.com/images/I/318yfV6jnIL._SY355_.jpg https://assetscdn1.paytm.com/images/catalog/product/S/ST/STACELLO-BUTTERHEAD8922516E054A9/1563558363956_2..jpg] 70 [compressed_images/31hFt5mqAoL._SY355__compressed.jpg compressed_images/318yfV6jnIL._SY355__compressed.jpg compressed_images/1563558363956_2._compressed.jpg] 0001-01-01 05:53:28 +0553 LMT 0001-01-01 05:53:28 +0553 LMT}
```

In the compressed directory, we should see three images of cello butterfree pen resized and compressed.



There is a decrease in size as seen in unit testing for consumer.

Benchmark Testing:

I’m quite unfamiliar with performance testing and benchmark testing so I am generalizing performance observations.

I am making use of one RabbitMQ queue which can handle more than 20K+ messages per second. The API has little to no performance delay since it is running in local and has very few handlers. We can see a downgrade in performance if the API was hosted online. Due to networks delays, et cetera.