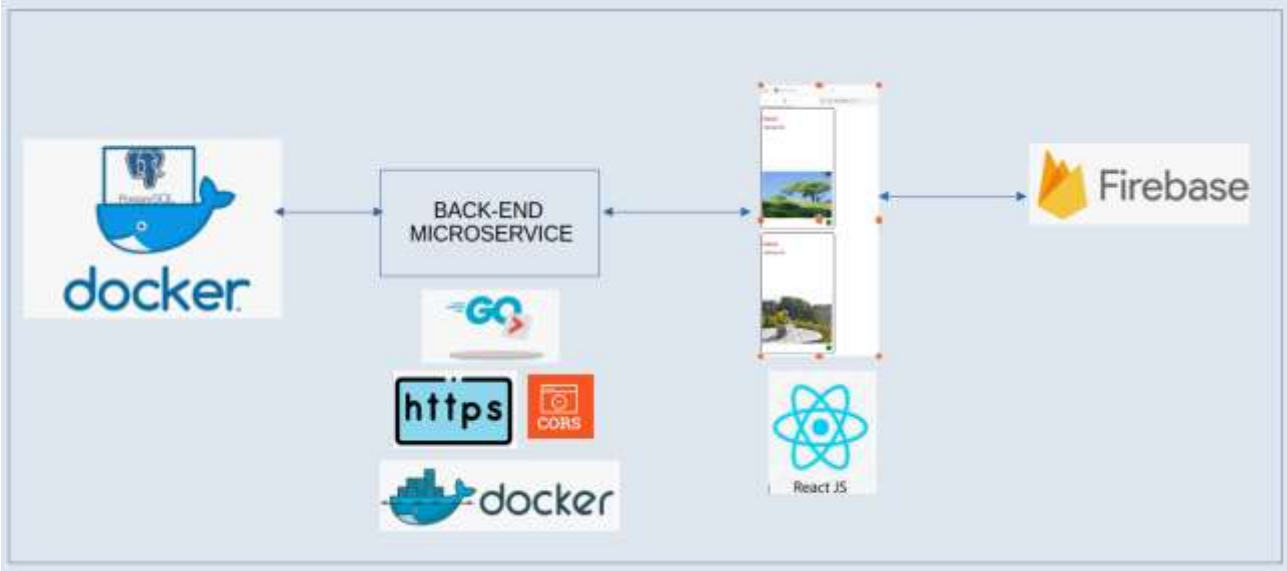
Golang Sofware Development Position Case Study

Murat Tunç

+90 531 731 58 54

murat.tunc8558@gmail.com

Genaral Project Overview



Basic Steps

FRONT-END

In This Case Study, ***React*** framework is used. Each component has its own css file.

App.js basic component calls all other necessary components. ***Props*** property is used for parameter transfer to other components, ***State*** property is used for process results in components.

There are 5 components in total, 4 ***Main*** components and 1 ***Header*** component.

After this process, the non-visual SnapshotScreen.js and FireBaseImageUpload.js components are called. SnapshotScreen.js takes the screen snapshot and keeps all the operations in its memory and lists them down the main screen.

FireBaseImageUpload sends the selected image to the Firebase environment and transmits the ***imageURL***, ***creation time*** information, ***title*** and ***description*** information previously entered by the

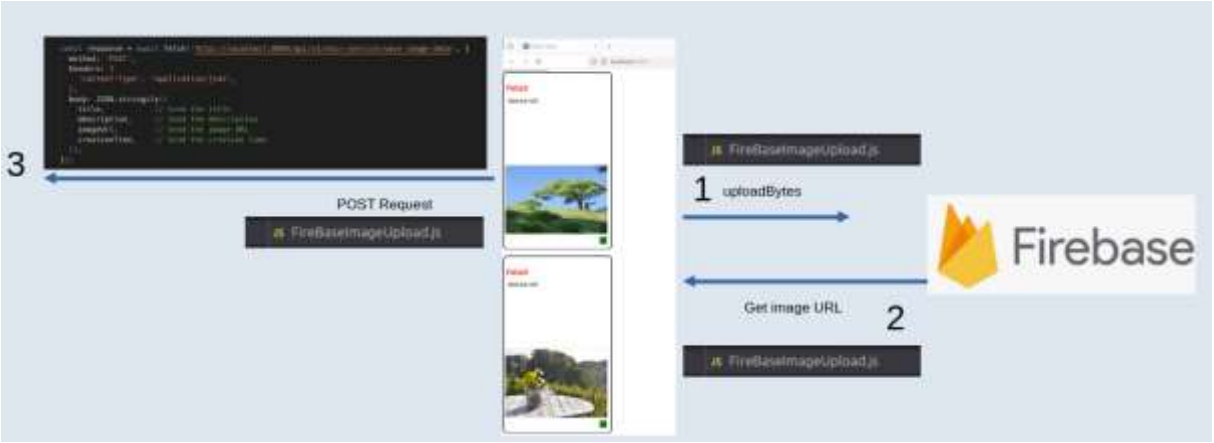
```
const response = await fetch('http://localhost:8080/api/v1/main-service/save-image-data', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
  },
  body: JSON.stringify({
    title,           // Send the title
    description,     // Send the description
    imageUrl,        // Send the image URL
    creationTime,    // Send the creation time
  }),
});
```

user from Firebase to the Backend service in ***JSON*** format via ***POST http*** request.

When all components have ***finished*** working, the reset process is performed. After the ***reset process***, the first selection status on the main screen is reached.

```
19  // Reset all components' states
20  const resetComponents = () => {
21    setTitle('New Title');
22    setDescription('New Description');
23    setImage(null);
24    setIsImageSelected(false);
25  };
26
27  // Handle snapshot after ActivatedIcon turns green
28  const handleSnapshot = () => {
29    if (frameRef.current) {
30      html2canvas(frameRef.current).then((canvas) => {
31        const imageData = canvas.toDataURL();
32        setSnapshots((prevSnapshots) => [...prevSnapshots, imageData]);
33        resetComponents();
34      });
35    }
36  };
```

The basic 3 steps on the front end side



BACK-END

The back end part consists of two parts: the basic ***postgresql-database*** and the ***go microservice.***

POSTGRESQL-DATABASE

In the back end folder, there is a script file with the following file extension that creates and starts the desired table. You need to run this script file from the terminal. main-service/scripts/start_postgres.sh

```
mutu@mutu:~/projects/backend/main-service/scripts$ ./start_postgres.sh
Container name 'depixen-postgres' is already in use.
Stopping and removing the existing container...
depixen-postgres
depixen-postgres
Pulling the latest PostgreSQL Docker image...
latest: Pulling from library/postgres
6e909acdb790: Pull complete
fec99121872b: Pull complete
133acbc970df: Pull complete
e02d97322fc6: Pull complete
db9643c6baf3: Pull complete
9bcedd9434e7: Pull complete
fc8982ec96d9: Pull complete
1824bd6b75d7: Pull complete
fbad2bf2d5e6: Pull complete
221788d72606: Pull complete
e5f43b682bc0: Pull complete
e7a2d9e24ab0: Pull complete
a96cb29b0d13: Pull complete
140970538145: Pull complete
Digest: sha256:aa7569015a6ce74ea2814ba0c044a18afebadaa02d691f5767b9a58d45ecfbf6
Status: Downloaded newer image for postgres:latest
docker.io/library/postgres:latest
Starting the PostgreSQL container...
38b4c9b7995e5b87f558bd40f7a0c3aa35671aea71a397fe5c24355410460eec
Waiting for PostgreSQL to start...
Creating the table 'tb_casestudy'...
NOTICE:  relation "tb_casestudy" already exists, skipping
CREATE TABLE
Table 'tb_casestudy' created successfully!
```

You can use the ***psql*** command-line tool to display rows from your ***tb_casestudy*** table.

Follow these steps:

1. Access PostgreSQL Inside Docker

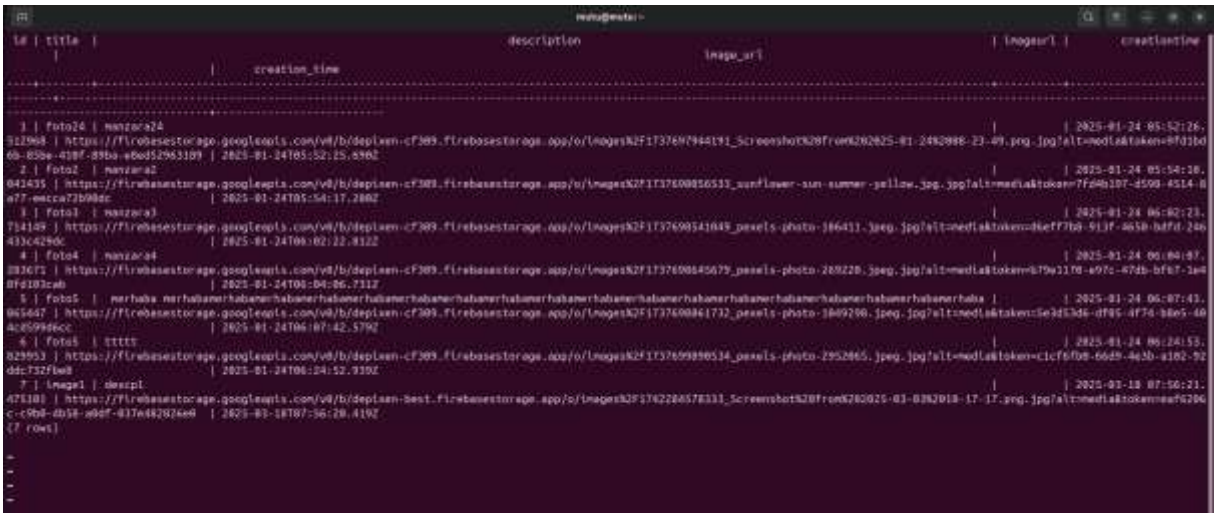
Run the following command to enter the PostgreSQL container:

docker exec -it depixen-postgres psql -U postgres -d postgres

2. Show Rows from tb_casestudy

Once inside `psql`, run:

SELECT * FROM tb_casestudy;



GO MICROSERVICE

🛠️ *Project Hierarchy & Code Flow*

```
main-service/
├── cmd/
│   └── main.go           # Entry point of your service
├── go.mod                # Go modules dependencies
├── go.sum                # Checksums for dependencies
├── internal/
│   ├── endpoint/
│   │   └── endpoints.go  # Defines request/response transformation & business logic
│   ├── models/
│   │   └── models.go     # Defines data structures (DB models, DTOs, etc.)
│   ├── service/
│   │   └── service.go    # Implements business logic (core functionalities)
│   └── transport/
│       └── http.go       # Handles HTTP transport (maps endpoints to HTTP routes)
└── Readme.md             # Documentation
```

Step	Component	Function
1️⃣ **Main**	`main.go`	Entry point, initializes DB, service, routes
2️⃣ **Service Layer**	`service.go`	Business logic (saves image data)
3️⃣ **Transport Layer**	`http.go`	Maps API requests to service functions
4️⃣ **Endpoint Layer**	`endpoints.go`	Transforms request/response data
5️⃣ **Database**	`models.go`	Defines `ImageData` model, stores records

main.go-->

Go Kit (kitHttp) → Used for handling HTTP transport.

- **Gorilla Mux** → Routing library for defining API endpoints.
- **CORS** → Allows frontend (React) to communicate with backend.
- **GORM** → ORM library for connecting to **PostgreSQL**.
- **Fatih Color** → Enhances log readability with colors.
- **Internal Modules** → Includes models, service logic, and transport logic.

🔗 Full Flow (Request to Response)

1️⃣ **User sends a** `POST` **request** to:

/api/v1/main-service/save-image-data

2️⃣ **HTTP Transport (**`http.go`**)**

- Calls `DecodeSaveImageDataRequest` to parse JSON request.
- Passes request to `saveImageDataEndpoint`.

3️⃣ **Endpoint (**`endpoints.go`**)**

- Calls the `Service` method (`SaveImageData`).

4️⃣ **Service (**`service.go`**)**

- **Saves the image metadata** to the PostgreSQL database.
- Returns success or error.

5️⃣ **Response is sent back** to the client.

service.go-->

This `service.go` file defines the service layer of your application, which interacts with the database to handle image metadata storage.

☑ Defines the `Service` interface for saving image metadata.
☑ Implements `ImageService`, which stores data in PostgreSQL using GORM.
☑ Logs both **success** and **error** messages using **colored output**.
☑ Uses **GORM** and **context-aware database operations** for better request handling.

http.go-->

This `http.go` file is responsible for handling HTTP transport logic in your service. It provides functions to decode requests, call the service layer, and encode responses.

☑ **Defines request/response structures** for JSON communication.
☑ **Creates an endpoint** (`MakeSaveImageDataEndpoint`) that calls the service layer.

☑ **Decodes HTTP requests** into Go structs (`DecodeSaveImageDataRequest`).
☑ **Encodes responses** into JSON (`EncodeResponse`).

This file acts as a **bridge** between the HTTP layer and the service layer.

endpoints.go-->

This `endpoints.go` file is part of the **service layer**, and it acts as an intermediary between the **transport layer (HTTP handlers)** and the **business logic (service layer)**.

☑ Defines **request/response** structures.
☑ **Handles request validation** (checks for invalid requests).
☑ **Maps request data to** `models.ImageData`.
☑ Calls the **service layer** (`SaveImageData`).
☑ Returns **success or error responses**.

models.go-->

This `models.go` file defines the data model for storing image metadata in the database using GORM (Go Object Relational Mapper).

Defining the `ImageData` Struct

```
type ImageData struct {
    ID           uint   `gorm:"primaryKey"`   // Auto-incrementing ID field
    Title        string `json:"title"`        // Title of the image or case study
    Description  string `json:"description"`  // Description of the image
    ImageURL     string `json:"imageUrl"`     // URL of the image
    CreationTime string `json:"creationTime"` // Time when the image was created in
}
```

- Represents **the structure of the** `tb_casestudy` **table** in the database.

- **GORM tags (`gorm:"..."`)** define database behavior.
- **JSON tags (`json:"..."`)** specify how the struct fields should be **serialized/deserialized in API requests/responses**.

Explanation of Fields

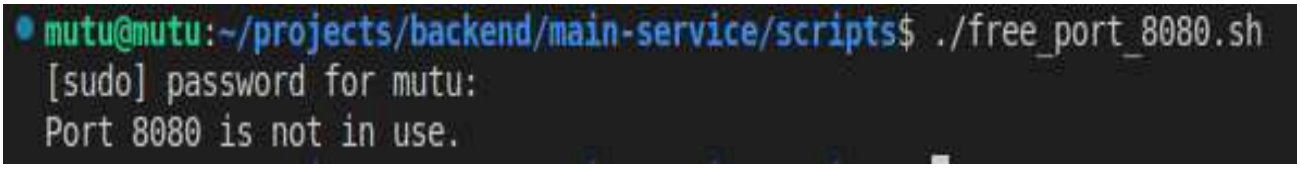
Field	Type	Database Behavior	JSON Field	Purpose
`ID`	`uint`	Primary Key (Auto-increment)	N/A	Unique identifier for each image record
`Title`	`string`	Stored as a text field	`"title"`	Title of the image/case study
`Description`	`string`	Stored as a text field	`"description"`	Short description of the image
`ImageURL`	`string`	Stored as a text field	`"imageUrl"`	URL link to the uploaded image
`CreationTime`	`string`	Stored as a text field	`"creationTime"`	Timestamp of image creation in Firebase Cloud

☑ Defines a **GORM model** for storing image metadata.
☑ **Overrides the table name** to `tb_casestudy`.
☑ **Ensures correct JSON serialization/deserialization**.
☑ Supports **database operations** like create, read, update, and delete (CRUD).

This model is essential for **saving and retrieving case study images** efficiently.

Steps to run the project:

1-Let's run the main-service/scripts/free_port_8080.sh file



2-Let's run the main-service/scripts/start_postgres.sh file

```
mutu@mutu:~/projects/backend/main-service/scripts$ ./start_postgres.sh
Container name 'depixen-postgres' is already in use.
Stopping and removing the existing container...
depixen-postgres
depixen-postgres
Pulling the latest PostgreSQL Docker image...
latest: Pulling from library/postgres
6e909acdb790: Pull complete
fec99121872b: Pull complete
133acbc970df: Pull complete
e02d97322fc6: Pull complete
db9643c6baf3: Pull complete
9bcedd9434e7: Pull complete
fc8982ec96d9: Pull complete
1824bd6b75d7: Pull complete
fbad2bf2d5e6: Pull complete
221788d72606: Pull complete
e5f43b682bc0: Pull complete
e7a2d9e24ab0: Pull complete
a96cb29b0d13: Pull complete
140970538145: Pull complete
Digest: sha256:aa7569015a6ce74ea2814ba0c044a18afebadaa02d691f5767b9a58d45ecfbf6
Status: Downloaded newer image for postgres:latest
docker.io/library/postgres:latest
Starting the PostgreSQL container...
38b4c9b7995e5b87f558bd40f7a0c3aa35671aea71a397fe5c24355410460eec
Waiting for PostgreSQL to start...
Creating the table 'tb_casestudy'...
NOTICE:  relation "tb_casestudy" already exists, skipping
CREATE TABLE
Table 'tb_casestudy' created successfully!
```

3-Let's run the main-service/cmd/main.go file

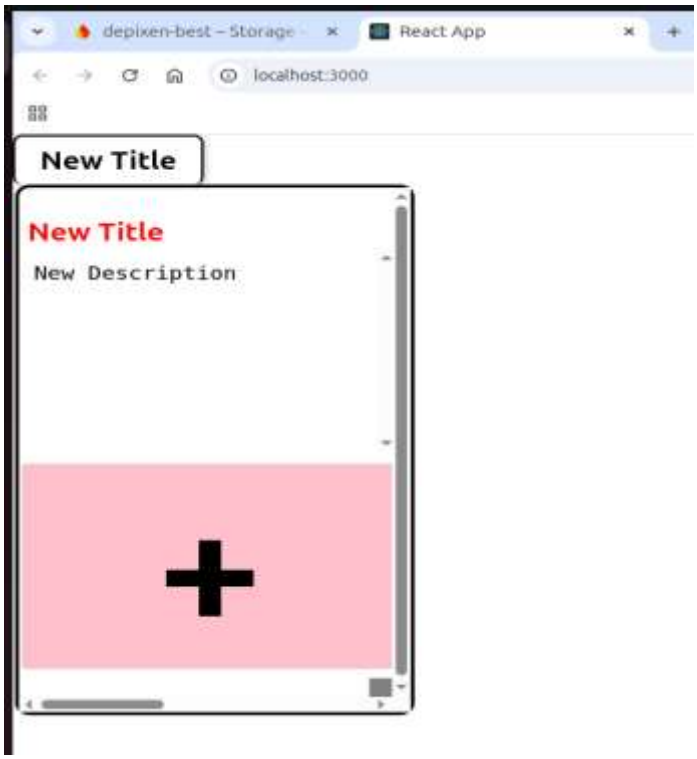
```
mutu@mutu:~/projects/backend/main-service/cmd$ go run main.go
2025/03/18 11:49:14 INFO: Successfully connected to the database!
2025/03/18 11:49:14 INFO: Starting server on port :8080...
```

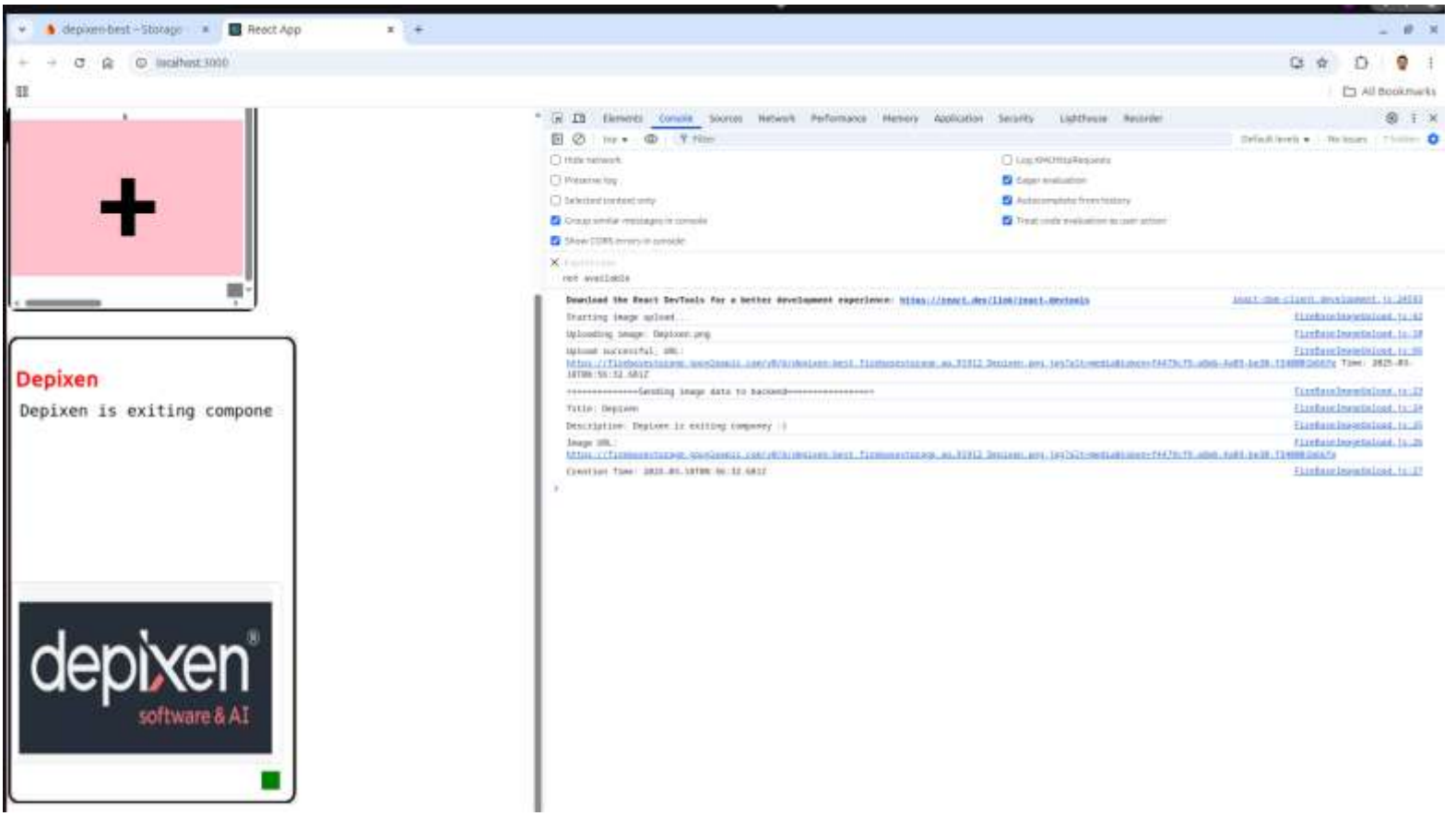
Backend and Database are up, waiting for information from the frontend

4-Running the Front End

```
mutu@mutu:~/projects/frontend$ npm start
```

```
Compiled successfully!

You can now view frontend in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.0.8:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```



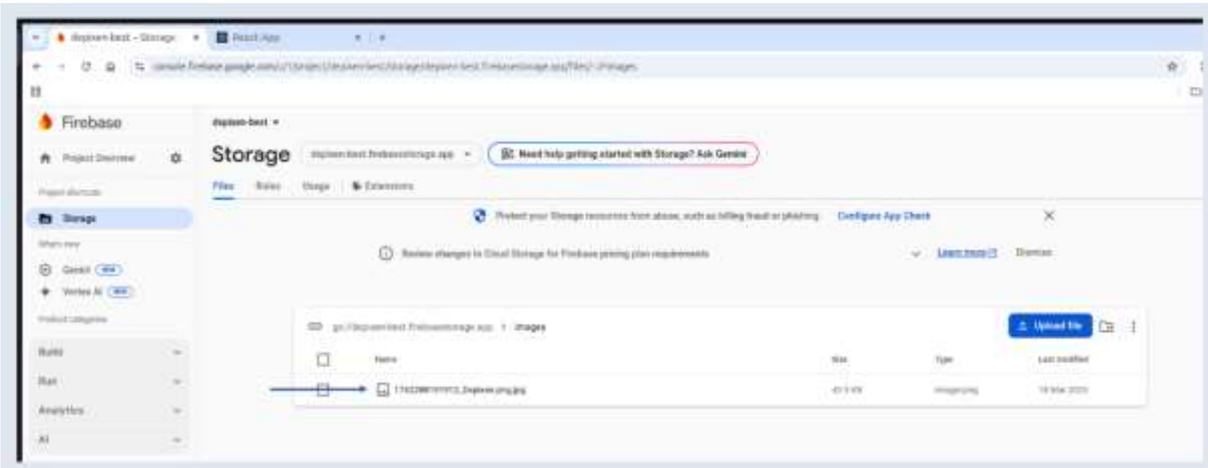



```
mutu@mutu:~/projects/backend/main-service/cmd$ go run main.go
2025/03/18 11:49:14 INFO: Successfully connected to the database!
2025/03/18 11:49:14 INFO: Starting server on port :8080...
2025/03/18 11:56:33 Decoded request: {Title:Depixen Description:Depixen is exiting componey :) ImageURL:https://firebasestorage.googleapis.com/v0/b/depixen-best.firebasestorage.app/o/images%2F1742288191912_Depixen.png.jpg?alt=media&token=f4479cf9-a8eb-4a89-be30-f340001b66fe CreationTime:2025-03-18T08:56:32.681Z}
2025/03/18 11:56:33 Received data: {ID:0 Title:Depixen Description:Depixen is exiting componey :) ImageURL:https://firebasestorage.googleapis.com/v0/b/depixen-best.firebasestorage.app/o/images%2F1742288191912_Depixen.png.jpg?alt=media&token=f4479cf9-a8eb-4a89-be30-f340001b66fe CreationTime:2025-03-18T08:56:32.681Z}
2025/03/18 11:56:33 Successfully saved data to the database: {ID:8 Title:Depixen Description:Depixen is exiting componey :) ImageURL:https://firebasestorage.googleapis.com/v0/b/depixen-best.firebasestorage.app/o/images%2F1742288191912_Depixen.png.jpg?alt=media&token=f4479cf9-a8eb-4a89-be30-f340001b66fe CreationTime:2025-03-18T08:56:32.681Z}
```

```
mutu@mutu: $ docker exec -it depixen-postgres psql -U postgres -d postgres
psql (17.4 (Debian 17.4-1.pgdg120+2))
Type "help" for help.

postgres=# SELECT * FROM tb_casestudy;
postgres=#
postgres=#
postgres=#
postgres=#
postgres=# SELECT * FROM tb_casestudy;
```

As you can see, the last image file information I showed was written to the db.

FireBase Cloud Result:



Thank you so much for taking the time to read...

Murat Tunç
+90 531 731 58 54
murat.tunc8558@gmail.com