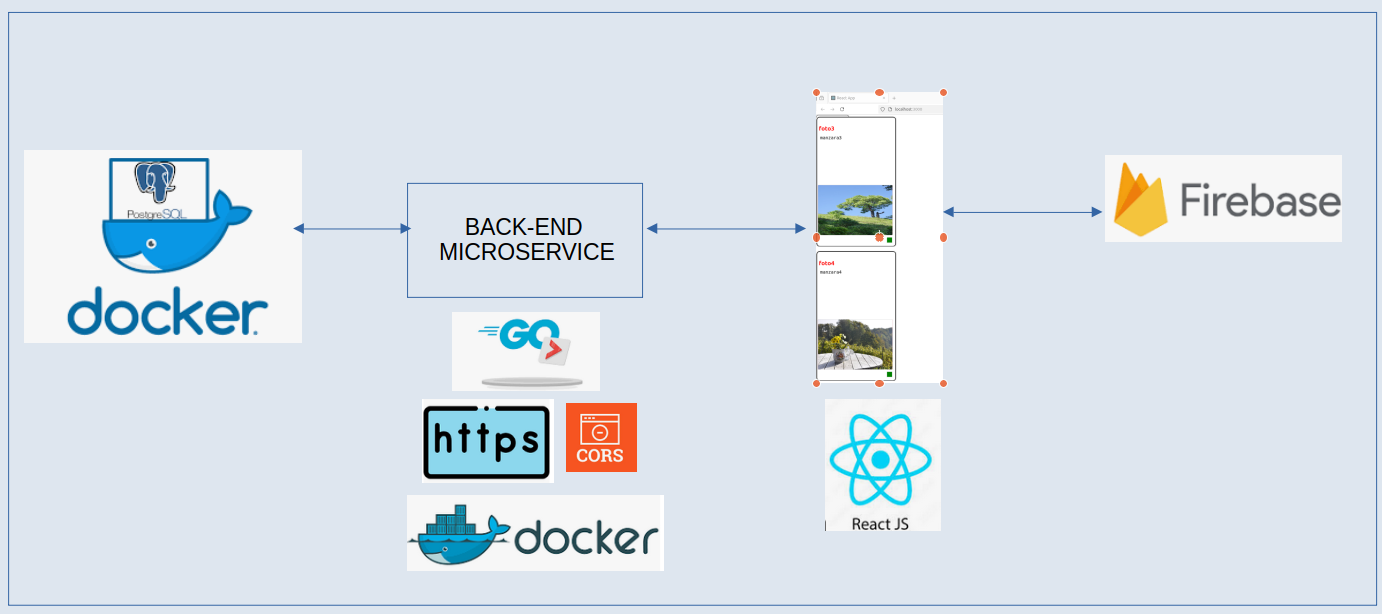
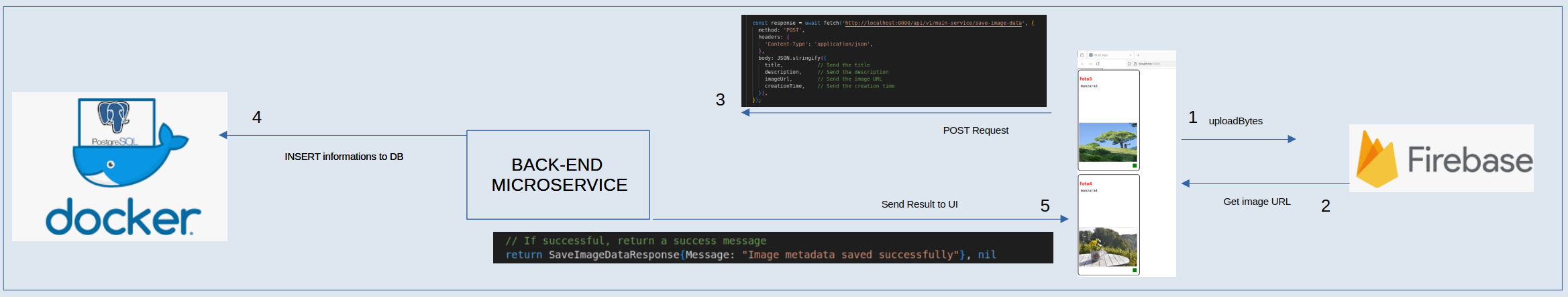
****

***Genaral Project Overview***



***Basic Steps***

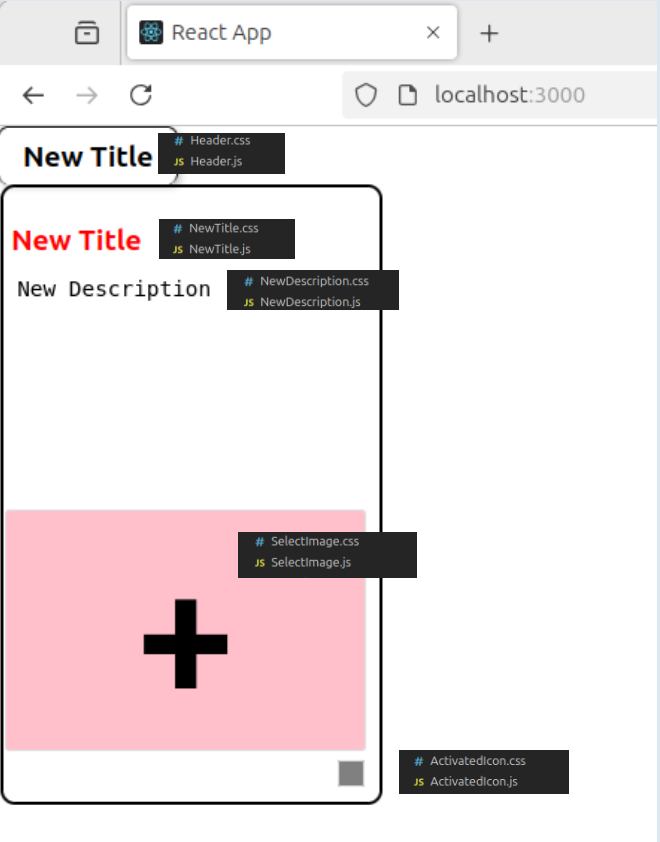


***FRONT-END***

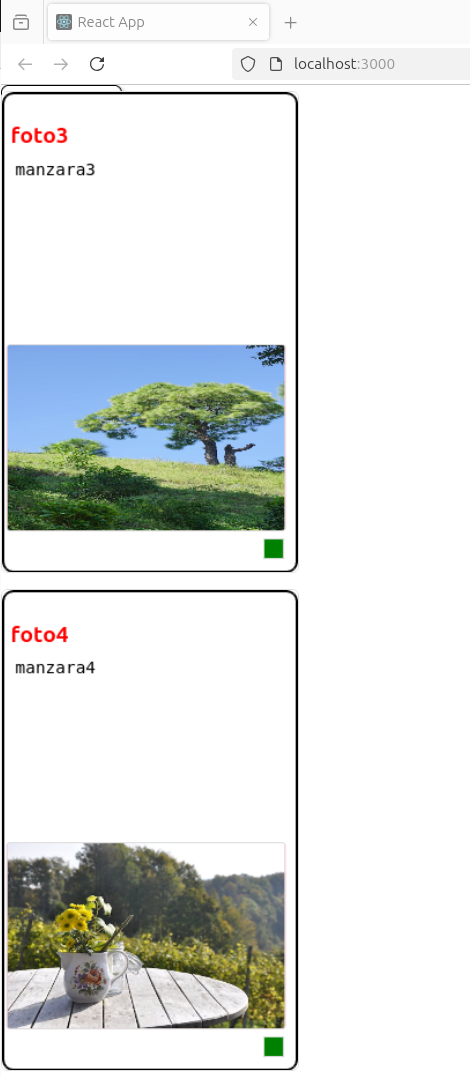
In This Case Study, ***React*** framework is used. Each component has its own css file.

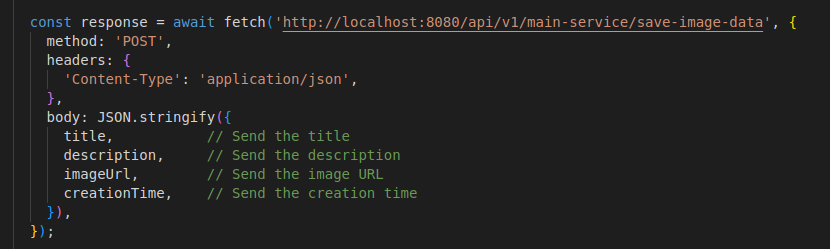
***App.js*** basic component calls all other necessary components. ***Props*** property is used for parameter transfer to other components, **S*tate*** property is used for process results in components.

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

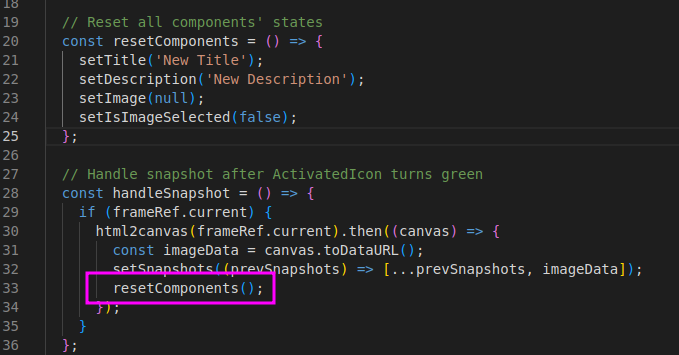


After this process, the non-visual SnapshotScreen.js and FireBaseImageUpload.js components are called. SnapshotScreen.js takes the screen snapshot and keeps all the operations in its memory and lists them down the main screen.

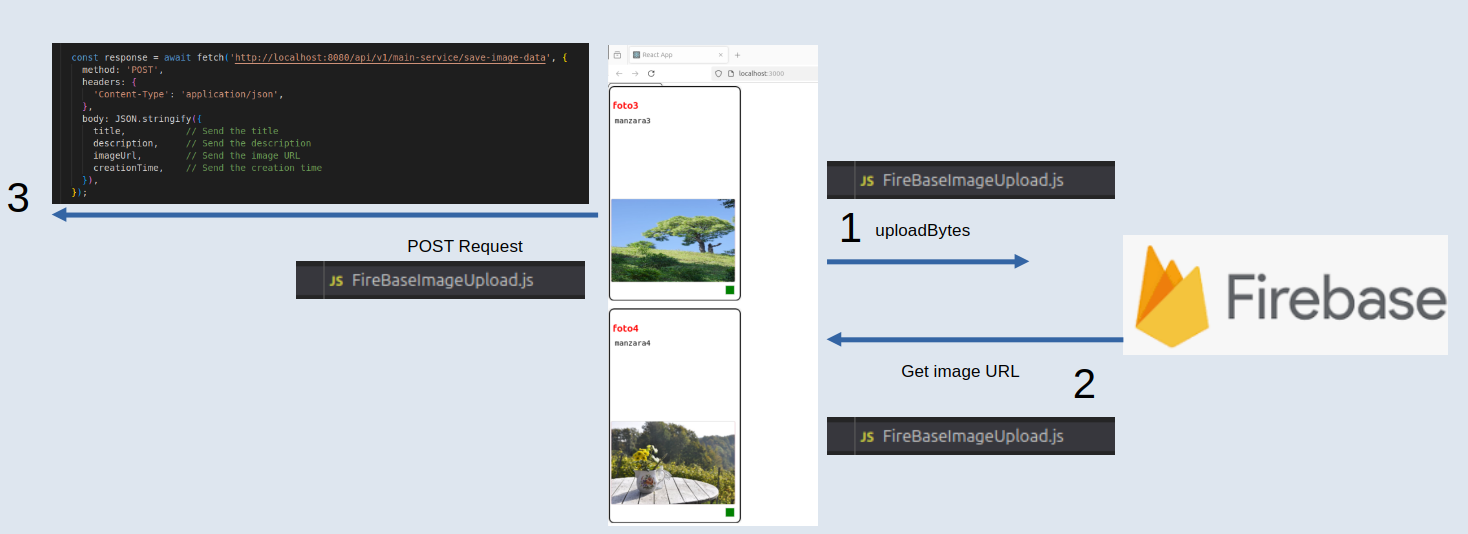


***FireBaseImageUpload*** sends the selected image to the Firebase environment and transmits the ***imageURL***, ***creation time*** information, ***title*** and ***description*** information previously entered by the user from Firebase to the Backend service in ***JSON*** format via ***POST*** ***http*** 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.



***The basic 3 steps on the front end side***

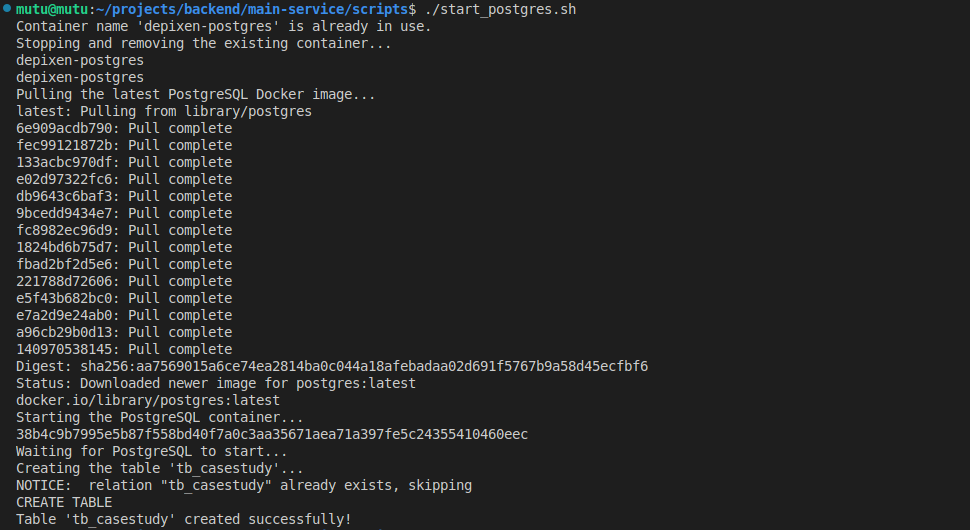


***BACK-END***

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



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

Follow these steps:

**1. Access PostgreSQL Inside Docker**

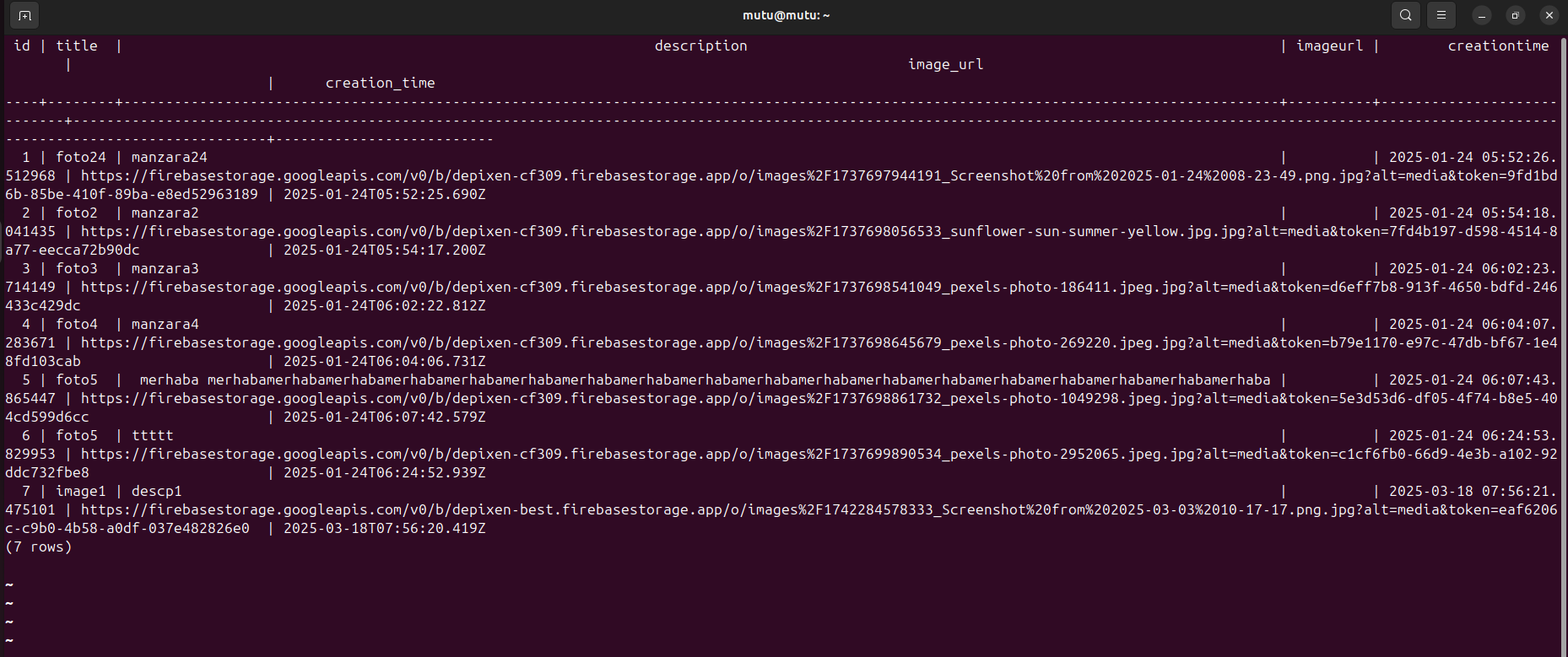
Run the following command to enter the PostgreSQL container:

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

### ****2. Show Rows from**** tb\_casestudy

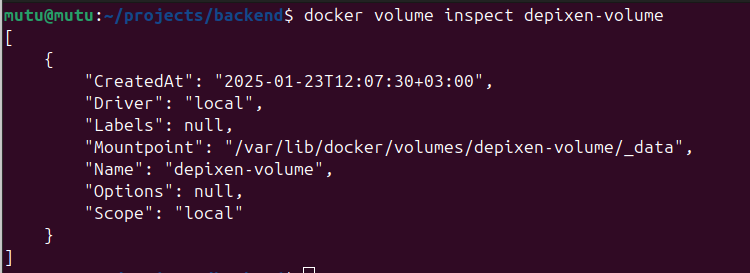
Once inside psql, run:

***SELECT \* FROM tb\_casestudy;***



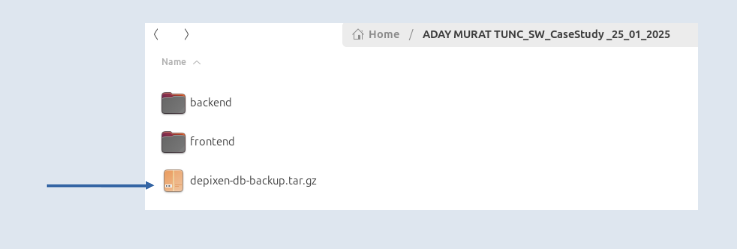
### **Database Volume on my Ubuntu System**

To check where Docker is storing this volume, run:



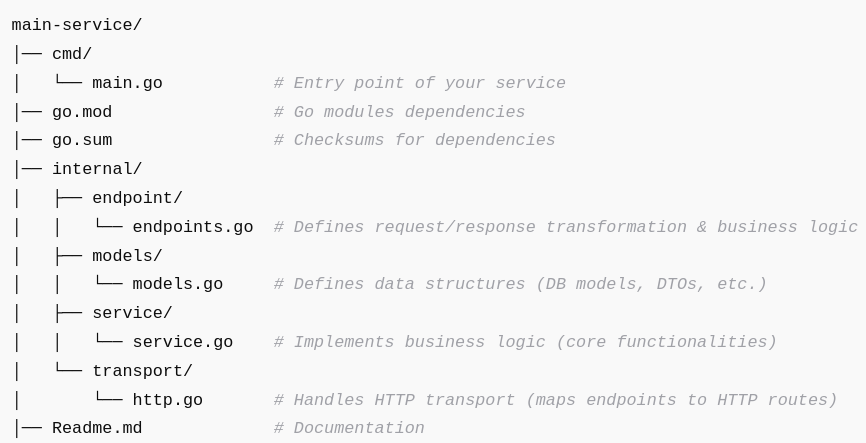
The **Mountpoint** field shows the actual path where your PostgreSQL data is stored (***/var/lib/docker/volumes/depixen-volume/\_data***).

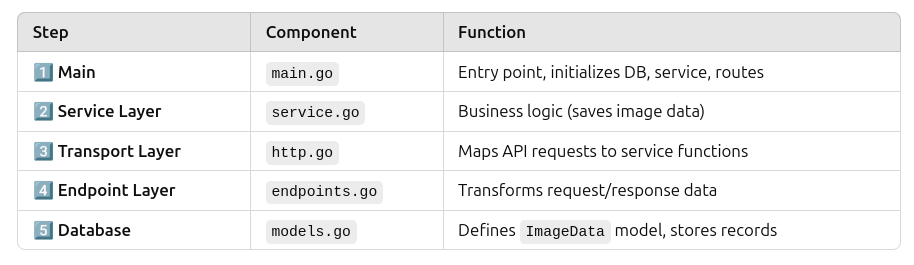
1️⃣ Archive the Database Volume to a .tar.gz File

tar -czvf depixen-db-backup.tar.gz -C /var/lib/docker/volumes/depixen-volume/\_data .

***GO MICROSERVICE***

***Project Hierarchy & Code Flow***

******

******

***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**.

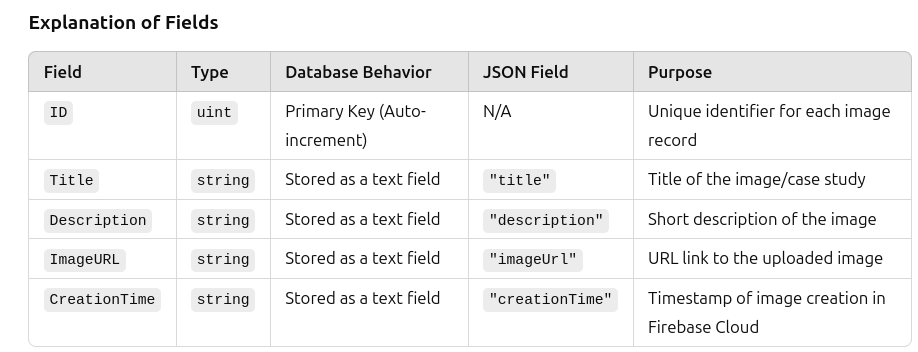
***models.go-->***

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



* 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**.



✅ 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).

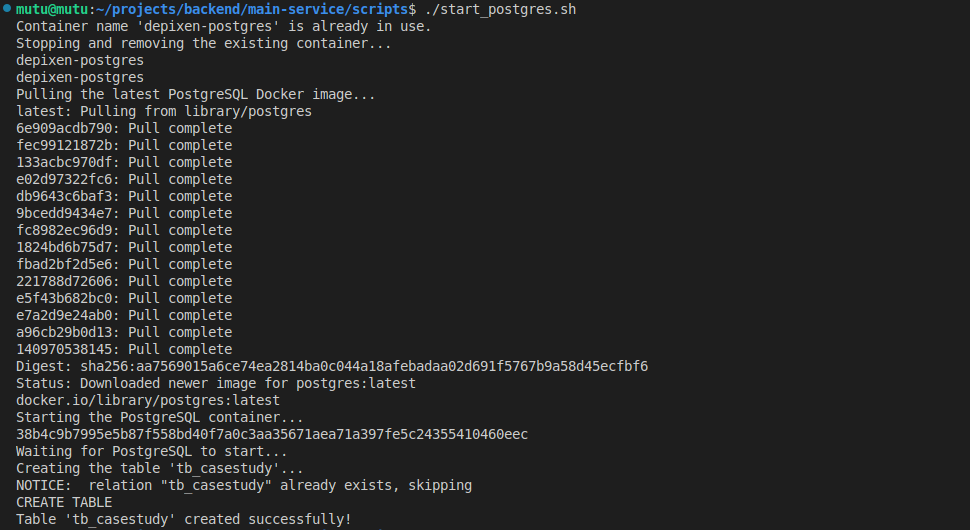
This model is essential for **saving and retrieving case study images** efficiently.

***Steps to run the project:***

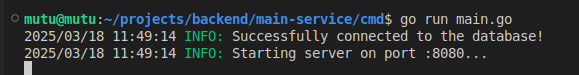
***1-Let's run the main-service/scripts/free\_port\_8080.sh file***

******

***2-Let's run the main-service/scripts/start\_postgres.sh file***

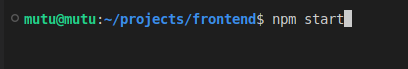
******

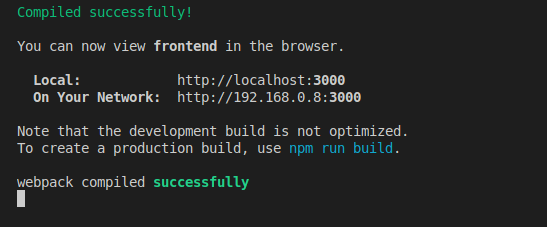
***3-Let's run the main-service/cmd/main.go file***

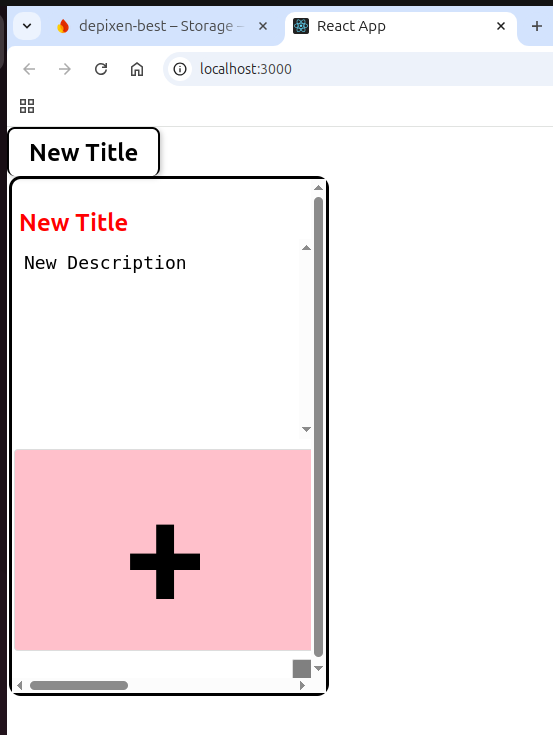
******

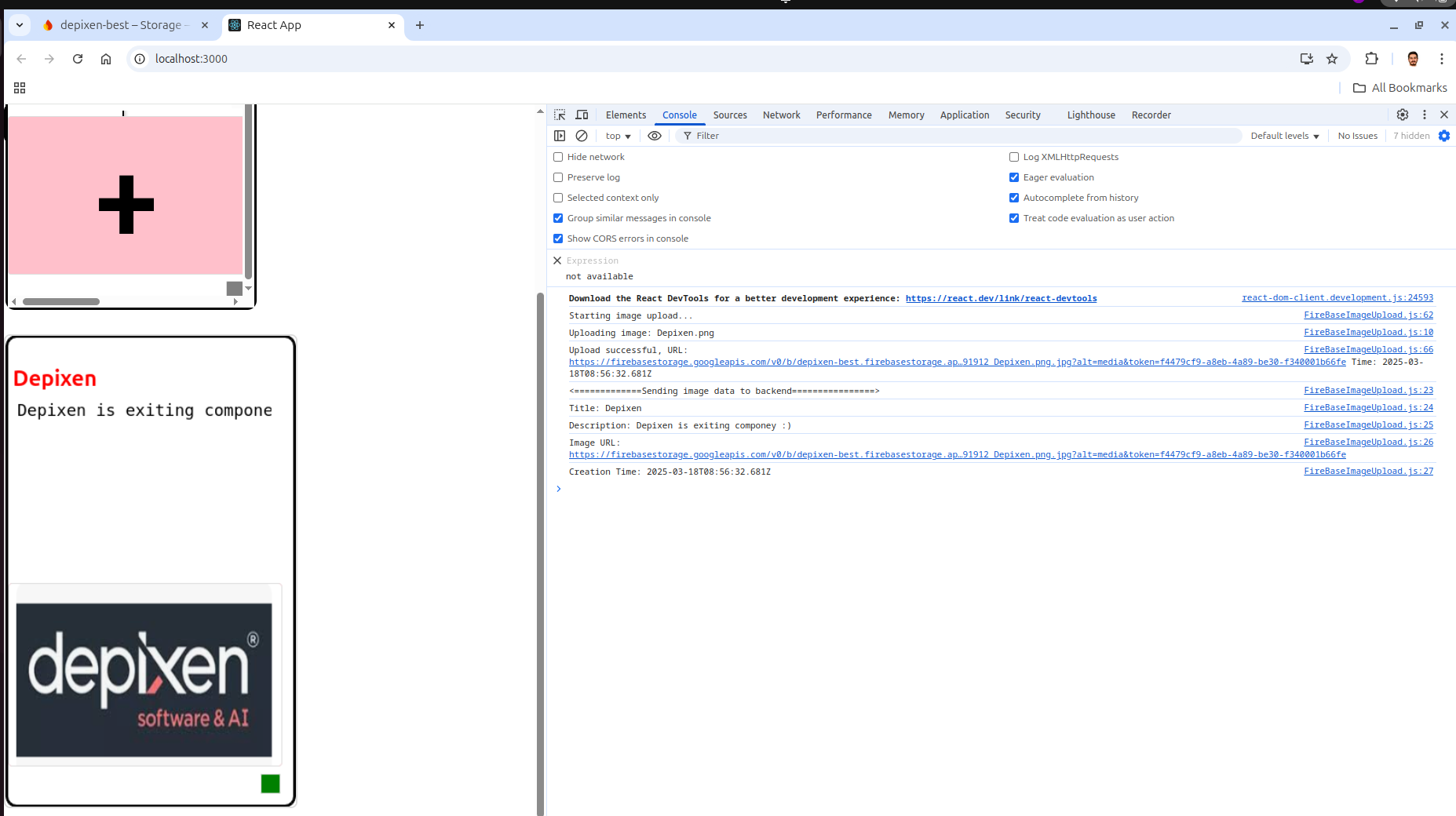
***Backend and Database are up, waiting for information from the frontend***

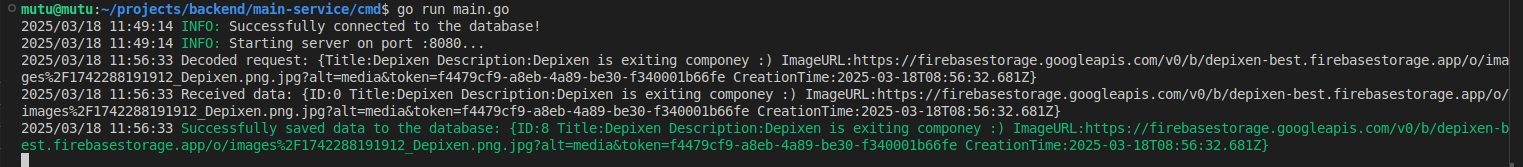
***4-Running the Front End***

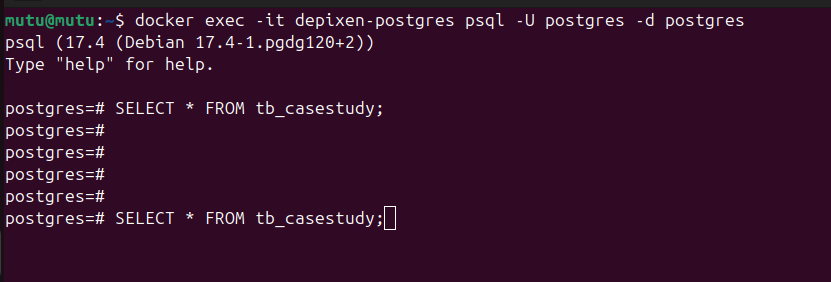


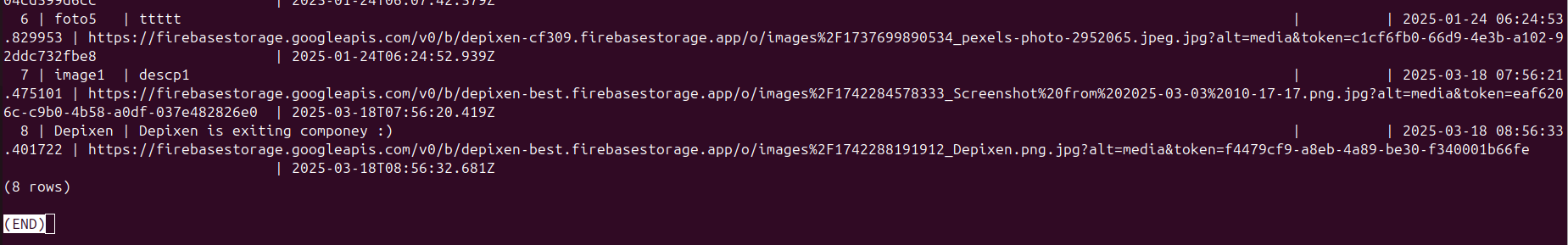






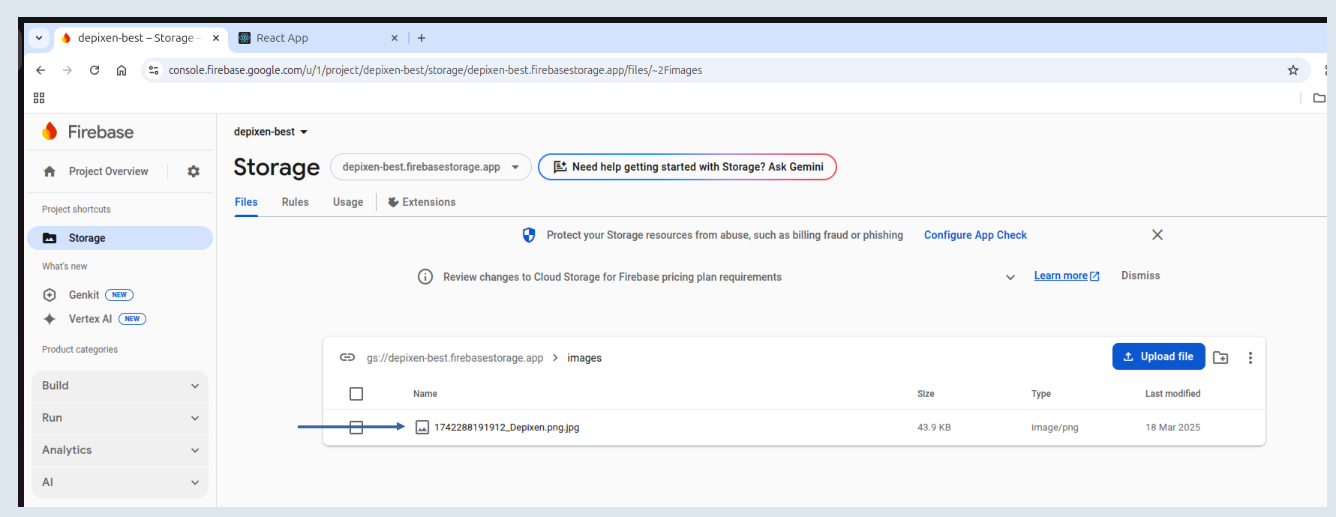






***As you can see, the last image file information I showed was written to the db.***

***FireBase Cloud Result:***

******

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

***Murat Tunç***

***+90 531 731 58 54***

***murat.tunc8558@gmail.com***