BUILD: Project 2 / Design & Implement a Document Database

Problem description

Problem Statement

Many photographers have two types of accounts, namely social media accounts, and photos stock website accounts, which require them to upload the same photo twice on different platforms. If photographers are able to upload their photos to two platforms simultaneously, then they no longer need to open two websites and click repeatedly.

Objective

A database that manage the photos that photographers uploaded to it, while storing the pictures with the related EXIF data for future updating purposes. The process will read photos uploaded to social media account, store them to the database, and upload to photo stock websites with a click.

Tables and Their Roles:

1. Photos: Stores information about each photo including its dimensions, location where it was taken, and other descriptive data.
2. Photo tags: Contains tags associated with each photo for easier categorization and searchability.
3. Photographers: Holds details about photographers such as their name and contact information.
4. Photographer account: Links photographers to their accounts on social media platforms and photo stock websites.
5. Time for posting: Manages scheduling for posts, allowing photos to be uploaded at specific times.
6. Authentication: Stores authentication details necessary for accessing the artist's social media and photo stock website accounts securely through APIs.
7. Camera information: Stores information about the camera used for taking a photograph, including model name and lens type.

Identified Nouns:

- Photographers
- Photos
- Process
- Database
- Photo stock websites
- Social media account

Identified Actions

- update

- simplifies
- upload
- storing
- enable

Feature Summary

1. Uploading Photos: Photographers can upload photos directly through this platform instead of visiting each platform individually.
2. Batch Uploading: Photographers can upload a batch (series) of photo through this platform to different platform.
3. Scheduling Uploads: Allows setting specific times when photos should be posted automatically.

UML Diagram

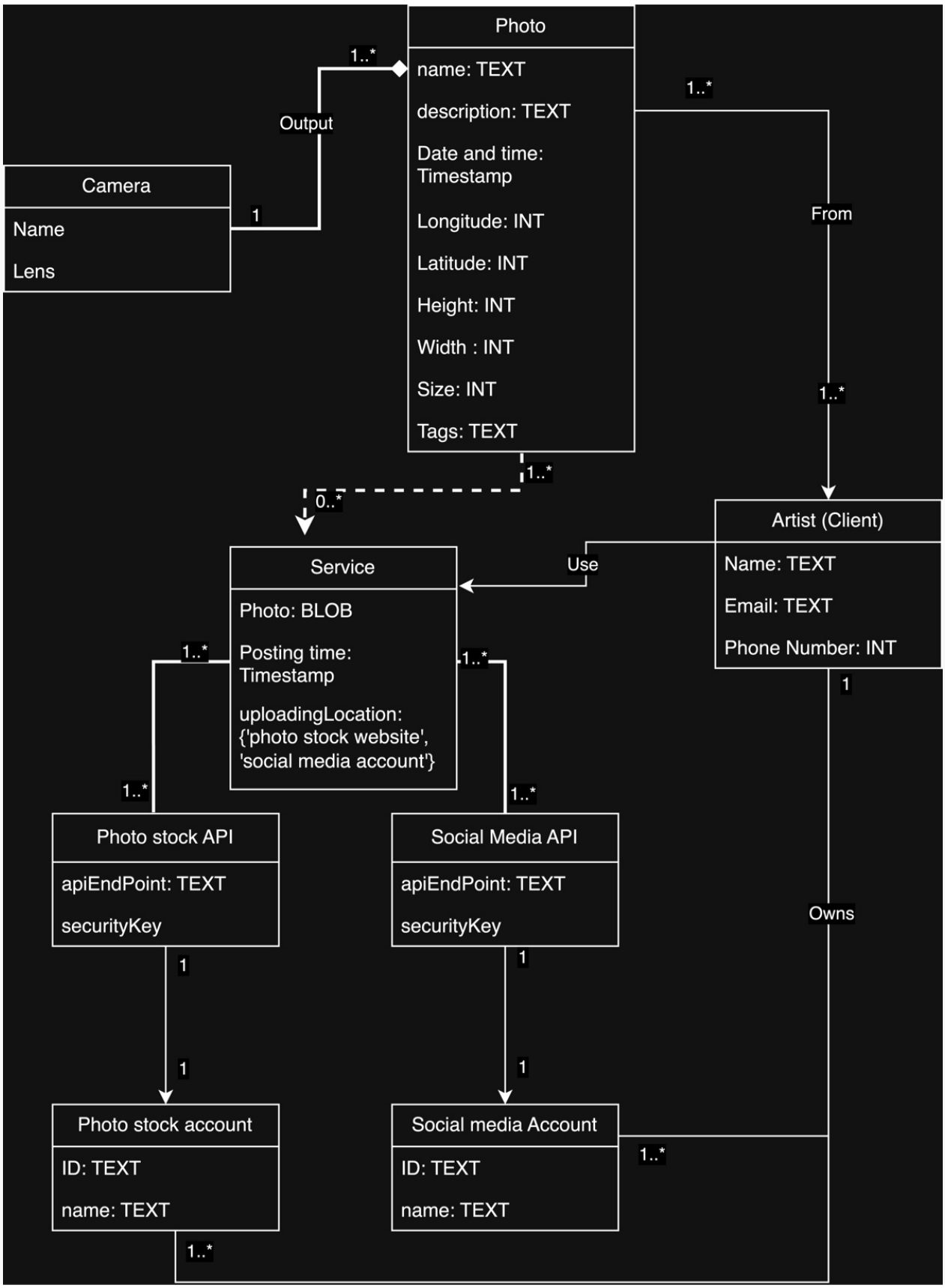
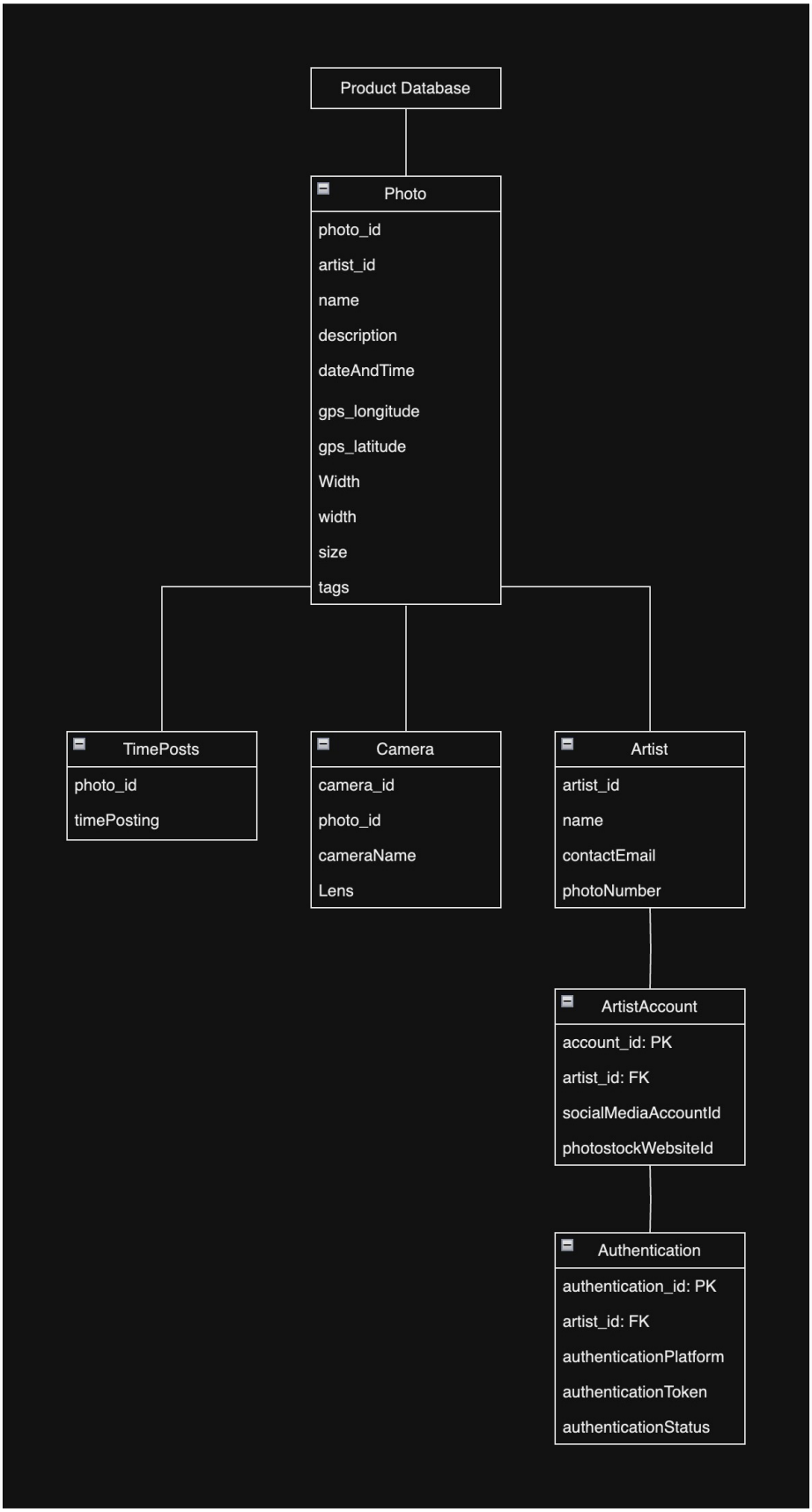


Photo
name: TEXT
description: TEXT
Date and time: Timestamp
Longitude: INT
Latitude: INT
Height: INT
Width : INT
Size: INT
Tags: TEXT
1..*
Output
Camera
Name
Lens
1
From
1..*
1..*
0..*
Artist (Client)
Name: TEXT
Email: TEXT
Phone Number: INT
Use
Service
Photo: BLOB
Posting time: Timestamp
uploadingLocation: {'photo stock website', 'social media account'}
1..*
1..*
1..*
1..*
1
Photo stock API
apiEndPoint: TEXT
securityKey
Social Media API
apiEndPoint: TEXT
securityKey
Owns
1
1
1
1
Photo stock account
ID: TEXT
name: TEXT
Social media Account
ID: TEXT
name: TEXT
1..*
1..*

Hierarchical tables



Defined main collection

```
{
  _id: ObjectId(),
  photo_id: Number,
  artist_id: Number,
  name: String,
  description: String,
  dateAndTime: Date,
  gps: { // The GPS location of the picture
    longitude: Number,
    latitude: Number
  },
  dimensions: { // The dimension of the picture
    width: Number,
    height: Number,
    size: Number
  },
  tags: [ // The tags that belongs to the pictures
    {
      tag_id: Number,
      name: String,
      category: String
    }
  ],
  timePosts: [
    {
      timePosting: Date //The date that the picture is estimated to post
    }
  ],
  camera: { // The camera information, originally Camera relation
    camera_id: Number,
    cameraName: String,
    lens: String
  },
  artist: { // Artist info, originally Artist relation
    artist_id: Number,
    name: String,
    contactEmail: String,
    photoNumber: String,
    artistAccount: { // Artist's accounts
      account_id: Number,
      socialMediaAccountId: String,
      photostockWebsiteId: String
      authentication: {
        authentication_id: Number,
        authenticationPlatform: String,
        authenticationToken: String,
        authenticationStatus: String
      }
    },
  }
}
```

Example object

```
/**
* Paste one or more documents here
*/
{
  "photo_id": 95094,
  "name": "Tranquil river flow",
  "description": "Abstract art with vibrant colors",
  "dateAndTime": {
    "$date": "2023-04-07T00:00:00.000Z"
  },
  "gps": {
    "longitude": -5.2697497,
    "latitude": 34.6756374
  },
  "dimensions": {
    "width": 1596,
    "height": 1246.98
  },
  "size": 45.64,
  "tags": {
    "tag_id": 0,
    "tagName": "Scenery"
  },
  "timePosts": {
    "timePosting": {
      "$date": "2024-06-07T00:55:44.000Z"
    }
  },
  "camera": {
    "cameraId": "CAM3429",
    "cameraName": "Blackmagic Pocket Cinema Camera 6K",
    "lens": "Sigma 35mm f/1.4"
  },
  "artist": {
    "artist_id": 73628,
    "artistName": "Alonso Candlish",
    "contactEmail": "acandlish0@usatoday.com",
    "photoNum": "919-193-1812",
    "artistAccount": {
      "account_id": 9705,
      "socialMediaAccountId": "acandlish0",
      "photostockWebsiteId": "acandlish0",
      "authentication": {
        "authentication_id": 7893,
        "authenticationPlatform": "Instagram",
        "authenticationToken": "2u5QW935Hz",
        "authenticationStatus": true
      }
    }
  }
}
```

Generating data

- I used Mockaroo to generate the data. The dump file was exported by “save as JSON schema”, and importing the schema into Mockaroo would basically generate the file in the same format.

Import

- I used Mongo Compass to import the file. I created a collection “photos” inside the database "photoStock”, then imported the JSON file generated from Mockaroo into the database.