WildWatch RESTful API Specification

Student: *Aly Shmahell* **Matricula:** *258912* **E-mail:** *aly.shmahell@student.univaq.it*

Preamble:

This project is a part of a large multi-course project:

- WildWatch: a mobile application implemented within the **Mobile Application course**.
- WildWatch Web: a web interface implemented within the **Web Engineering course**, consists of two web interfaces:
  - A public web app which can be accessed by anonymous guests.
  - An admin web app which can be accessed by the database administrators.
- WildWatch RESTful API: an API implemented within the **Advanced Web Development course**, which would handle the databases; therefore will manage the data generated by the mobile users, and provide the data to the mobile users, the guests and the admins.

The Service:

The urls will be split into three categories:

- /user: as an interface to the mobile app.
- /admin: as an interface to the admin web app.
- /guest: as an interface to the public web app.

URL: auth/user/register

VERB: POST

Input:

{“firstname”: str, “lastname”: str, “username”: str, “password”: str}

Semantics: creates a new user account based on the basic name info and provided credentials.

URL: auth/user/{SID}/unregister

VERB: DELETE

Semantics: deletes the user account, anonymizes their personal info, then closes the session, and invalidates the Session ID.

URL: auth/user|admin/login

VERB: POST

Input:

{“username”: str, “password”: str}

Output:

{“SID”: str}

Semantics: establishes a session based on the provided credentials and grants an {SID}.

URL: auth/user|admin/{SID}/logout

VERB: DELETE

Semantics: closes the session and invalidates the {SID}.

URL: auth/user/{SID}/profile

VERB: GET

Output:

{“fullname”: str, “website”: str, “bio”: str, “photo”: image/jpeg}

Semantics: returns the user profile information associated with the {SID}.

URL: auth/user/{SID}/update/{INFO}

VERB: PUT

Input:

{ **{INFO}**: str }

Semantics: updates the user info according to the category {INFO}, it could be one of the following: (fullname, website, bio).

URL: auth/user/{SID}/change/{PASSWORD|PHOTO}

VERB: PUT

Input:

{“password”: str} | {“photo”: image/jpeg}

Semantics: updates the user password or profile picture.

URL: auth/user/{SID}/add

VERB: POST

Input:

{“id”: int, “type”: str, “species”: str, “username”: str,
“photo”: image/jpeg, “notes”: str, “location”: int, “date”: int}

Semantics: submits a new wildlife entry to the database.

URL: guest/report

VERB: POST

Input:

{“id”: int, “report”: {“code”: int, “text”: str} }

Semantics: submits a new report about a wildlife entry to the database, the report could be about animal abuse, improper fire, fake entries ...etc.

URL: auth/admin/{SID}/delete?[which=bool]

VERB: DELETE

Input:

{“id”: int }

Semantics: submits a report deletion request to the database, either the report is fake, and the report is deleted, or the report concerns a fake entry and the entry is deleted, then the report is deleted.

URL: auth/admin/{SID}/resolve

VERB: PUT

Input:

{“id”: int }

Semantics: submits a report resolution request to the API, which marks the report as solved, this kind of report concerns animal abuse or similar issues, if found genuine the admin would contact the authorities, then resolve the report, it will be **updated** as solved in the database but not deleted.

URL: auth/user/{SID}/populate?{QUERY}[filters=dict & location=tuple & area=tuple]

VERB: GET

Output:

```
{
  {"id": int, "type": str, "species": str, "photo": image/jpeg, "notes": str, "location": int},
}
```

Semantics: fetches all the wildlife data in the database based on the user's location and the size of their map area, the results are filtered according to the user's discovery settings and a query (it could be empty).

URL: guest/populate?{QUERY}[filters=dict & location=tuple & area=tuple]

VERB: GET

Output:

```
{
  {"id": int, "type": str, "species": str, "photo": image/jpeg, "notes": str, "location": int},
}
```

Semantics: fetches all the wildlife data in the database based on the guest's selected location and the size of their map area, the results are filtered according to the guest's discovery settings and a query (it could be empty).

URL: auth/admin/{SID}/populate?{QUERY}[filters=dict & location=tuple & area=tuple]

VERB: GET

Output:

```
{
  {"id": int, "type": str, "species": str, "photo": image/jpeg, "notes": str, "location": int,
    report: {"code": int, "text": str}
  },
}
```

Semantics: fetches all the wildlife data in the database based on the admin's selected location and the size of their map area, the results are filtered according to the admin's discovery settings and a query (it could be empty).

URL: guest/download

VERB: GET

Input:

```
{id: "str"}
```

Output:

```
{"id": int, "type": str, "species": str, "photo": image/jpeg, "notes": str, "location": int}
```

Semantics: downloads a single wildlife entry from the database according to its id.

URL: guest/download?{QUERY}[filters=dict & location=tuple & area=tuple]

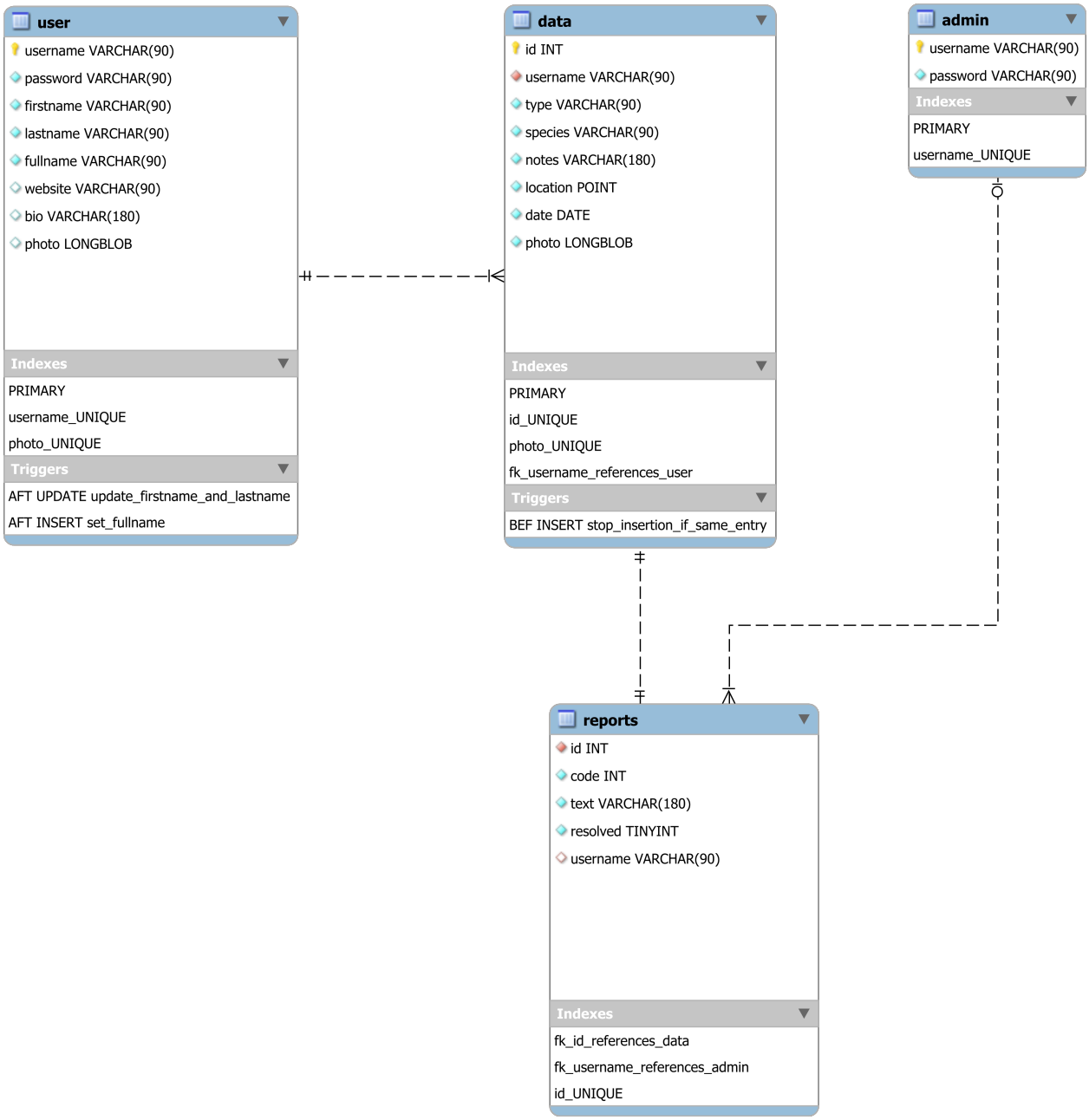
VERB: GET

Output:

```
{
  {"id": int, "type": str, "species": str, "photo": image/jpeg, "notes": str, "location": int},
}
```

Semantics: downloads all the wildlife data in the database as a file, based on the guest's selected location and the size of their map area, the results are filtered according to the guest's discovery settings and a query (it could be empty).

Entity Relationship Model:



Technologies:

In this project I've chosen Flask to use in the design of the RESTful API server, for the following reasons:

- It supports Object Oriented Design Patterns.
- It supports RESTful development right out of the box, but more so with the flask-restful extension.
- It encourages compact and minimal code.
- It has as few layers of abstraction as possible.
- It's easy to write unit tests for it.
- It supports multitudes of database technologies (MySQL, Postgres, MongoDB …).
- It allows to interface database tables directly with python classes which inherit the database template.
- It has a good CORS support.
- It's thread-safe, allowing for running background services.
- It allows for the creation of loosely couples microservices, which makes it scalable.
- It treats the user as a consenting adult, allowing for easy modification of the core code, which makes it flexible.
- It's well integrated on heroku servers, which I will use to demo the API.
- Previous professional experience with flask and the desire to improve said experience.