OurAnimals

Introduction

OurAnimals is an image library specializing in images of foxes, bears, dogs and otters.

The main goals and expected features of *OurAnimals* lay in having a fully functional user interface, a full fledged REST API based on best practices and the ease of access to the image library. OurAnimals Image Library provides images of the following animals (exclusively):

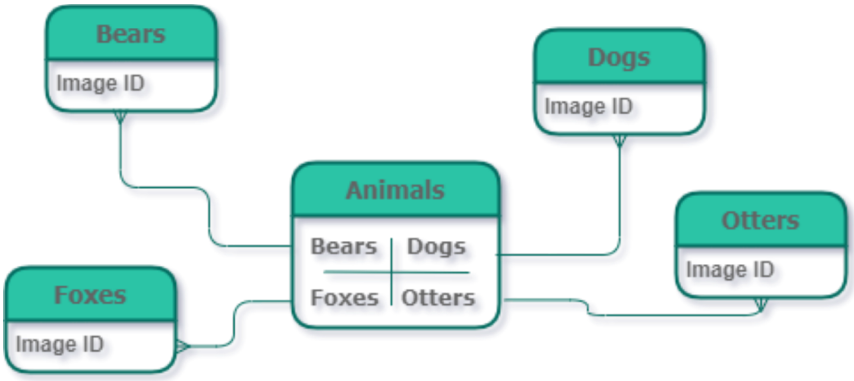
- Bears
- Foxes
- Dogs
- Otters

Design and Implementation

The REST API Specification

URI	HTTP Method	Description
/animals/	GET	List information about all OurAnimals Images
/bears/	GET	Show one bear image
/bears/{id}/	GET	Show the bear image with the specified id number
/dogs/	GET	Show one dog image
/dogs/{id}/	GET	Show the dog image with the specified id number
/foxes/	GET	Show one fox image
/foxes/{id}/	GET	Show the fox image with the specified id number
/otters/	GET	Show one otter image
/otters/{id}/	GET	Show the otter image with the specified id number

Database Schemas, Design and Structure



The above diagram is a good representation for both the **database structure** and the **schema structure** itself. *Animals* will be an "*index*" type of table based on the *many to many* relationships it has with all the individual animal tables ("individual animal tables", as in, the tables for bears, dogs, etc).

For most if not all, individual animal tables, the **Schema** will look something along the lines off:

```
const bearSchema = new Schema({
    image: {
        type: String,
        required: true,
        unique: true
    }
});
```

All images will also have IDs but I am going to use *mongoose* so the id is implicit and that's why it is not featured in the schema above.

The index style field "Animals" schema will look something along the lines of the below code example:

```
const animalSchema = new Schema({
    bears: [bearSchema],
    dogs: [dogSchema],
    foxes: [foxSchema],
    otters: [otterSchema]
});
```

Communication Overview

Since one of the goals of OurAnimals is to create a REST API, **communication between the server and the client will be facilitated using HTTP (*HyperText Transfer Protocol*)**.

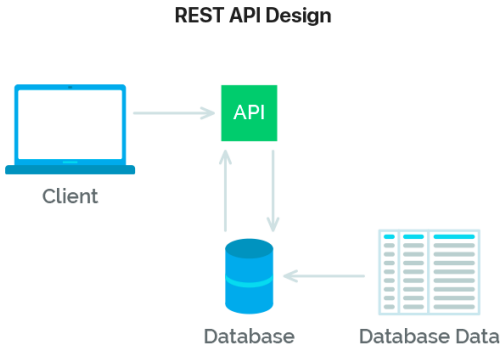
When links are clicked, urls are entered or items are searched for on the client side, a request will be sent to the server, in the form of an ***HTTP request***.

This request will include things such as,

- A **URL** identifying the target server and resource.
- An **HTTP Method** defining the action.
  - **GET** - To "*get*" (*to receive*, *show*, or *find*) a specific resource (an image).
- **URL Parameters** in the form of key/value pairs (e.g. ***/bears?id=1***) passed in as additional information encoded in the url sent to the server.

Conclusions

OurAnimals' architecture follows the REST Architectural Style. The overall structure of the application itself can be divided into 3 parts: the client, the server, and the api. They all work as if the others' did not exist. Modularity and the separation of concerns was a huge ideal in the development of this project.



I expect the most difficult part of OurAnimals API will be guarding against database injections but overall the architecture speaks for itself. OurAnimals is a simple and cute image library with a focus on bears, dogs, foxes and otters.

References

Relevant Material[3]

- ***"Architectural Patterns" by Pethuru Raj, Anupama Raman, and Harihara Subramanian***
- ***HTTP Header Fields***
- ***"REST API Best Practices - REST Endpoints Design Examples" by freecodecamp.com***
- ***RFC-2119***
- ***MicroFormats RFC-2119***
- ***W3's WIKI on RFC Keywords***
- ***RFC 2119 TXT***
- ***W3 Manual of Style***
- ***Client-Side Overview***
- ***Swagger UI***
- ***OpenAPI Specification***
- ***"Hands on RESTful API Design Patterns and Best Practices" by Harihara Subramanian and Pethuru Raj***
- ***HyperText Protocol Semantics and Content***
- ***Express Documentation***

[3] *References to any material / websites / books etc. relevant to the application idea*