Web-services & saas-services

Final project: management of an animal shelter

Clarisse Tarrou

December 15, 2019

Contents

1 Subject reminder

1. Design and develop a system, consisting of at least 3 microservices. For example – e-shop contains following microservices: basket, showcase, payment gateway, currencies e t.c. Every microsystem should implement some separate business process.

2. You may use any programming languages to create microservices (f.e. C#, Go, Java, Python, PHP e t.c.). You may use any Database to create database or databases for the system (f.e. MS SQL, PostgreSQL, MySQL, MongoDB e t.c.). You may use any technology to design interaction between your services (direct http, http2, Queue/bus e t.c.).

3. To deploy you should use Kubernetes and if you use CI/CD – any CI/CD tool like Gitlab, Github Actions, MS Azure etc.

4. Every microservice should be ready to deploy:
   (a) Docker Image to upload;
   (b) CI/CD items (yml file(s));

5. You are to write a specification, which must include following points:
   (a) Description of your project;
   (b) Endpoints of every microservice (uri, HTTP-method, arguments e t.c.);
   (c) Instructions to deploy if it's not standard;

6. For "3" mark you need to implement points 1-2.

7. For "4" mark you need to create a logging and tracking system.

8. For "5" mark you need to implement points 3-4.

2 Project specification

The system developed for this project is aimed at managing an animal shelter. In this case, we want to be able to manage animals, volunteers, and food stocks.

2.1 Database

2.1.1 Objects properties

Typical properties have been defined to characterize each of the specified objects.

Object	*Animal*	*Volunteer*	*Stock*
Properties	ID	ID	ID
	Name	Name	Specie
	Specie	Surname	Date of delivery
	Gender	Birthday	Quantity delivered
	Birthday	Working day	
	Date of arrival	Date of arrival	
	Date of departure	Gender	
	Adopter		

Table 1: Object properties

2.1.2 Tables

To define the tables needed to create the database, I used the following method:

- A type of food is linked to a species of animal.
- An animal has one species but a species can have several animals.
- A volunteer has a single working day but a working day can have several volunteers.
- A stock with a single delivered quantity but a quantity can correspond to several stocks.
- A volunteer/animal has a unique gender/anniversary/birthday/date of arrival but a date can have several volunteers/animals.
- An animal has a single departure date but a departure date can correspond to several animals.
- The fields birthday, arrival date, departure date, delivery date correspond to a single date but a date...

It will therefore require 8 tables: animals, volunteers, stocks, dates, quantities, genders, species and working days.

Animals	**Volunteers**	**Stocks**
id_animal	id_volunteer	id_stock
name	name	id_specie
id_specie	surname	id_delivery_date
id_gender	id_birthday	id_quantity_delivered
id_birthday	id_working_day	
id_arrival	id_arrival	
id_departure	id_gender	
adopter		

Table 2: Specification of tables *Animals*, *Volunteers* and *Stocks*

Dates	**Quantities**	**Genders**	**Species**	**WorkingDays**
id_date	id_quantity	id_gender	id_specie	id_day
date	weight	gender	specie	day

Table 3: Specification of tables *Dates*, *Quantities*, *Genders*, *Species* and *WorkingDays*

2.2 Micro-services

2.2.1 Actions

Micro-services	*Animal management*	*Volunteer management*	*Stock management*
Actions	Add		
	Delete		
	Mark as adopted	Consult who works for a given working day	Consult stocks by decreasing delivery date
	Consult the animals to be adopted of a species		Consult the quantities for a given species

Table 4: Actions of the different micro-services

2.2.2 Interactions between tables and micro-services

Micro-services	*Animal management*	*Volunteer management*	*Stock management*
Tables	Animals	Volunteers	Stocks
	Dates	Dates	Dates
	Genders	Genders	Species
	Species	WorkingDays	Quantities

Table 5: Interactions between tables and micro-services

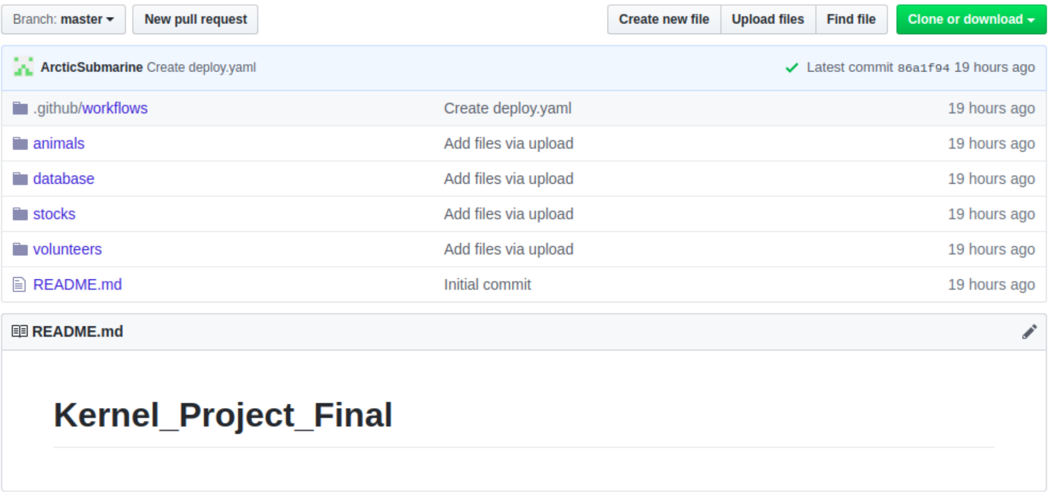
3 How to run the project ?

3.1 Structure and technologies

3.1.1 Strucure

The project is made of five files:

- *.githubworkflows*, which contains a script that allow to deploy the project;
- *animals* which contains the animal management micro-service;
- *database* which contains the files required to create the database;
- *volunteers* which contains the volunteer management micro-service;
- *stocks* which contains the stock management micro-service.



ArcticSubmarine Create deploy.yaml ✓ Latest commit 86a1f94 19 hours ago

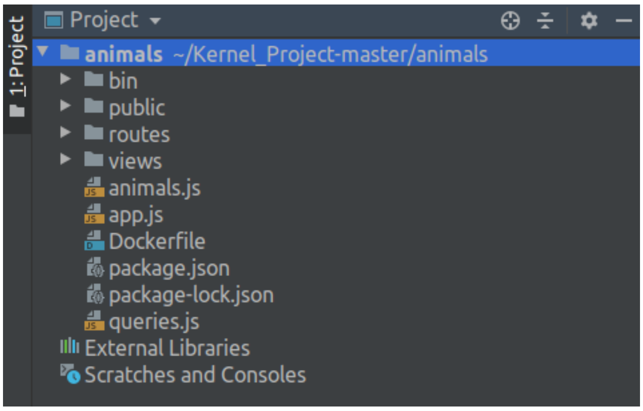
.github/workflows	Create deploy.yaml	19 hours ago
animals	Add files via upload	19 hours ago
database	Add files via upload	19 hours ago
stocks	Add files via upload	19 hours ago
volunteers	Add files via upload	19 hours ago
README.md	Initial commit	19 hours ago

README.md

Kernel_Project_Final

Figure 1: Structure of the project

As the three micro-services have the same inner structure, I will only detail one.



```
▼ animals ~/Kernel_Project-master/animals
    ▶ bin
    ▶ public
    ▶ routes
    ▶ views
      animals.js
      app.js
      Dockerfile
      package.json
      package-lock.json
      queries.js
  External Libraries
  Scratches and Consoles
```

Figure 2: Structure of a micro-service (*animals*)

Files		Decription
bin	www	Sets the port to listen and handle errors.
public stylesheet	style.css	Stylesheet.
routes	index.js	Sets the HTTP request methods.
	user.js	Get user listening.
views	error.pug	Replace variables in the files with actual values and send the resulting HTTP string to the client.
	index.pug	
	layout.pug	
animals.js		Contains the functions allowing to interact with the database.
app.js		Engine setup and error handler.
Dockerfile		Run docker.
package.json		Packages used for the project.
package-lock.json		
queries.js		Define the pool.

Table 6: Files of the micro-service *animals*

3.1.2 Technologies

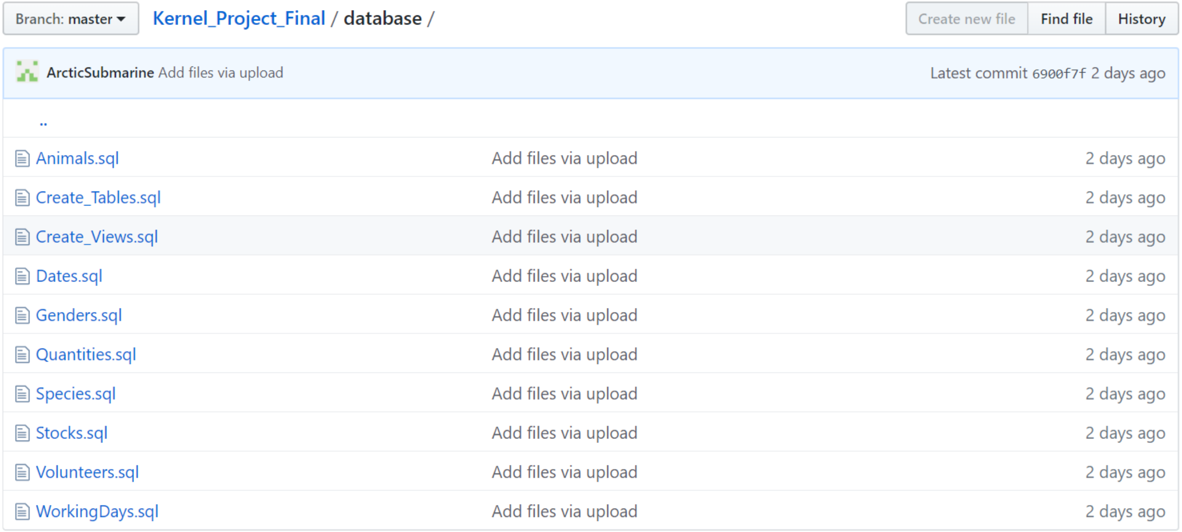
OS	Ubuntu 18.04
Languages	JavaScript (NodeJS/Express) PostgreSQL
Development softwares	WebStorm DataGrip
Deployment	Docker Google Cloud GitHub

Table 7: Technologies used to develop the app

3.2 Database set-up

The database is made of eight tables (see Tables 2 and 3 in 2.1.1): *Animals*, *Volunteers*, *Stocks*, *Dates*, *Quantities*, *Genders*, *Species* and *WorkingDays*. The database file contains ten files:

- eight to create and eventually create default values for each table;
- one to create views of the tables called by the micro-services (*Animals*, *Voluteers*, *Stocks*);
- one to fill those with values.



Kernel_Project_Final / database /

ArcticSubmarine Add files via upload — Latest commit 6900f7f 2 days ago

..		
Animals.sql	Add files via upload	2 days ago
Create_Tables.sql	Add files via upload	2 days ago
Create_Views.sql	Add files via upload	2 days ago
Dates.sql	Add files via upload	2 days ago
Genders.sql	Add files via upload	2 days ago
Quantities.sql	Add files via upload	2 days ago
Species.sql	Add files via upload	2 days ago
Stocks.sql	Add files via upload	2 days ago
Volunteers.sql	Add files via upload	2 days ago
WorkingDays.sql	Add files via upload	2 days ago

Figure 3: Files necessary to set-up the database

To properly create the database, the files should be executed in a certain order:

1. Dates.sql, Genders.sql, Quantities.sql, Species.sql, WorkingDays.sql
2. Animals.sql, Volunteers.sql, Stocks.sql
3. CreateTables.sql, CreateViews.sql

I personally used Google Cloud as host, which has the advantages of being user friendly, and offering $300 of free using -which is more than what is necessary for such a project.

3.3 Use the micro-services

Here how to use the functionalities of the micro-services, using the following tables:

- when a route is given, just use it to access the corresponding page in the browser. For instance, for the route '*/animals*', search: https://*your-domain-address*/**animals**
- when a curl instruction is indicated, run it in a terminal setting coherent parameters -respecting default ones for instance.

Function called	**HTTP request**	**How to ? (example) / Route**
getAnimals	get	/animals
getDogs	get	/animals/dog
getCats	get	/animals/cat
getBirds	get	/animals/bird
getRodents	get	/animals/rodent
createAnimal	post	curl –data "name=Madonna&id_specie=3 &id_gender=1&id_arrival=4" http://localhost:3000/animals
markAdopted	put	curl -X PUT -d "id=5" -d "id_departure=5" -d "adopter=John" http://localhost:3000/animals
deleteAnimal	delete	curl -X "DELETE" -d "id=10" http://localhost:3000/animals

Table 8: Use the animal management micro-service

Function called	**HTTP request**	**How to ? (example) / Route**
getAllVolunteers	get	/volunteers
GetVolunteersMonday	get	volunteers/monday
GetVolunteersTuesday	get	/volunteers/tuesday
GetVolunteersWednesday	get	/volunteers/wednesday
GetVolunteersThursday	get	/volunteers/thursday
GetVolunteersFriday	get	/volunteers/friday
GetVolunteersSaturday	get	/volunteers/saturday
GetVolunteersSunday	get	/volunteers/sunday
createVolunteer	post	curl –data "name=Mickael&surname=Johnny& id_birthday=5&id_working_day=1 &id_arrival=10&id_departure=12&id_gender=2" http://localhosts:3000/volunteers
deleteVolunteer	delete	curl -X "DELETE" -d "id=2" http://localhost:3000/volunteers

Table 9: Use the volunteer management micro-service

Function called	**HTTP request**	**How to ? (example) / Route**
getStocks	get	/stocks
getDogsStocks	get	/stocks/dog
getCatsStock	get	/stocks/cat
getBirdsStock	get	/stocks/bird
getRodentsStocks	get	/stocks/rodent
createDelivery	post	curl –data "id_specie=3&id_delivery_date=12 &id_quantity_delivered=3" http://localhost:3000/stocks
deleteStock	delete	curl -X "DELETE" -d "id=2" http://localhost:3000/stocks

Table 10: Use the stock management micro-service