Programming II

Project Description

Zooma: A Zoo Management Software (25 Points)

Deadline **Sunday 10.04.2022, 23:59 CET.**

Let us imagine, that we are building up a new software for management of zoos. The software shall be used by employees of the zoo to manage the animals, monitor their status and plan the tasks within the zoo.

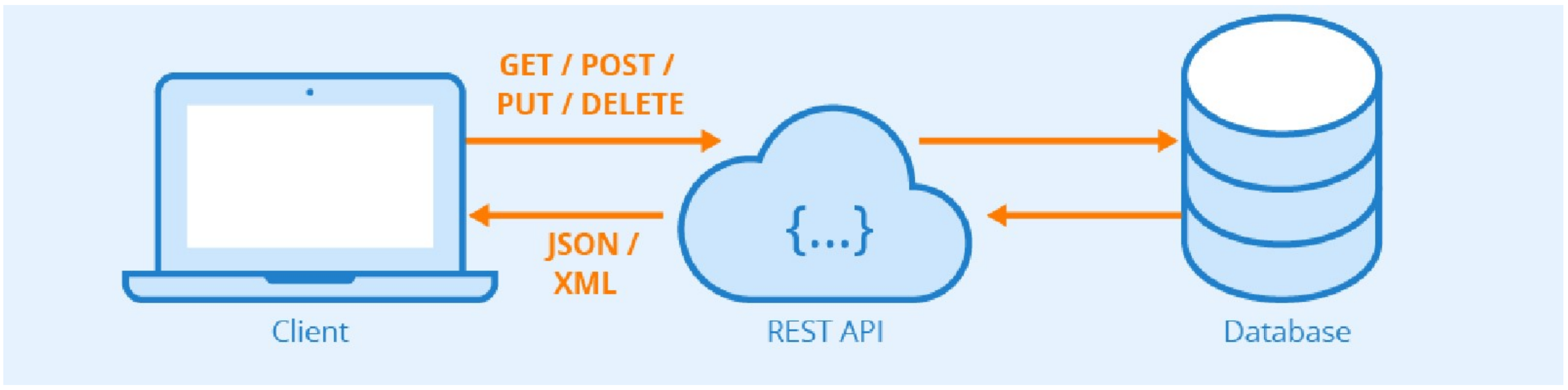
This programming task is about implementing a web API, so that the same software can be customized by different zoos to build their own applications based on the provided API. At the moment, we do not care about the front-end of the application, which could be any browser or a mobile app. And we also don't care about a database that stores the data persistently. In this exercise, we deal with the API implementation only. You may use a **REST client like Postman** to quickly see the results of the API call. It is recommended to use **Python requests library** and **Pytest** to thoroughly test the API. A **Swagger based UI** is provided, so that you can also test the application based on a browser front-end. Design your objects and classes allowing for easy future extensions

The API consists of the following functionality:

- Management of animals
  - Add/remove animals to/from the zoo. Each animal has an animal-id, age, species-name, common-name.
  - Add/remove animals to/from a zoo enclosure. Each animal has an address within the zoo (e.g., "tiger cave 213")
  - Each animal has a care-taker, who is the zoo employee, who is responsible for the animal.
  - For each animal keep track of when the animal got its food, the dates for its medical check-up.
- Management of care-takers
  - Add/remove care-takers to/from the system. Each care-taker has an employee-id, name, address and a list of animals, s/he takes care of.
  - When a care-taker is removed (e.g., quits the job), transfer all animals in his/her supervision to another care-taker.
- Management of enclosures
  - Animals in the zoo are kept in their corresponding enclosures. Each enclosure has a name and the information about available space (in sq. meters).
  - In any enclosure, there may be multiple animals, sometimes even multiple species, if different types of animals share a common enclosure.
  - Keep track of when the enclosure was cleaned.
- Monitoring births and deaths of animals
  - The system keeps track of births and deaths. When a new animal is born, it stays in the same enclosure as the mother. But may be moved to another enclosure afterwards.
  - When an animal dies, remove it from the enclosure and from the zoo.
- Management of tasks for the care-takers
  - Generate a feeding plan for the animals. Animals must be fed every second day.
  - Generate a enclosure cleaning plan for the available enclosures. The enclosure of the animals must be cleaned every third day.
  - Generate a medical check-up plan for the animals. The animals must be check by a vet every five weeks.

The API is implemented in Python using a package called Flask, which allows you to define HTML methods GET, POST, PUT etc. Each method returns a JSON object, which can be used by the front-end application in adequate ways. The summary of HTML methods to be implemented as part of the homework is listed in the following table. The first few methods have already been implemented as **examples**.

Write **Test cases to thoroughly** test your API – manual testing with Postman is good, but **not enough.** Come up with an automated testing script, which simulates the daily operations of the zoo. For example, add a few animals, feed them, take them to the vet, clean the enclosures, generate feeding and cleaning plans and simulate birth and death of the animals.



In this exercise, we focus on the API implementation only. Feel free to create your own client application based on the API as your side project. The side project is not graded but a lot of fun!

Summary of HTTP Methods:

	URL Path	Type	Description
✓	/animal	POST	Add a new animal (parameters: **age, species_name, common_name.**) This method returns all the details of the animal, including the id of the animal after it is added to the zoo. *An example is given.*
✓	/animal/<animal_id>	GET	Return the details of an animal with the given **animal_id**.
✓	/animal/<animal_id>	DELETE	Delete the animal with the given **animal_id**. *An example is given.*
✓	/animal/<animal_id>/feed	POST	Calling this method will feed the animal. Keep track of the time and date. *An example is given.*
✓	/animals	GET	Return a list of all animals with all the details about the animal.
	/animal/<animal_id>/vet	POST	Calling this method will trigger a medical checkup for the animal. Keep track of the time and date.
	/animal/<animal_id>/home	POST	Assign a home to the animal (parameters: **enclosure_id**). Make sure to remove the animal from the original enclosure it used to live in.
	/animal/birth/	POST	An animal is born. (parameters: **mother_id**). The child lives in the same enclosure as the mother and shares the species and common name.
	/animal/death/	POST	An animal has died. (parameters: **animal_id**). Remove the animal from the zoo, and also from the enclosure it lives in.
	/animals/stat	GET	Get statistics about the zoo animals: • Total number of animals per species (based on the scientific name) • Average number of animals per enclosure • Number of enclosures with animals from multiple species • Available space per animal in each enclosure
	/enclosure	POST	Add a new enclosure to the zoo (parameters: **name, area**).
	/enclosures	GET	Return the details of all the enclosures
	/enclosures/<enclosure_id>/clean	POST	Calling this method will trigger a clean-up of the enclosure Keep track of the time and date.
	/enclosures/<enclosure_id>/animals	GET	Get the details of all the animals living in the corresponding enclosure.
	/enclosure/<enclosure_id>	DELETE	Delete the enclosure with the given enclosure_id. Transfer the animals in the corresponding enclosure to another enclosure first.
	/employee/	POST	Add a new employee (care taker) (parameters: **name, address**).
	/employee/<employee_id>/care/<animal_id>/	POST	Assign an animal to a care taker. Make sure that every animal has only one care taker.
	/employee/<employee_id>/care/animals	GET	Get a list of animals under the supervision of an employee.
	/employees/stats	GET	Get statistics about the employees: • The min, max and the average number of animals under the supervision of a single employee.
	/employee/<employee_id>	DELETE	Delete the employee with the given employee_id. Transfer the animals under the supervision of the corresponding employee to another employee first.
	/tasks/cleaning/	GET	Generate a cleaning plan for all the enclosures. For every enclosure, calculate the next date for cleaning and the person responsible for cleaning.
	/tasks/medical	GET	Generate a medical check-up plan for all the animals. For every animal, calculate the next date for medical check-up.
	/tasks/feeding	GET	Generate a feeding plan for all the animals. For every animal, calculate the next date for feeding and the person responsible for feeding.

Example code implementing the first five methods is provided in a Github repository.

https://github.com/deepak-dhungana/INF-SS22-Programming-II

Submission of the project must be done through a Github Repository. Create your own repository and upload your code to your private repository before the deadline. Add https://github.com/deepak-dhungana as a collaborator to your repository. Submit the URL of your repository via MS Teams.

In order to create a new Github account, visit https://github.fhkre.ms and login with your IMC FH Krems account.