Assignment 1: RESTFUL service

Objectives

Background:

REST (Representational State Transfer) has emerged in the few years ago as a predominant Web service design model. It is an alternative to SOAP and WSDL Web services (covered during the lecture). REST is now the mainstream approach in developing Web 2.0 service.

Assignment:

In this assignment, you will be creating your very own RESTful service! Specifically, you will be implementing a *URL shortening* service. The idea of this service is act as a repository of URLs, where every URL is mapped to a unique (and as short as possible!) identifier. Then, another service or program can call your rest API to resolve the short identifier to the full URL it refers to, or to manage the URLs themselves (add new mappings, delete old mappings, etc).

Your service must adhere to the following specification:

Path & method	Parameters	Return value (HTTP code, Value)

`/:id - GET`	`:id-unique identifier of a URL`	`301, value`
		`404`

`/:id - PUT`	`:id-unique identifier of a URL`	`200`
		`400, "error"`
		`404,`

`/:id - DELETE`	`:id-unique identifier of a URL`	`204`
		`404`

`/ - GET`		`200, keys`

`/ - POST`	`:url-URL to shorten`	`201, id`
		`400, "error"`

`/ - DELETE`		`404`

This specification is left ambiguous on purpose, and it is part of the assignment to find your own interpretation of it that adheres to REST- and CRUD principles (see the slides). In particular, pay attention to your application supporting all of the paths specified, and that it can return all of the given error codes in situations that make sense. You might notice, however, that the specification says nothing about the contents of the body of a request; this is left for you to decide.

Implementing the specification will earn you a 8. For full points, also:

1. Assign *as short as possible* identifiers to new URLs, and explain your algorithm in your report.
    a. Pay attention to issues with scaling, e.g., simply trying random numbers until you found one does *not* scale, nor does assigning a number 1-10, incrementally
2. Check URL validity with a regex expression before creating a mapping for it (look at the "Important Notes"-section!).

In addition to the full points, you can earn additional bonus points for this assignment if you implement anything else not described here. These will then be used as a small bonus on your final course grade, as a way to cover less-than-perfect grades later down the line.
<u>However, you only earn them if you implemented the specification correctly.</u>
If you go for any bonus, you must describe your additions during your demo and in your report (it does not count towards the page limit if you put it in a separate section; see below).

Implementation
There are plenty of good online tutorials to help you develop and deploy your first RESTful service. We strongly suggest you implement your service in Python, using the Flask microframework [1]. However, if you really want, you can also implement your service with other libraries in other languages. However, if you choose to do so, be aware that the TAs can only provide limited support (see the "Important notes" at the end of the document!).

A list of libraries you might look at if you want to develop in other languages: Jersey (open-source reference implementation for JAX-RS, Java) [2-7], RESTeasy (Java) [8-9], Ruby on Rails (Ruby) [10] or Sinatra.rb (Ruby) [11-12].

Grading
There are two parts to this assignment:
1. *Code* – Your implementation will account for 50% of your grade. While you have to submit the code itself to Canvas, as usual (see 'Submission'), most of the grading will be done during a **demo** that you have to show to your TA during the lab sessions. Don't worry, you don't have to create any slides or anything; instead, it is meant as a moment for you to show the TAs how you interpreted the specification, how you implemented it and that your implementation works. Specifically, you should show the following:
    a. Explain how you interpreted the specification (i.e., what do your functions do specifically)
    b. Show to us that your implementation works by making HTTP requests and showing us the result
        i. <u>Tip:</u> Most clients, like Postman, can prepare requests in advance, often even in multiple tabs for easy switching; use this to your advantage!

   c. **[For full points]** Present your algorithm that generates new identifiers
   d. **[For full points]** Show us how you validate URLs and show that it works
   e. **[Bonus points]** Highlight any other aspect of your design that you think is relevant / you are proud of, most notably any bonus things you did.
2. *Report* – The other 50% of your grade is decided by a short "report" that you have to write. It doesn't have to be a full scientific report, but pay attention that your language is easily understandable and suitable for a scientific environment. Specifically, include the following in your report:
   a. Describe your implementation (max 1 page). Be specific in how you interpreted the specification and how your service exactly behaves (consider providing a similar, but more detailed, table as given in the assignment). Also highlight any important design decisions, including, but not limited to, how you implemented the ID generation algorithm and how you validate URLs.
   b. Give answer to the following question (max ½ page):
   How would you implement a URL-shortener where multiple users can create/remove URL mappings? Give a short description of how you would go about implementing this.
   c. Anything you did for the bonus part. While there is no page limit to this part, try to keep it succinct to shorten grading times.
   d. Provide a small table or description detailing which of your project members did what. What you write here won't influence your grade, but we will look at this if you encounter problems with cooperation in your group.

Note that your presentation style and language used in your report will be taken into account, although lightly (both ~9%, so ~18% in total).

1. For your presentation style, this is mostly based on whether you have made efforts to make the presentation go smoothly and whether you stay within time (10 minutes). Also note that we may deduct points if members of your groups are missing on presentation day (unless you have a good reason and let us know beforehand).
2. For the language in your report, this is mostly based on whether what you wrote is easily understandable and relevant. We might also deduct a few points if your language is too colloquial (it is a scientific environment, after all).

Submission
Your work should be submitted in Canvas, under Assignment 1. Specifically, bundle your code and your report (as PDF) in a tarball or zipfile called *<group number>_web_service_1.tar.gz* or *<group number>_web_service_1.zip*, respectively. Concretely, your archive must contain the following:

1. Your working**(!!!)** source files

2. A *README.md* that explains how to get your code up-and-running
3. Your report, as a PDF

Note that, before you submit, you should have already joined a group in the People tab, or else we won't grade your submission. If you did, however, then only one member in your group has to submit, and Canvas will count it as a submission for everyone.

Tips

a. You are designing a *REST* service, not a website. You don't have to make a graphical interface for it, but instead create a service that responds to HTTP calls as discussed in the lecture.
b. Although you must support to the specification provided, you are free to implement additional functionality and/or make additional requirements on parameters given in the body of a request (i.e., you cannot require additional parameters in the URL, but you can request additional parameters given as JSON, for example).
c. URL-shortener can store data in memory – it does not have to maintain its state between restarts (i.e., not need to connect to a database).

Important Notes:

- TAs, and the rest of the team, are happy to help you! However, they will only do so **during working hours**.
- TAs have extensive experience in Python, but they can only provide **best-effort support** if you decide to use another language than Python, or another library than listed here.
- Using code written by someone else is not a sin; however, **you must provide a source in your README.md / comments!!!** You are also expected to provide a detailled explanation of how the copied code works, to show that you understand it. **Failure to do so will be considered plagiarism!**
  - Don't hesitate to ask a TA for advice if you are unsure about this or a particular piece of code.
- Code written by ChatGPT (or similar AI assistants) does not count as your own work. And even if you provide a source, you will not get a passing grade using these tools.

Links:

1. Python Flask webpage: http://flask.pocoo.org/ (good guide at http://flask.palletsprojects.com/en/2.0.x/quickstart/)
2. Jersey web page: https://jersey.java.net
3. How to create a simple Restful Web Service using Jersey JAX RS API: http://theopentutorials.com/examples/java-ee/jax-rs/create-a-simple-restful-web-service-using-jersey-jax-rs/

4. REST with Java (JAX-RS) using Jersey – Tutorial: http://www.vogella.com/articles/REST/article.html
5. http://www.bhaveshthaker.com/13/introduction-developing-implementing-restful-web-services-in-java/
6. Build a RESTful Web service using Jersey and Apache Tomcat: http://www.ibm.com/developerworks/library/wa-aj-tomcat/
7. RESTful Java Client with Jersey Client: http://www.mkyong.com/webservices/jax-rs/restful-java-client-with-jersey-client/
8. RESTEasy webpage: http://www.jboss.org/resteasy
9. RESTful Java Client with RESTEasy Client Framework: http://www.mkyong.com/webservices/jax-rs/restful-java-client-with-resteasy-client-framework/
10. Ruby on Rails webpage: http://rubyonrails.org/ (with a good REST-guide at http://guides.rubyonrails.org/api_app.html)
11. Sinatra.rb webpage: http://www.sinatrarb.com/
12. Beginners guide to creating a REST API (Sinatra.rb, Ruby): http://www.andrewhavens.com/posts/20/beginners-guide-to-creating-a-rest-api