CS209A Project (2020 Spring)

Due: End of week 16

This project continues in the same scenario as the two assignments.

You are a Natural Language Processing Engineer, and you need to collect data from the Internet to train your models. Your data has become widely used by other members of your team and you now need to develop an application that will allow you to share training data with other members of your team. You also want to develop a way to manage storing and sharing your files in a way that will be "extensible and scalable" – that is you want to allow your friends and colleagues to also submit new training examples and allow many people to access the system and download training examples that have been submitted. You also decide you would like to be able to perform some simple analysis such as comparing how similar two files are (this may be useful in analyzing the performance of Machine Learning algorithms and considering the training data).

After some meetings you and your team have decided that the best approach is to write a client server application that will allow anyone to submit new files to the storage system, and to query the system to download the data (or some subsets) for training their models. You also decide your application will need the following services:

- A EXISTS service that allows checking whether a file has been stored,
- A COMPARE service that compares two files,
- An UPLOAD service that allows to upload one or more files to the server (it should check that it doesn't upload duplicates)
- A DOWNLOAD service that downloads a file given its reference
- A LIST service that get all files stored in server

And for part 4:

- Levenshtein_distance service that returns the Levenshtein distance between the two files.

Important: the system should work by using the md5 hash sum of the files to reference them.

Important: the system should use a standard encoding for the text files as was discussed in assignment and attempt to convert any files provided into this format

You discuss the project with your boss and he said he has heard of the concept of a "RESTful" application and you should follow this approach in building your application.

Design Requirement

After some initial reading you find that a RESTful architecture means "Representational State Transfer" and refers to an architectural style for client server applications. REST focuses on avoiding application state, making sure that important concepts of an application scenario are represented as URI-identified resources, and that all interactions with a client

through a server contain all state information that is necessary, so that the server does not have to maintain session state with clients. You still have to do a little bit more research and write up a brief description of the application as RESTful system (this should be about half a page).

Your team has made you responsible for developing and documenting the system.

Task Overview

- Part 1: Document the design and then develop an application to store and retrieve text files.
- Part 2: Implement the server
- Part 3: Implement the client
- Part 4: Implement the Levenshtein Distance measure method to compare text files

Note on part 4: In the project we are using Chinese text files, you should adapt a metric to measure either the similarity of the text files (which are in Chinese). You could use a number of different ways to do this that could be very simple or more complex... In part 2 you should just compare the md5 hash keys, in part 4 you will add a more complex comparison method that you develop using the Levenshtein Distance measure (see https://en.wikipedia.org/wiki/Levenshtein_distance).

Part 1 Design the application before you start coding

First write down a design document of your own that includes the following RESTful architecture of your system as well as additional details of your implementation and the way it works (in detail for each operation etc).

- The resources in your system (this is just the stored text files)
- The identification (URI) of the resources:
    - For Example the files should have the following URI schema: /files/md5_sum_of_file
- The operations that use the resources:
    - eg1 check if exists is by a GET html request:
        - GET /files/exists/md5_sum_of_file
    - eg2 compare files should use the following request:
        - GET /files/md5sum_of_file1/compare/md5sum_of_file2
    - eg3 Submitting a file uses a POST query where the body of the request contains the file:
        - POST /files/md5sum_of_file
    - Eg4 Downloading a file uses a GET request:
        - GET /files/md5_sum_of_file
    - Eg5 List all files uses a GET request:
        - GET

IMPORTANT The above operations should return a result in JSON format.

The overall design that you should produce and implement will be similar to the following figure:

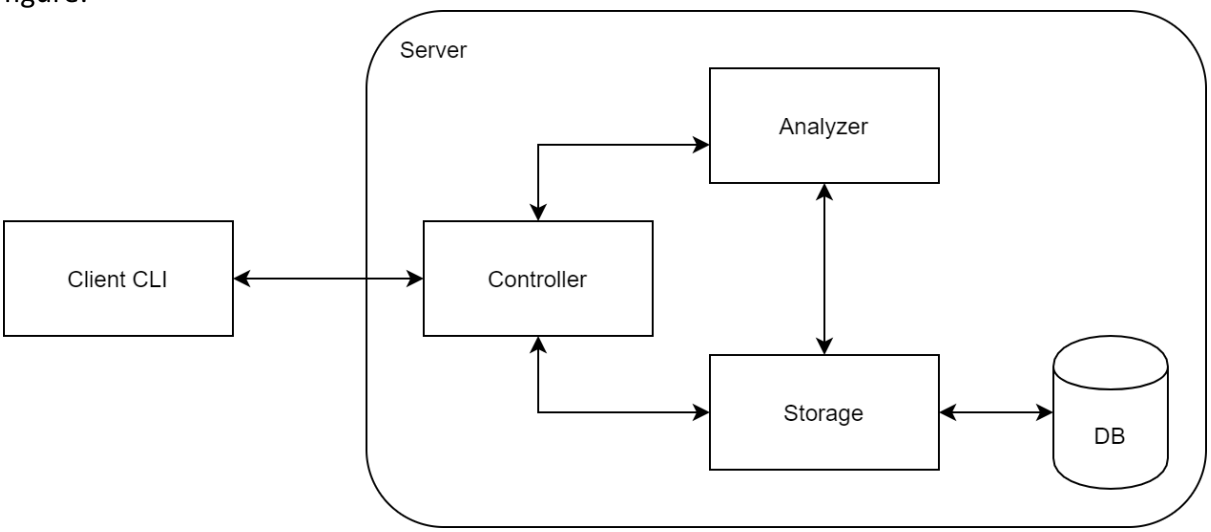

Figure 1 The overall architecture of the assignment.

The client can be a simple command line interface that provides a way to execute each of the operations. The web app API is the part of the server that listens, receives and responds to the html requests. You will use SQLite to implement the storage module (in this module each file is referenced by a key that is its md5 hash). The controller/analyser module provides a "backend" to access the database storage and perform calculations and other operations as needed.

Part 2 Implement the server application

The server application should use SQLite (https://sqlite.org/index.html) so download the binaries to run SQLite databases from the sqlite.org site and a JDBC driver (e.g. see https://github.com/xerial/sqlite-jdbc and download from their download page https://bitbucket.org/xerial/sqlite-jdbc/downloads/).

Now implement the following operations by putting appropriate code in the web app API and supporting code in the storage and analysis modules.

Comparison operation: one of the operations – comparing two text files – requires the server to perform a calculation and return the similarity of two text files. This may be useful when analyzing the results of machine learning algorithms (for example to test whether a new example is very different from the rest of the training sample). In this part you should just perform a character by character scan and see if the files have the same character at each index (ie check are the characters at index i identical or not?) and then return a percentage difference that you calculate by dividing the count of identical characters by the length of the longest file.

Operation Action	Input Representation <String>	Output Representation <JSON map> or in one case a file
Check if a file exists in storage	Header with the operation and the md5sum hash that indexes the file(GET)	A map of key value pairs: { code ->0 or error code, message ->{} or error description,

	Sample: exists/97A2383DA9FB3ED1B610CD3393E7F9C5	result -> {“exists” -> true or false}} If a file is submitted that is already in storage, exists will be “false”. Sample: {"code":0,"message":"","result":{"exists":false}}
Uploading a file	Header with md5sum of the file(POST) Body of the post request contains the file Sample: Header: 97A2383DA9FB3ED1B610CD3393E7F9C5 Body: “教学新闻我校语言中心与医学…”	A map of key value pairs: { code ->0 or error code, message ->{} or error description, result -> {}} If a file is submitted that is already in storage, message will be “already exist”. If a file is submitted that is new and it is successfully ingested, code will be 0 and message will be {}. If the file doesn't match the hash or some other error occurs, code will be an error code you define, and message will be the error description. Sample: {"code":3,"message":"already exist","result":{}}
Comparing two text files	Md5 hash key of the first and second binary(GET) Sample: Header: 97A2383DA9FB3ED1B610CD3393E7F9C5/compare/97A2383DA9FB3ED1B610CD3393E7F9C6	A map of key value pairs: {code ->0 or error code, message ->{} or error description, result -> { "simple_distance"-> different value}} Return a JSON map specifying the results of the comparison – that how different are the files based on the character by character difference normalized by the file length? If the md5 file key doesn’t reference a file in storage or no result is generated the this is returned: {"code"->error code, “ message” -> error description, "result":{}} Sample: {"code":0,"message":"","result":{"simple_distance":1.0}}
Downloading a file	Header with md5sum of the file(GET) Sample: Header: 97A2383DA9FB3ED1B610CD3393E7F9C5	Returns a document that is the text file as plain text or if unsuccessful the following is returned: {“ message” -> error description} Sample: Success: {"code":0,"message":"", "result": {"content" :“教学新闻我校语言中心与医学…”}

		Unsuccess: {"code":1,"message":"file not found","result":{}}

list	GET
	No URL parameters
		Return a json like the following format:
		{
		"code": 0,
		"message": "",
		"result": {
		"files": [{
		"md5":
		"fac27083ec676a0999e6d22e4c40f31a",
		"length": 20,
		"preview": "这是一段测试文本:我能吞下玻璃而不伤身体"
		}, {
		是一段超长文本!!这是一段超长文本!!这是一段超长文本!!这是一段超长文本!!这是一段超长文本!!这是一段超长文 本!!这是一段超长文本!!这是一段超长文本!!"
		}]
		} }

Part 3 Implement the client application

The client will be implemented as a command line interface (CLI). It has two main purposes:

1. It will take a simple command line argument and process it to send a html request in the correct format to the server.
2. It will perform an operation that allows bulk uploading of a directory files by sending multiple POST requests to upload files and then confirm whether the upload was successful (by checking if the files exist).

Command	Operation Action
EXISTS filename	Check if a file exists in storage
UPLOAD filename	Uploading a file
COMPARE filename1 filename2	Comparing two text files
DOWNLOAD filename	Downloading a file
LIST	List all files in server

Part 4 Add a more advanced file comparison method

In this part we add a more advanced comparison method to compare the degree of difference files instead of just checking if they are the same or not using the Levenshtein distance: see https://en.wikipedia.org/wiki/Levenshtein_distance or https://baike.baidu.com/item/%E7%BC%96%E8%BE%91%E8%B7%9D%E7%A6%BB/8010193?fromtitle=Levenshtein%20Distance&fromid=792226&fr=aladdin

Marking Scheme / Rubric

Component	Description	Marks (total = 100)
Code Documentation and formatting	• Use JavaDoc • Well formatted with descriptions of all classes, fields, and including return values, method parameters etc. • Complex routines have comments • Follow the architecture in Figure 1	10
Part 1 (Design Document)	• 1 Page = 450 – 650 words • Short description of RESTful interface is correct • Short description of Model View Controller design is reasonable • To get full marks the descriptions should refer to the problem in the assignment (not just be general abstract textbook type descriptions).	10
Part 2 Implement server	• Equal weightings for each of the server functions (exists, upload file, …) and should work with tests on a variety of different files and request formats • Error messages should be returned for certain situations: e.g. storage problem, multiple uploads of the same file, issues with file type (cannot upload a binary file) • Follow the specification for requests and response formats in the table • Use GET and POST requests correctly (including putting the file in the body of the request when doing upload)	40
Part 3 Implement client	• To get full marks the CLI should be presented well – does it show a list of options/help? Does it check for basic errors in the arguments? And so on. • Correctly creates the requests for each of the server commands (equal marks for each) • The function to upload several files works properly	20
Part 4 Levenshtein distance	• The Levenshtein distance function implementation should be adequately documented • The function should be useable in the server and the client	20

	• The function should work correctly to provide a way to measure Chinese text (the specific way should be defined in your code documentation of the method but should generally make sense and follow the accepted definition of the Levenshtein distance)	