Project 1

RESTful PDA processor

In the previous project, you implemented a simulator for PDAs. Your simulator is effectively a processor of PDA "programs". We will call your simulator a ***PDA processor***. In this project, you will implement a RESTful API in GoLang for a client-server for the PDA processor. Your implementation will manage PDA programs for a PDA processor.

What to do:

Task 3a.

Study the RESTful web services tutorial. Take note of its guidelines for designing RESTful APIs.

Task 3b [95% of points].

Implement a RESTful API for a PDA processor (ie. Web server that handles HTTP requests for calling methods of a PDA processor in accordance to the RESTful architectural paradigm).

Extensions to the PDA API.
First, modify the PDA from project 0 to present tokens out of order. Despite presenting tokens out of order, the PDA will still consume tokens in order (by FIFO-queuing tokens whose immediate predecessor token has not been consumed yet). The signature of the put method will include the position of the token in the input token-stream**:** ***put(position, token)***. The 1st token in the input token-stream has position 0, the 2nd token has position 1, etc.

Second, implement a method ***queued_tokens()*** that returns the tokens that have been presented but not consumed yet, in increasing order of their position in the input token-stream().

RESTful API to the PDA processor
Let ***base*** be the URL for your PDA processor: eg. http://localhost:8080/
The following URLs should be handled by your implementation:

HTTP method	URL	Meaning
GET	base/pdas	List of names of PDAs available at the server
PUT	base/pdas/*id*	Create at the server a PDA with the given id and the specification provided in the body of the request; calls open() method of PDA processor
PUT	base/pdas/*id*/reset	Call reset() method
PUT	base/pdas/*id*/tokens/*position*	Present a token at the given position
PUT	base/pdas/*id*/eos/*position*	Call eos() with no tokens after (excluding) position
GET	base/pdas/*id*/is_accepted	Call and return the value of is_accepted()
GET	base/pdas/*id*/stack/top/*k*	Call and return the value of peek(k)
GET	base/pdas/*id*/stack/len	Return the number of tokens currently in the stack
GET	base/pdas/*id*/state	Call and return the value of current_state()
GET	base/pdas/*id*/tokens	Call and return the value of queued_tokens()
GET	base/pdas/*id*/snapshot/*k*	Return a JSON message (array) three components: the current_state(), queued_tokens(), and peek(k)
PUT	base/pdas/*id*/close	Call close()
DELETE	base/pdas/*id*/delete	Delete the PDA with name from the server

The *id* of a PDA is a positive integer and is independent of its name in its specification. Any warning or error messages generated when calling methods of a PDA should be returned in the HTTP response.

Full credit will be given only to correct implementation with exactly-once RPC semantics.

Task 4 [5% of points].

Implement a driver along the lines of the earlier Task 1 to create a PDA processor and feed it an input token-stream. Your driver for this task can be a simple HTML file that contains a sequence of hyperlinks for the appropriate HTTP requests to your server; or a Bash script that sends the relevant HTTP requests to your server.

What to submit:

Submit a .tar.gz archive with your

- complete GoLang code
- sample input files (with PDA specs or marshalled token-streams)
- a Bash script for startup/shutdown of your server
- a (Bash/HTML) script file with commands demonstrating the execution of your client/driver with different command-line arguments
- sample output (text output of screen shots) of the execution of your client
- README file (with relevant documentation and usage guidance)