Go

13/2/2020

Overview

In this project, we initiate the learning of Golang. We explore its various native functions and its utility in web development. The best feature of Go being an open source programing language. There are many Pros and Cons for this language but it is important to note that Go gives the developer more access to its features than many high level languages owing to possible increase in efficiency and better optimization. The Pros and Cons have been explored in the following section.

- ***Pro: ease of use***
    - While Go might not be as popular as JavaScript or Python, it does have one important thing in common with them: it's eminently easy to understand. The syntax is clean and accessible to newcomers, and there aren't a lot of complex functions to learn. But that slick and clean syntax offers advantages to more than just newbies. That's because it's an easy language to read as well, and that makes it a great choice for legacy code that may involve multiple coders iterating over each other's code blocks. The close similarities to C-style languages mean that it will be a simple language to learn for programmers proficient in C++ or C#.
- ***Con: sometimes too simplistic***
    - One of Go's main advantages is also one of its biggest weaknesses. Go may be an easy language to pick up, but that brings with it a lack of versatility. Some of the hottest languages on the market pride themselves on their complexity. Choices like Swift and Haskell may be more difficult to learn, but they manage to find their fans by packing in a wealth of smart abstractions that allow coders to achieve complex and clever results with less. By stripping out this high-level functionality, Go also sacrifices range.
- ***Pro: A smart standard library***
    - Go users can accomplish a lot without having to import or learn complicated secondary libraries. The standard library that comes packaged with Go is sophisticated without overwhelming, and it reduces the risk of errors from conflicting function names. Take the example of slices. It's one of Go's smartest additions to the programming world, and they offer a simpler way for incorporating

data structures into your code blocks. A number of tasks that would require complicated workarounds in other languages can be accomplished with a single line of code through the Go interface.

- ***Con: It's still a young language***
    - There's a lot of promise in Go, but it's a language still in its adolescence, and that means that in many ways it can't compete with its older siblings. Go's native library may be smartly designed and efficient, but it's competing with languages like Java that comes supported by a huge collection of code built in and a cottage industry of new libraries created by an enthusiastic and engaged community. While Go may eventually catch up with its peers, it still has a long way to go in terms of library support.
- ***Pro: Strong security built-in***
    - Simpler code is generally safer than complicated code, and that's absolutely the case with Go. As a statically typed language, you don't have to worry about complex and hard to identify errors that come from the huge number of variable types present in more dynamic languages. Then there's the included garbage collector which helps prevent memory from bleeding off in your code. While the lack of generics means that coders need to be more diligent in running tests, identifying errors is easier than it is in many alternatives, and that generally promotes a more thorough approach to writing clean code.
- ***Con: Lack of a virtual machine***
    - While the decision to not base Go on a virtual machine was a conscious choice, and it's one that comes with some distinct advantages designed for ease of use, the bad may very well outweigh the good here. There's a reason that so many of the popular languages today are based off of VMs. VMs offer more efficient code, and that means that Go file sizes often dwarf those of competing programming languages.

Objectives of project

- Understand various native functions of GO theoretically
- In the practical part understand how http requests are returned in different types of content type
- Request to return only the following headers:
    - HTTP status code
    - A custom header
        - Example: Server ➜ A Go Web Server
- Request to return an HTML template
    - You can use any HTML code, the important thing is that you know how to make the API return the request from an HTML file
- Request to return XML
    - The API must return an object that contains the following structure:
- <Profile>
-   <Name>Your name</Name>
-   <Hobbies>

- <Hobby>Hobby 1</Hobby>
- <Hobby>Hobby 2</Hobby>
- </Hobbies>

</Profile>

- Request returned by JSON
    - The API must return an object that contains the structure used previously but in json language
- Request to return plain text
    - The API must return information of your choice in plain text
- Request to return an image
    - The API must return an image of your choice obtained through a file

Table of Contents

Project requirements

- Requirements documentation.
- List of tasks to be performed.
    - Priority of each task
    - Title and description of each of them
    - Difficulty level
    - Estimated time for each task.
- Record of incidents that were detected during the execution of the pill.
- Documentation about the git WORKFLOW you are going to use
- Documentation about the tools used in the project
- Record of lessons learned.

Risk Management

Every project has risks. These risks must be taken into account to improve the workflow of the project. Managing these risks is important for due completion of the project. Unchecked risks could not only lead to hamper of project deliverance but also a badly executed project. The risks hence associated with the project are documented as follows:

Risk	Risk level
Unfamiliarity with Go lang	Medium to High
Installation of Go lang	Medium to Low
Health and computer issues	High
Delivering in time	Medium
Completing all the project requirements	Medium

Task Management

The following are the tasks which needed to be accomplished for the completion of the project. The tasks have been divided according to the project specifics. Each task has been clearly defined and their priority as well as their difficulty and time needed. Difficulty level is explained on a scale of 1 to 5 (5 being the most difficult). Priority is explained on the level of 1 to 5 (again 5 the highest parameter being the most prioritized work).

Task	Priority	Description	Difficulty	Time
Read the description of the project	5	This task involves complete understanding of requirements for the project	1	30 min
Create git repo	5	Creating of git repository for project execution and delivery	1	2 mins
Create files for the project	4	Creating the necessary files, folders and subfolders for this project.	1	5 mins
Install Go	4	Install Golang in the machine	2	10-20 mins
Theoretical part	2	Write the uses of various functions in Go	3	30 mins
Http status code	2	Create a http request to return its status	4	2-2:30 hr
Return Html template	2	Use an http request to return an html template	4	30 mins
Return xml template	2	Use an http request to return a xml template	4	30 mins
Return image	2	Use an http request to return an image template	4	30 mins
Return JSON	2	Use an http request to return an JSON template	4	30 mins
Return plaintext	2	Use an http request to return an plaintext template	4	30 mins

Presentation	1	Create a presentation for the theoretical part	2	1 hr
Documentation	1	Create a project documentation	1	1-1:30 hr

Defining this part is crucial to the development of the project. It is important to make a good analysis of the situation to organize the project in a good way.

Chronogram

This shows the time-line it took for the project to finish and also the tasks that were achieved during those timeline.

The tasks have been mentioned in the left axis

Tasks	Fri 1500 hrs	Fri 1700 hrs	Mon 1000 hrs	Mon 1100 hrs	Mon 1200 hrs	Mon 1400 hrs	Mon 1500 hrs	Mon 1600 hrs
Read project descriptio-n	**X**							
Git Repo	**X**							
Create files	**X**							
Install Go	**X**							
Theoretical part								**X**
Http status code		**X**						

Return html			X					
Return xml				X				
Return json				X				
Return image					X			
Return plain text						X		
Documen tation							X	X

Git Workflow

For this project, all commits we'll be pushed directly to the Master branch. All commits will use a descriptive message, so that the admin and other internet user can follow through the processes.
For more information go to :
https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

Incidents

- None

Technologies used

For this project, we will use the following technologies:

- HTML
- Postman
- GO