# go-naming-conventionSummary

This document describes the **“go-naming-convention”**/“**gocasesystem**”.

gocase API in GOLang is heart of this system. The intent is to provide documentation that partner developers can leverage to understand how to interact with the system/ api at technical level. This document details the operations that are available, shape of requests/responses for those operations, partner visible effects of the operations, errors that can be returned as part of those operation and how to interpret these errors.

# Key resources

|  |  |
| --- | --- |
| Codebase | <https://github.com/surenderssm/go-naming-convention> |
| Docker | docker pull surenderssm/testapi |
| Hosted Endpoint | |  |  | | --- | --- | | Central US | <https://gocase-cus.azurewebsites.net/v1> | | East US 2 | <https://gocase-eus2.azurewebsites.net/v1> | | ATM | http:// | | Sample request | <https://gocase-cus.azurewebsites.net/v1/name?token=onetwothree&type=camel>  <https://gocase-cus.azurewebsites.net/v1/track?trackingid=bicm1hjg4men0e3ue2og>  <https://gocase-eus2.azurewebsites.net/v1/name?token=booktrue&type=pascal> | |
| Interactive GOCaseDemoTerminal | <https://gocase.azurewebsites.net/>  (*Do check this out*) |
| Environment | Azure App service compute running on Alpine Linux   * OS version: Unix 4.4.0.128, Processor count: 1 * B1, 100 total ACU ,1.75 GB memory, A-Series compute equivalent * ~36 $/month(estimate) |
| Telemetry, Alerts | Azure application insight, Azure Log alerts |

# Conceptual Model

At a high level (and in simplified terms), gocase API convert the text to specified case (format). Naming convention is the core of any programming language or system.Extracting tokens out of given text is the core step in Lexical analysis, first step of compiler/ interpreter.

In our case tokens are extracted based on valid English words.

Terms/concepts

* **token**:
  + A free text, made out of only English alphabets
  + At least of length 3, without any space, special character
* **type (casetype)**:
  + intended naming case which has to be applied after extracting the words
  + Following case are supported in the system

|  |  |
| --- | --- |
| camel | CamelCase <https://en.wikipedia.org/wiki/Camel_case> |
| lowercamel | LowerCamelCase <https://en.wikipedia.org/wiki/Camel_case>  (~CamelCase) |
| pascal | PascalCase <http://wiki.c2.com/?PascalCase> (~UpperCamelCase) |
| uppercamel | UpperCamelCase <https://en.wikipedia.org/wiki/Camel_case> (~PascalCase) |
| snake | SankeCase <https://en.m.wikipedia.org/wiki/Snake_case> (lowerCase + "\_") |
| darwin | DarwinCase <https://en.wikipedia.org/wiki/Camel_case> The combination of "TitleCase " and "snake case" |
| title | TitleCase |
| lower | LowerCase |
| upper | UpperCase |

# System Overview



1. GoCaseAPI
   * Rest endpoint written in GOLang to process the tokens
   * Hosted via docker container in Azure App service over image of Alpine Linux
   * Hosted in Azure paired region Central US, East us 2 for BCDR
2. GoCaseATM
   * Traffic manager to route the traffic to GoCaseAPI
   * Client system are exposed the URI of the same
   * Affinity based routing , Health endpoint is configured to watch the health of the node
3. Platform
   * Storage
     1. Azure blob store is used for all storage need of the system
     2. Results of long running operations are stored in store
   * Application insights
     1. Telemetry needs of the system
     2. Azure Alerts / log alerts on metric to keep tap on pulse of the system
4. GoCaseDemoTerminal
   * Webpage hosted in azure appservice , to enable users to play with the endpoint
   * Interactive terminal to test operations

Commands like “case onetwothree” => oneTwoThree

# GOCase api Operations

## Get name

Process the given token to return the result in the specified case (format)

|  |  |
| --- | --- |
| **Operation** | GET /{api-version}/name?token={text} &type={casetype} |
| **Summary** | Gets the processed token in the given case |
| **Success Responses** | 200 StatusOK  NamingModel structure is returned in the response payload. |
| **Error**  **Responses** | 404 – StatusBadRequest, for any invalid token  413 – StatusRequestEntityTooLarge, if token length exceeds 200. For large token size use POST |
| **Sample** | <https://gocase-cus.azurewebsites.net/v1/name?token=helloworld&type=camel> |

### Response Payload

#### schema NamingModel

// NamingModel case model

type NamingModel struct {

    Token string `json:"token"`

    Result string `json:"result"`

    CaseType string `json:"caseType"`

    TrackingID string `json:"trackingID"`

    TrackingURL string `json:"trackingUrl"`

    Message string `json:"message"`

    StatusCode int `json:"statusCode"`

}

## POST name

Asynchronously process the given token to return the result in the specified case.

Asynchronous / Long Running operation, with this work is submitted and client has to track the same at the Tracking URL

|  |  |
| --- | --- |
| **Operation** | POST /{api-version}/name |
| **Summary** | Asynchronously process the given token to return the result in the specified case.  Optimal for Long Running operation or bigger token |
| **Success Responses** | 202 StatusAccepted  Schema ***NamingModel*** is returned in the response payload.  Client has to hit URI given in TrackingURL of the NamingModel to get the result  TrackingURL will have ~/Track/{trackingID} to get the result |
| **Error**  **Responses** | 404 – StatusBadRequest, for any invalid token  413 – StatusRequestEntityTooLarge, if token length exceeds 1000 |

## GET TRACK

Retrieve the result of the work submitted via POST / name

|  |  |
| --- | --- |
| **Operation** | **GET /track?trackingid={trackingID}** |
| **Summary** | Retrieve the result of the work submitted via POST / name |
| **Success Responses** | **200 – StatusOK,** if the result has been computed  202 – StatusAccepted, indicates result is still not computed.  Caller is expected to retry  Schema ***NamingModel*** is returned in the response payload. |
| **Error**  **Responses** | 404 – StatusBadRequest |

## GET HEALTH

Get the health snapshot of the node.

This can be used to emit health of dependency or the signal like memory, CPU, I/O, Certificates, other artifacts of the node

|  |  |
| --- | --- |
| **Operation** | **GET /**{api-version}/health |
| **Summary** | GET health |
| **Success Responses** | **200 – StatusOK,** if the result has been computed |
| **Error**  **Responses** | 500 – Issue |

## GET PING

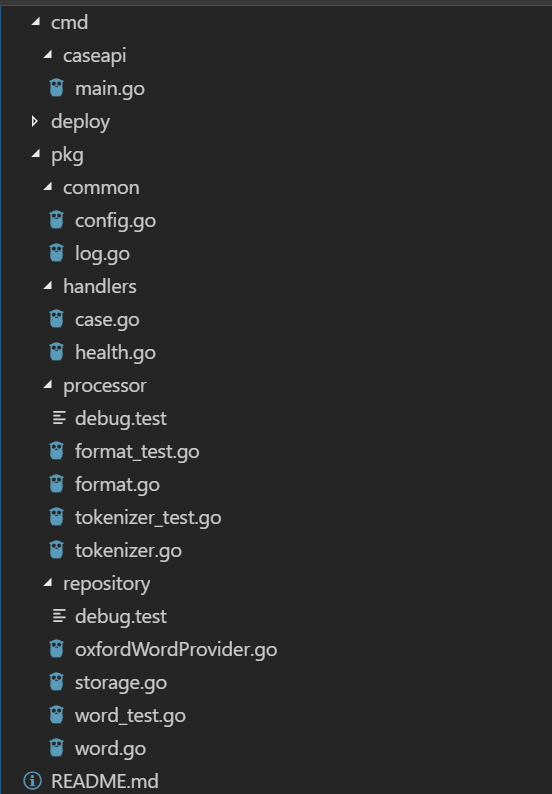
PING of the service, this will be configured at Azur traffic manager, to check availability of the service

|  |  |
| --- | --- |
| **Operation** | **GET /**{api-version}/health/ping |
| **Summary** | GET health |
| **Success Responses** | **200 – StatusOK,** if the result has been computed |
| **Error**  **Responses** | 500 – Issue |

# CODEBASE – Overview

GOCaseAPI is powered by awesome Golang.

Details of package and dependency are listed below.



1. package handlers
   * all the http handler to serve the request
2. package common
   * log (application insight), config – common artifacts of the system
3. package processor
   * contains the core logic of tokenization
   * tokenizer.go – handles the split of text to valid tokens
   * fomat.go – handles the formatting of the text in the given case
   * takes dependency on package repository
4. package repository
   * acts as a repository of the various provider
   * words.go – handles the operations around word
   * storage.go – deals with azure storage
   * oxfordWordProvider.go – one of the word provider

# Assumptions

1. Word to qualify as valid, it should be of at least length 3 and should just have English alphabet
2. There can be multiple possible solution of valid words which can be generated out of a given text. As of now program terminate at first occurrence of valid word. For eg : “basketball” can be broken into [“basketball”] or [“basket”,”ball”]
3. Invalid token / unsupported case will lead to 400
4. If text cannot be broken into valid tokens, empty string is being returned

# Product backlog

1. SSL cert for FQDN via ATM, to stop HTTP traffic and allow only HTTPS.
2. Long running operations POST/NAME is using go routines, this can be improved by keeping PUB-SUB model, compute can be moved to light weight on demand azure function.
   1. Or keeping a check on go routine itself to avoid starvation of resources
3. Cache provider like redis can be used instead of in-memory
4. Pool of wordProvider like Oxford, can be maintained to avoid throttling of single provider
5. In recursion of tokenizer.process left and right node can be computed in parallel with go routine to speed up the compute. This approach will have more hit of word provider, as memorization of the records might not be that effective
6. Memorization of valid tokens is done per request; this can be moved to centralized cache
7. Threshold on length of input token can be exceeded if system can afford compute
8. If required Support multiple languages
9. Integration / BVT tests, which can be run post deployment to check sanity of the node