2016

RESTful API Framework: Golang Proof of Concept

Gerard Briones
Virginia Commonwealth University

David Igou
Virginia Commonwealth University

Aaron Throckmorton
Virginia Commonwealth University

Team Members: Gerard Briones, David Igou, Aaron Throckmorton
Faculty Adviser: Robert Dahlberg, PhD
Sponsor: Capital One

Sponsor Mentor: Abhi Ghosh

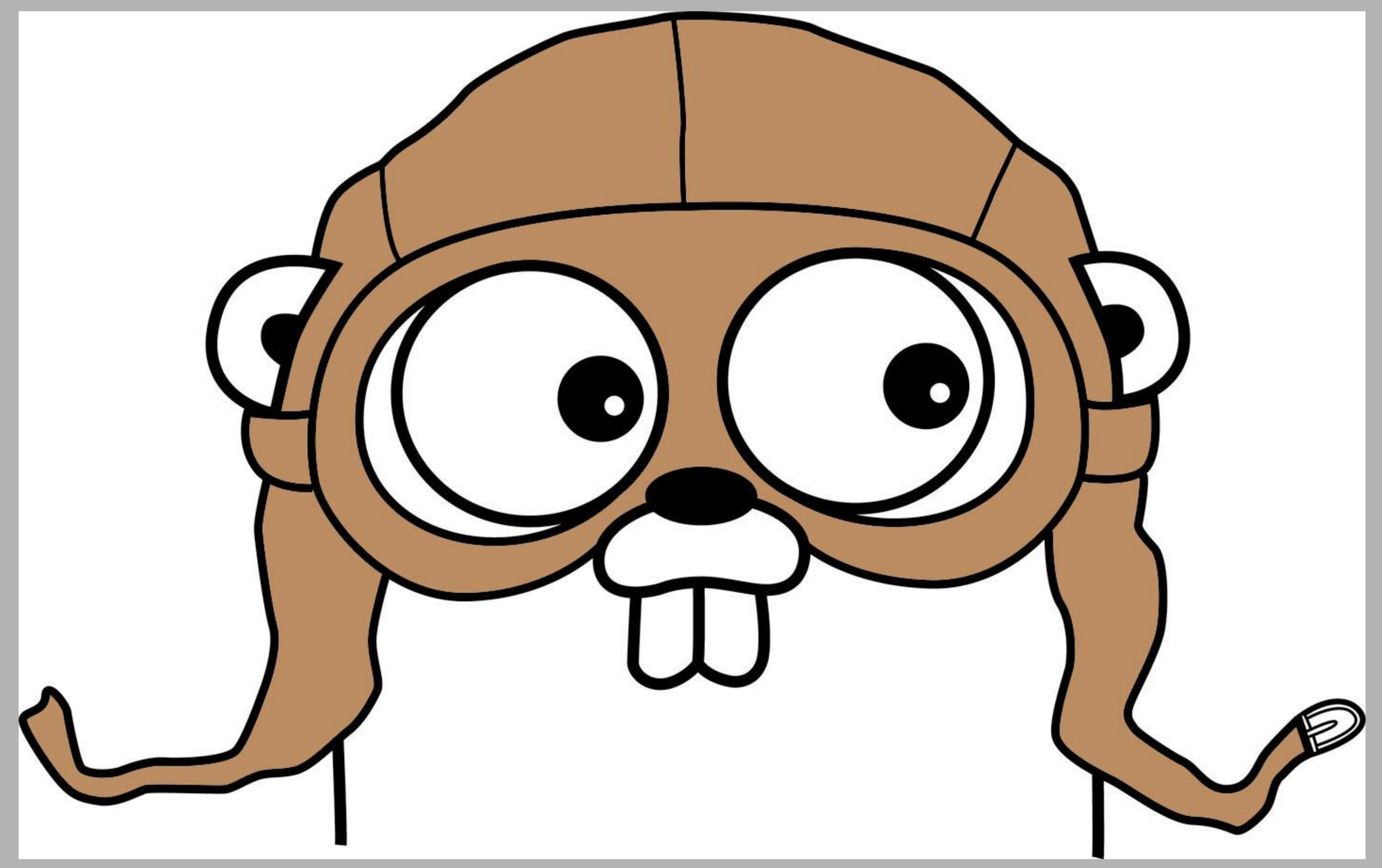
COMPUTER SCIENCE

RESTful API Framework

Golang Proof of Concept

CAPSTONE DESIGN EXPO 2016

The Right Tool for the Job



The aim of this project was to develop a RESTful API framework using Go as its foundation. This framework would serve as a bridge to simplify accessing Capital One's webservices.

The **RESTful** architecture focuses on providing a simple and uniform methodology of acquiring resources and services through the web.

- The client-server model shifts the focus from how the other side is handling the data to the data itself, reducing tasks for both sides.
- Communication is further streamlined through a stateless protocol. Each request from any client contains all the information necessary to service the request.
- The uniform interface simplifies and decouples the architecture, enabling each part to evolve independently.

Golang (Go) is an open source programming language developed by Google and other contributing members from the open source community. Go emerged as a premier language for systems development.

- Concurrency is built into the language, offering better flexibility and performance for complex applications
- Unit testing is integrated and simple to use
- Scalability is a main focus with multicore and multithreading support

The Team at Work

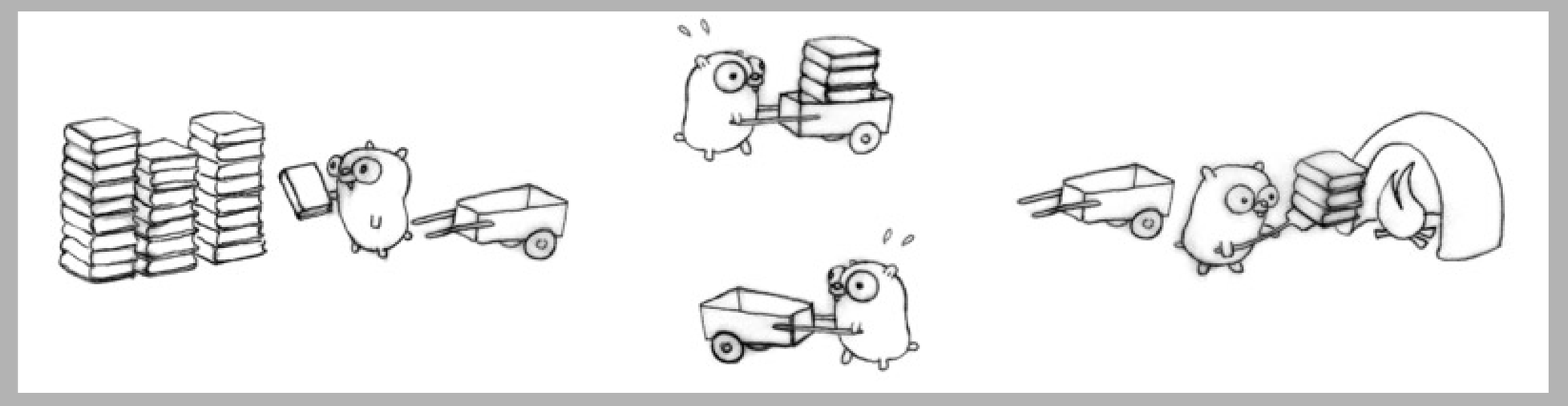
With a clone of a popular framework called Go-Restful, we were tasked with adding key features to handle important cross-cutting concerns. Dependency management was also a focus to ensure the framework would be deployment ready.

Versioning

Custom headers for HTTP requests were made to indicate which version of the resource or service was needed.

```go
func main() {
    kv := new(keyvalue)

    restful.RegisterEntityAccessor("application/test+v1", kv)

    // spin up a new webservice
    ws := new(restful.WebService)

    // create path to /testing
    ws.Path("/testing").
        Consumes(restful.MIME_JSON).    // standard json input
        Produces(restful.MIME_JSON)     // standard json output

    ws.Route(ws.GET("/").To(hello).
        Consumes("application/test+v1").    // consumes our custom accept header
        Produces("application/test+v1"))    // produces our custom accept header

    restful.Add(ws)
    http.ListenAndServe(":8080", nil)
}

// checks the Accept header of request
func hello(req *restful.Request, resp *restful.Response) {
    // fmt.Println(req.Request.Header.Get(restful.HEADER_Accept))

    // print VERSION 1 if header == application/test+v2
    if(req.Request.Header.Get(restful.HEADER_Accept) == "application/test+v1") {
        fmt.Println("VERSION 1")
    } else { // print DEFAULT if header == "" or "*/*"
        fmt.Println("DEFAULT")
    }

    // any other header will be unaccepted!
}
```

Dependency Management



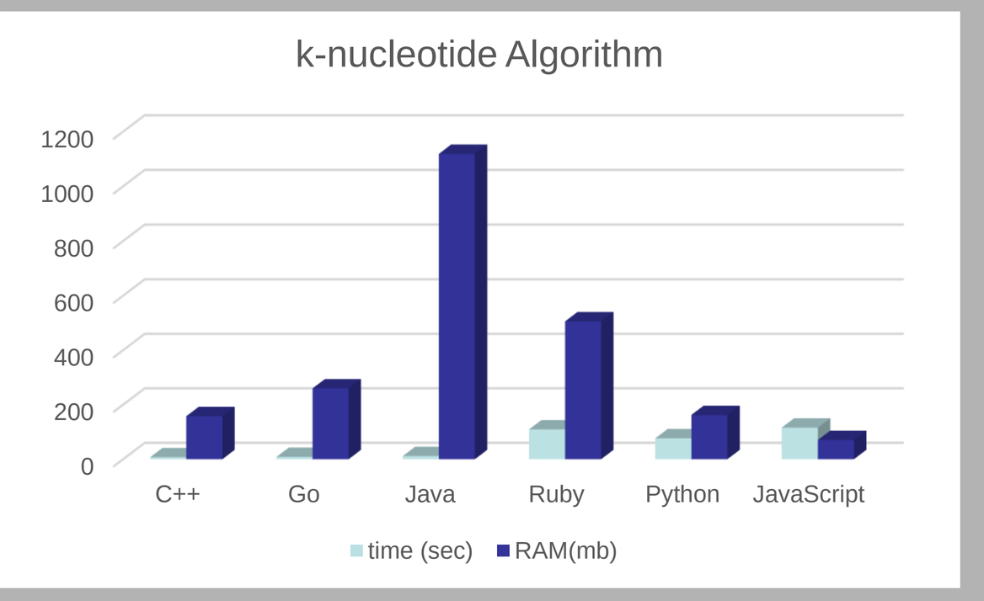
Through the integration of the GoDep library, the framework is now capable of consolidating external modules and packages to keep everything up to date. Snapshots are kept to make code redistribution even easier.

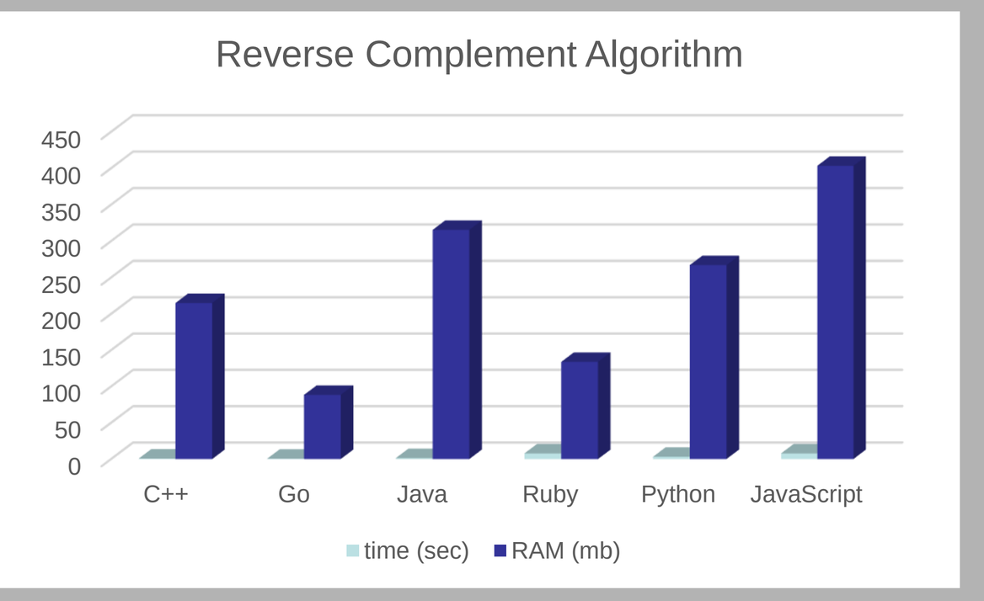
```go
//Creates Logger before RouteFunction is executed via wrapping
func LoggingAspect(fn restful.RouteFunction) restful.RouteFunction {
    return func(r *restful.Request, w *restful.Response) {
        log.Printf("[WebserviceLogging] {%s}, \n%s  \n%s  \n%s  \n%s  \n%s\n",
            strings.Split(r.Request.RemoteAddr, ":")[0],
            "[Method]: "+r.Request.Method,
            "[URL]: "+r.Request.URL.String(),
            "[Protocol]: "+r.Request.Proto,
            "[Status]: "+strconv.Itoa(w.StatusCode()),
            "[Length]: "+strconv.Itoa(w.ContentLength()),
        )
        fn(r, w)
    }
}

//Creates Authentication at RouteFunction level.
func AuthAspect(fn restful.RouteFunction) restful.RouteFunction {
    return func(r *restful.Request, w *restful.Response) {
        token := r.HeaderParameter("X-AUTH-TOKEN")
        if token == "" {
            http.Error(w, "missing auth token", http.StatusUnauthorized)
            return
        }
        //Write own authentication mechanism here.
        fn(r, w)
    }
}
```

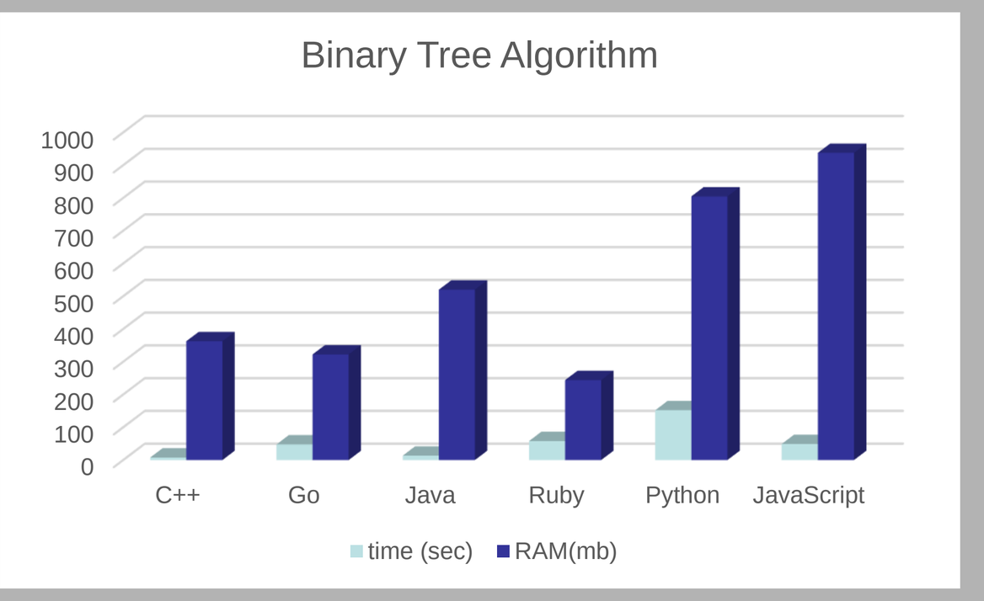
Aspect Oriented Programming

By wrapping customized functions into the existing logic, we were able to modify the behavior of the framework without directly modifying its code.

Better, Faster, Stronger







In a head-to-head comparison with the industry's more popular languages, Go rivals its seasoned predecessors in both speed and efficiency.

```go
package main

import "crypto/sha1"
import "fmt"

func main() {
    s := "sha1 this string"
    h := sha1.New()

    h.Write([]bytes(s))
    bs := h.Sum(nil)

    fmt.Println(s)
    fmt.Printf("%x\n", bs)
}
```

```go
package main

import (
    "fmt"
    "log"
    "net/http"
)

func main() {
    http.HandleFunc("/hello", handleHello)
    fmt.Println("serving on http://localhost:7777/hello")
    log.Fatal(http.ListenAndServe("localhost:7777", nil))
}

func handleHello(w http.ResponseWriter, req *http.Request) {
    log.Println("serving", req.URL)
    fmt.Fprintln(w, "Hello, 世界!")
}
```

Not only does Go have an edge in performance, but it is also convenient to use due its readability and writability. Why write an epic if all you need is a haiku?