# *APPENDIX*

**MAIN CODE**

```
package main
import (
	modul "samperin/modul"
)
func main() {
	modul.ApiList()
}
```

**API LIST**

```
package modul
import (
	"log"
	"net/http"

	"github.com/gorilla/mux"
)
func ApiList() {
	backup := mux.NewRouter()
	router := mux.NewRouter()
	router.Handle("/api/manager/{sbuid}/getsaleslist/{stat}",
JWTAuth(http.HandlerFunc(GetSalesStatusList))).Methods("GET")
	router.HandleFunc("/api/user/login",
UserLogin).Methods("POST")
	router.Handle("/",
JWTAuth(http.HandlerFunc(homePage))).Methods("GET")
	log.Fatal(http.ListenAndServe(":9999", router))
}
```

**JWT AUTHENTICATION**

```
package modul
import (
	"encoding/json"
	"log"
	"net/http"
	"structs"
	"time"

	"jwt-go-master"
)
var mySigningKey = []byte("supersecretkey")
func JWTAuth(next http.Handler) http.Handler {
	var (
		ip      string
		port    string
		forward string
	)
	return   http.HandlerFunc(func(w   http.ResponseWriter,   r
*http.Request) {
```

```
            ip, port, forward = GetIp(w, r)
            tokenString := r.Header.Get("key")
            if tokenString == "" {
                  respondWithError(w,     http.StatusUnauthorized,
"Unauthorized")
            } else {
                  result, err := jwt.Parse(tokenString, func(token
*jwt.Token) (interface{}, error) {
                        return mySigningKey, nil
                  })
                  if err == nil && result.Valid {
                        next.ServeHTTP(w, r)
                  } else {
                        respondWithError(w,
http.StatusUnauthorized, "Unauthorized")
                        log.Println("Invalid,", ip, port, forward)
                  }
            }
      })
}
```

**USER LOGIN**

```
package modul
import (
      "bytes"
      "context"
      "crypto/md5"
      "encoding/json"
      "fmt"
      "log"
      "net/http"
      "strings"
      "structs"
      "time"
      "jwt-go-master"
      "github.com/elastic/go-elasticsearch"
)
func UserLogin(w http.ResponseWriter, r *http.Request) {
      var (
            user    structs.Account
            buf     bytes.Buffer
            buf2    bytes.Buffer
            finale  []map[string]interface{}
            counter int
            slot    map[string]interface{}
            out     map[string]interface{}
      )
      es, err := elasticsearch.NewDefaultClient()
      checkError(err)
      res, err := es.Info()
      checkError(err)
      if res.IsError() {
            log.Fatalf("Error: %s", res.String())
```

```
	}
	if err := json.NewDecoder(r.Body).Decode(&user); err != nil
{
		log.Fatalf("Error parsing the response body: %s", err)
	}
	passna := []byte(user.Password)
	enc := md5.Sum(passna)
	passfin := fmt.Sprintf("%x", enc)
	log.Println(passfin)
	token	:=	jwt.NewWithClaims(jwt.SigningMethodHS256,
jwt.MapClaims{
		"Username": user.Email,
		"Password": user.Password,
		"exp":	time.Now().Add(time.Hour * 168).Unix(),
	})
	tokenString, err2 := token.SignedString(mySigningKey)
	if err2 != nil {
		respondWithError(w,		http.StatusBadRequest,
err2.Error())
	}
	query := map[string]interface{}{
		"query": map[string]interface{}{
			"bool": map[string]interface{}{
				"must": []interface{}{
					map[string]interface{}{
						"match_phrase":
map[string]interface{}{
							"email": user.Email,
						},
					},
					map[string]interface{}{
						"match_phrase":
map[string]interface{}{
							"password": passfin,
						},
					},
				},
			},
		},
	}
	if err := json.NewEncoder(&buf).Encode(query); err != nil {
		log.Fatalf("Error encoding query: %s", err)
	}
	res, err = es.Search(
		es.Search.WithContext(context.Background()),
		es.Search.WithIndex("samperin_user"),
		es.Search.WithBody(&buf),
		es.Search.WithTrackTotalHits(true),
		es.Search.WithPretty(),
	)
	checkError(err)
	defer res.Body.Close()
	if res.IsError() {
		var e map[string]interface{}
```

```
            if err := json.NewDecoder(res.Body).Decode(&e); err !=
nil {
                  log.Fatalf("Error    parsing    the    response
body: %s", err)
            } else {
                  log.Fatalf("[%s] %s: %s",
                        res.Status(),

      e["error"].(map[string]interface{})["type"],

      e["error"].(map[string]interface{})["reason"],
                  )
            }
      }
      if  err  :=  json.NewDecoder(res.Body).Decode(&slot);  err  !=
nil {
            log.Fatalf("Error parsing the response body: %s", err)
      }
      if
int(slot["hits"].(map[string]interface{})["total"].(map[string]int
erface{})["value"].(float64)) == 0 {
            w.Write([]byte("No data"))
      }
      log.Printf(
            "[%s] %d hits; took: %dms",
            res.Status(),
      int(slot["hits"].(map[string]interface{})["total"].(map[stri
ng]interface{})["value"].(float64)),
            int(slot["took"].(float64)),
      )
      buf2.WriteString(string("["))
      for            _,            hit            :=            range
slot["hits"].(map[string]interface{})["hits"].([]interface{}) {
            s                 :=                 fmt.Sprintf("%v",
hit.(map[string]interface{})["_source"])
            i := hit.(map[string]interface{})["_id"]
            key := "ICON@+@2020@SBU@REGIONAL@MEDAN!!"
            log.Println(" ID =", i, s)

            empt := map[string]interface{}{}
            output, _ := json.Marshal(empt)
            json.Unmarshal([]byte(output), &out)
            datana := fmt.Sprintf("%v", hit)
            encryptres := encrypt([]byte(datana), key)
            datano := fmt.Sprintf("%x", encryptres)
            log.Println(encryptres)
            out["token"] = tokenString
            out["data"] = datano
            out["type"] = "login"
            tempo, _ := json.Marshal(out)
            buf2.WriteString(string(tempo))
            counter++

      }
```

```
	buf2.WriteString(string("]"))
	finales   :=   strings.Replace(buf2.String(),   "}",   "},",
counter-1)
	json.Unmarshal([]byte(finales), &finale)
	grouped := groupBy(finale, "type")
	respondWithJson(w, http.StatusOK, grouped)
	checkError(err)
	log.Println(strings.Repeat("=", 37))
}
```

**GET SALES STATUS LIST**

```
package modul
import (
	"bytes"
	"context"
	"encoding/json"
	"fmt"
	"log"
	leg "log4go"
	"net/http"
	"strconv"
	"strings"
	"github.com/elastic/go-elasticsearch"
	"github.com/gorilla/mux"
)
func GetSalesStatusList(w http.ResponseWriter, r *http.Request) {
	var (
		finale  []map[string]interface{}
		er      map[string]interface{}
		buf     bytes.Buffer
		bufstr  bytes.Buffer
		counter int
		sumna   float64
	)
	es, err := elasticsearch.NewDefaultClient()
	checkError(err)
	res, err := es.Info()
	checkError(err)
	params := mux.Vars(r)
	query := map[string]interface{}{
		"query": map[string]interface{}{
			"bool": map[string]interface{}{
				"must": []interface{}{
					map[string]interface{}{
						"match_phrase":
map[string]interface{}{
							"sbu_id":
params["sbuid"],
						},
					},
					map[string]interface{}{
						"match_phrase":
map[string]interface{}{
```

```
                                        "role": "1",
                                  },
                            },
                      },
                },
          },
          "sort": []interface{}{
                map[string]interface{}{
                      "email": map[string]interface{}{
                            "order": "asc",
                      },
                },
          },
          "size": 1000,
    }
    if err := json.NewEncoder(&buf).Encode(query); err != nil {
          log.Fatalf("Error encoding query: %s", err)
          leg.LOGGER("Err").Debug("Error encoding query: %s",
err)
    }
    res, err = es.Search(
          es.Search.WithContext(context.Background()),
          es.Search.WithIndex("samperin_user"),
          es.Search.WithBody(&buf),
          es.Search.WithTrackTotalHits(true),
          es.Search.WithPretty(),
    )
    checkError(err)
    defer res.Body.Close()
    if res.IsError() {
          var e map[string]interface{}
          if err := json.NewDecoder(res.Body).Decode(&e); err !=
nil {
                log.Fatalf("Error parsing the response
body: %s", err)
                leg.LOGGER("Err").Debug("Error parsing the
response body: %s", err)
          } else {
                log.Fatalf("[%s] %s: %s",
                      res.Status(),

    e["error"].(map[string]interface{})["type"],

    e["error"].(map[string]interface{})["reason"],
                )
                leg.LOGGER("Res").Debug("[%s] %s: %s",
                      res.Status(),

    e["error"].(map[string]interface{})["type"],

    e["error"].(map[string]interface{})["reason"],
                )
          }
    }
```

```
      if err := json.NewDecoder(res.Body).Decode(&er); err != nil
{
            log.Fatalf("Error parsing the response body: %s", err)
            leg.LOGGER("Err").Debug("Error parsing the response
body: %s", err)
      }
      log.Printf(
            "[%s] %d hits; took: %dms",
            res.Status(),

      int(er["hits"].(map[string]interface{})["total"].(map[string
]interface{})["value"].(float64)),
            int(er["took"].(float64)),
      )
      leg.LOGGER("Res").Debug(
            "[%s] %d hits; took: %dms",
            res.Status(),

      int(er["hits"].(map[string]interface{})["total"].(map[string
]interface{})["value"].(float64)),
            int(er["took"].(float64)),
      )
      bufstr.WriteString(string("["))
      for                _,                hit                :=                range
er["hits"].(map[string]interface{})["hits"].([]interface{}) {
            emailna                :=                fmt.Sprintf("%v",
hit.(map[string]interface{})["_source"].(map[string]interface{})["
email"])
            fullnamena                :=                fmt.Sprintf("%v",
hit.(map[string]interface{})["_source"].(map[string]interface{})["
fullname"])
            sumna = 0
            query = map[string]interface{}{
                "query": map[string]interface{}{
                    "bool": map[string]interface{}{
                        "must": []interface{}{
                            map[string]interface{}{
                                "match_phrase":
map[string]interface{}{
                                    "status":
params["stat"],
                                },
                            },
                            map[string]interface{}{
                                "match_phrase":
map[string]interface{}{
                                    "email": emailna,
                                },
                            },
                        },
                    },
                },
            }
```

```
            if err := json.NewEncoder(&buf).Encode(query); err !=
nil {
                  log.Fatalf("Error encoding query: %s", err)
                  leg.LOGGER("Err").Debug("Error          encoding
query: %s", err)
            }
            res, err = es.Search(
                  es.Search.WithContext(context.Background()),
                  es.Search.WithIndex("samperin_status_detail"),
                  es.Search.WithBody(&buf),
                  es.Search.WithTrackTotalHits(true),
                  es.Search.WithPretty(),
            )
            checkError(err)
            defer res.Body.Close()
            if res.IsError() {
                  var e map[string]interface{}
                  if err := json.NewDecoder(res.Body).Decode(&e);
err != nil {
                        log.Fatalf("Error parsing the response
body: %s", err)
                        leg.LOGGER("Err").Debug("Error parsing the
response body: %s", err)
                  } else {
                        log.Fatalf("[%s] %s: %s",
                              res.Status(),

      e["error"].(map[string]interface{})["type"],

      e["error"].(map[string]interface{})["reason"],
                        )
                        leg.LOGGER("Res").Debug("[%s] %s: %s",
                              res.Status(),

      e["error"].(map[string]interface{})["type"],

      e["error"].(map[string]interface{})["reason"],
                        )
                  }
            }
            if   err   :=   json.NewDecoder(res.Body).Decode(&er);
err != nil {
                  log.Fatalf("Error    parsing    the    response
body: %s", err)
                  leg.LOGGER("Err").Debug("Error    parsing    the
response body: %s", err)
            }
            log.Printf(
                  "[%s] %d hits; took: %dms",
                  res.Status(),

      int(er["hits"].(map[string]interface{})["total"].(map[string
]interface{})["value"].(float64)),
                  int(er["took"].(float64)),
```

```
            )
            leg.LOGGER("Res").Debug(
                  "[%s] %d hits; took: %dms",
                  res.Status(),

      int(er["hits"].(map[string]interface{})["total"].(map[string
]interface{})["value"].(float64)),
                  int(er["took"].(float64)),
            )
            for          _,          hit2          :=          range
er["hits"].(map[string]interface{})["hits"].([]interface{}) {
                  tot              :=              fmt.Sprintf("%v",
hit2.(map[string]interface{})["_source"].(map[string]interface{})[
"total"])
                  totf, _ := strconv.ParseFloat(tot, 64)
                  sumna += totf
            }
            if sumna == 0 {
                  continue
            } else {
                  // key := "ICON@+@2020@SBU@REGIONAL@MEDAN!!"
                  // datana := fmt.Sprintf("%v", sumna)
                  // encryptres := encrypt([]byte(datana), key)
                  // datano := fmt.Sprintf("%x", encryptres)
                  // log.Println(encryptres)
                  out := map[string]interface{}{}
                  empt := map[string]interface{}{}
                  output, _ := json.Marshal(empt)
                  json.Unmarshal([]byte(output), &out)
                  out["email"] = emailna
                  out["fullname"] = fullnamena
                  out["sum"] = sumna
                  out["type"] = "sales"
                  tempo, _ := json.Marshal(out)
                  bufstr.WriteString(string(tempo))
                  counter++
            }
      }
      bufstr.WriteString("]")
      finales   :=   strings.Replace(bufstr.String(),   "}",   "},",
counter-1)
      json.Unmarshal([]byte(finales), &finale)
      grouped := groupBy(finale, "type")
      respondWithJson(w, http.StatusOK, grouped)
      log.Println(strings.Repeat("=", 37))
      leg.LOGGER("Res").Debug(strings.Repeat("=", 37))
}
```

**RESPOND CODE**

```
package modul
import (
      "crypto/aes"
      "crypto/cipher"
```

```
	"crypto/md5"
	"crypto/rand"
	"encoding/hex"
	"encoding/json"
	"fmt"
	"log"
	"net"
	"net/http"
	"io"
	leg "log4go"
)
func GetIp(w http.ResponseWriter, r *http.Request) (string,
string, string) {
	ip, port, _ := net.SplitHostPort(r.RemoteAddr)

	userIP := net.ParseIP(ip)
	if userIP == nil {
		log.Println("err")
		return "nil", "nil", "nil"
	}
	forward := r.Header.Get("X-Forwarded-For")

	return ip, port, forward
}
func CheckVer(w http.ResponseWriter, r *http.Request) {
	w.Write([]byte("version 1.0.0"))
}
func checkError(err error) {
	if err != nil {
		log.Fatal(err)
		leg.LOGGER("Err").Debug(err)
	}
}
func respondWithError(w http.ResponseWriter, code int, msg string)
{
	respondWithJson(w, code, map[string]string{"error": msg})
	leg.LOGGER("Err").Debug("error: ", msg)
}
func respondWithJson(w http.ResponseWriter, code int, payload
interface{}) {
	response, _ := json.Marshal(payload)
	w.Header().Set("Content-Type", "application/json")
	w.WriteHeader(code)
	w.Write(response)
	prettyJSON, _ := json.MarshalIndent(payload, "", "    ")
	leg.LOGGER("Res").Debug(string(prettyJSON))
}
func homePage(w http.ResponseWriter, r *http.Request) {
	validToken := "Hello There"
	fmt.Fprintln(w, validToken)
}
func groupBy(maps []map[string]interface{}, key string)
map[string][]map[string]interface{} {
	groups := make(map[string][]map[string]interface{})
```

```
	for _, m := range maps {
		k := m[key].(string) // XXX: will panic if m[key] is
not a string.
		groups[k] = append(groups[k], m)
	}
	return groups
}
func EmptPlan(dayset string, email string) string {
	var out map[string]interface{}
	empt := map[string]interface{}{}
	op, _ := json.Marshal(empt)
	json.Unmarshal([]byte(op), &out)
	out["date"] = dayset
	out["activity"] = "nil"
	out["detail"] = "nil"
	out["client"] = "nil"
	out["email"] = email
	out["type"] = "task"
	out["id"] = "nil"
	response, _ := json.Marshal(out)
	var finstring = string(response)
	return finstring
}
func trimQuotes(s string) string {
	if len(s) >= 2 {
		if s[0] == '"' && s[len(s)-1] == '"' {
			return s[1 : len(s)-1]
		}
	}
	return s
}
func createHash(key string) string {
	hasher := md5.New()
	hasher.Write([]byte(key))
	return hex.EncodeToString(hasher.Sum(nil))
}
func encrypt(data []byte, passphrase string) []byte {
	block, _ := aes.NewCipher([]byte(createHash(passphrase)))
	gcm, err := cipher.NewGCM(block)
	if err != nil {
		panic(err.Error())
	}
	nonce := make([]byte, gcm.NonceSize())
	if _, err = io.ReadFull(rand.Reader, nonce); err != nil {
		panic(err.Error())
	}
	ciphertext := gcm.Seal(nonce, nonce, data, nil)
	return ciphertext
}
```

PLAGIARISM CHECK.ORG

**0.68%** PLAGIARISM APPROXIMATELY

# Report #11013022

IntroductionBackgroundComputer security is a very important thing. Apart from the sophistication of the features contained in the computer, the security of the system is also very influential in the use of several features and applications that exist on the computer. This makes security a necessity for every application on the computer. There are many sides that must be considered in securing an application, one of which is on the communication path or also called the API. API (Application Programming Interface) is a set of advance tools that developers can use to integrate two parts of an application. The purpose of creating this API itself is to accelerate the creation and development of an application because it allows developers to access the features of the platform. The API form itself can be function, method, or URL endpoint. [1] Web service / API is divided into two types, Simple Object Access Protocol (SOAP) an d Representational State Transfer (REST). REST represents that each unique URL is an object; the object of the content can be obtained using HTTP GET, and can be modify by using the POST, PUT, or DELETE method. For the security, because REST using HTTP or HTTPS the REST administrator (firewall) can see the intent of each message by analyzing the HTTP commands used in the request. The authentication and authorization from REST itself assumes that



L