BAB III
PELAKSANAAN KERJA MAGANG

3.1. Kedudukan dan Koordinasi

Pada praktek kerja magang kali ini, penulis ditempatkan pada tim didalam divisi *product* yang bertanggung jawab atas produk PrivyID yaitu *Privacy Middleware*. Tim *Privacy Middleware* ini beranggotakan 1 orang *Product Manager*, 1 orang *Product Tech Lead Engineer*, 1 orang *Senior Backend Engineer*, 2 orang *QA Engineer* dan penulis sebagai *Backend Engineer Intern*.

Koordinasi dilakukan setiap hari pada jam 2 siang, dimana setiap hari Selasa dilakukan *Sprint Meeting* untuk memberikan tugas baru kepada masing-masing anggota dan pada hari lainnya dilakukan *Daily Standup Meeting* untuk memberikan informasi seputar apa yang telah dikerjakan kemarin, apakah ada kendala yang dihadapi dan apa rencana yang akan dilakukan pada hari ini dari masing-masing anggota.

3.2. Tugas yang Dilakukan

Tugas yang dilakukan selama kerja magang yaitu:

1. *Onboarding* dan mengurus dokumen administrasi untuk mendapatkan email dan akses kedalam *GitLab.*
2. Eksplorasi *repository GitLab* dari tim *on-premise* dan mempelajari tentang *library* gorilla/mux.
3. Membuat aplikasi sederhana yang mengimplementasikan gorilla/mux dan mempelajari *business process* dari tim *on-premise*.
4. Mempelajari tentang *logging* didalam *backend*, *library* .env dan API testing mengguanakn *postman*.
5. Pemasangan sistem *log* pada beberapa *Application Programming Interface* (API) yang belum terpasang.
6. Mempelajari *GitFlow*.
7. Perbaikan *bug* yang terdapat pada salah satu API yang dilaporkan oleh QA.
8. Memepalajari tentang dokuemntasi *Swagger*.
9. Pemasangan dokumentasi *swagger* pada setiap API yang terdapat pada produk *Privacy Middleware*.
10. Pembuatan *Centralized Log System* yang merupakan sentralisasi dari semua *log* yang dilakukan dalam produk *Privacy Middleware*.

3.3. Uraian Pelaksanaan Kerja Magang

3.3.1. *OnBoarding* dan mengurus dokumen

Pada tahap ini, dilakukan *OnBoarding* dari *product tech lead engineer*, dimana dalam *OnBoarding* diperkenalkan tim dari on-premise dan produk yang nantinya akan dikerjakan serta bisnis proses dari produk *on-premise*. Selain *OnBoarding*, dilakukan pengisian dokumen internal dari perusahaan untuk mendapatkan akses dari *repository* tim *on-premise* serta E-mail perusahaan guna mendapatkan informasi seputar internal perusahaan.

3.3.2. Eksplorasi *repository* dan mempelajari *library* gorilla/mux

Pada tahap ini, ketika sudah mendapatkan akses kedalam *repository GitLab* milik tim *on-premise*, dilakukan eksplorasi kedalam *repository* dimana eksplorasi yang dilakukan berupa pengenalan gaya pemrograman dari tim *on-premise* serta bagaimana mereka membangun *code-base* mereka. Selain melakukan eksplorasi, *product tech lead engineer* juga meminta untuk mempelajari *library* gorilla/mux secara khusus agar lebih memahami cara pembangunan aplikasi *backend*.

3.3.3. Pembuatan aplikasi sederhana dari *library* gorilla/mux

Pada tahap ini, *product tech lead engineer* meminta untuk membuat aplikasi sederhana yang mengimplementasikan *library* yang sudah dipelajari yaitu gorilla/mux untuk mengetahui apakah sudah memahami *library* tersebut atau belum, jika belum maka akan diarahkan dan diajarkan bagaimana cara penggunaannya. Selain diajarkan juga diberikan *feedback* bagaimana *best practice* dalam penggunaan gorilla/mux.

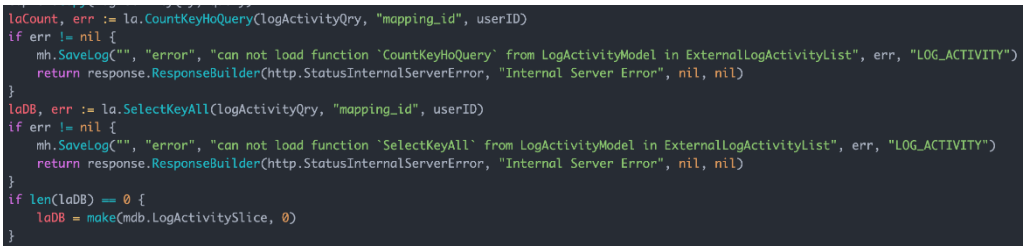
3.3.4. Mempelajari sistem *logging*, *library* godot .env dan *postman*

Sistem *logging* menjadi salah satu bagian yang krusial untuk melakukan *troubleshooting* kedalam aplikasi *backend* jika terdapat masalah atau untuk kepentingan analitik. Maka dari itu, *product tech lead engineer* meminta untuk mempelajari sistem *logging* yang terdapat pada *repository on-premise* juga cara penggunaannya. Selain itu juga diminta untuk mempelajari *library* godot .env serta aplikasi API testing yaitu postman.

3.3.5. Pemasangan Sistem *Log*

Terdapat banyak *endpoint* dari API yang terdapat dialam produk *Privacy Middleware* dan terkadang orang yang membuat *endpoint* tersebut lupa untuk menambahkan kodingan untuk menyimpan *log* bahwa API tersebut telah diakses. Maka sebagai langkah awal untuk menguji apakah mahasiswa sudah memahami

repository dari produk *Privacy Middleware* adalah menambahkan kodingan tersebut kedalam *endpoint* yang belum terpasang *log*. Berikut contoh *line* yang ditambahkan pada gambar 3.1:



Gambar 3.1 (Memasang sistem log dalam endpoint)

3.3.6. Mempelajari *GitFlow*

Pada tahap ini, karena *product tech lead engineer* melihat bahwa pengetahuan tentang penggunaan version control seperti *git* masih minim, maka *product tech lead engineer* meminta untuk mempelajari alur dari penggunaan dari *version control git* yaitu gitflow. Dalam *gitflow* terdapat beberapa perintah dasar atu istilah yang harus diketahui seperti *commit, push, add, stash, branch dan pull* yang masing-masing memiliki karakteristik tersendiri bagaimana kita akan berinteraksi dengan repository, karena tidak ada user interface yang digunakan untuk melakukan interaksi langsung dan semua perintah tersebut akan dimasukkan melalui terminal.

3.3.7. Perbaikan Bug

Pada tahap ini, mahasiswa diberikan tugas untuk melakukan *debugging* atas *error* yang ditemukan oleh *QA Engineer*. Waktu yang diberikan adalah 1 minggu, namun dalam kurun waktu tersebut tidak ditemukan masalah yang terdapat didalam API, maka dari itu saat *sprint meeting* minggu berikutnya memberikan *update* bahwa tidak ditemukan masalah dari *endpoint* API. Setelah beberapa hari kemudian *Tech Lead* memberikan *update* dari isu tersebut bahwa ternyata masalah bukan terdapat pada API melainkan data di database yang memiliki *time stamp* yang berbeda.

3.3.8. Mempelajari dokumentasi *Swagger*

Pada tahap ini, *product tech lead engineer* meminta untuk mempelajari teknologi bernama *swagger* dimana *swagger* digunakan untuk melakukan dokumentasi API secara interaktif tanpa harus mengimplementasikan API yang telah dibuat kedalam aplikasi secara langsung. *Swagger* dapat menampilkan *request* dan *response* secara interaktif berdasarkan *request input* yang dimasukkan dan mengeluarkan *response* berdasarkan *request* yang dimasukkan.

3.3.9. Pemasangan Dokumentasi *Swagger*

Jumlah API yang banyak mengakibatkan setiap API yang tidak terdokumentasi dengan baik. Oleh karena itu mahasiswa diminta untuk melakukan dokumentasi atas semua API yang ada bersama dengan QA untuk mengetahui *request* dan *respond* yang diharapkan dari masing-masing API. Berikut salah satu contoh API yang didokumentasikan menggunakan *swagger*:

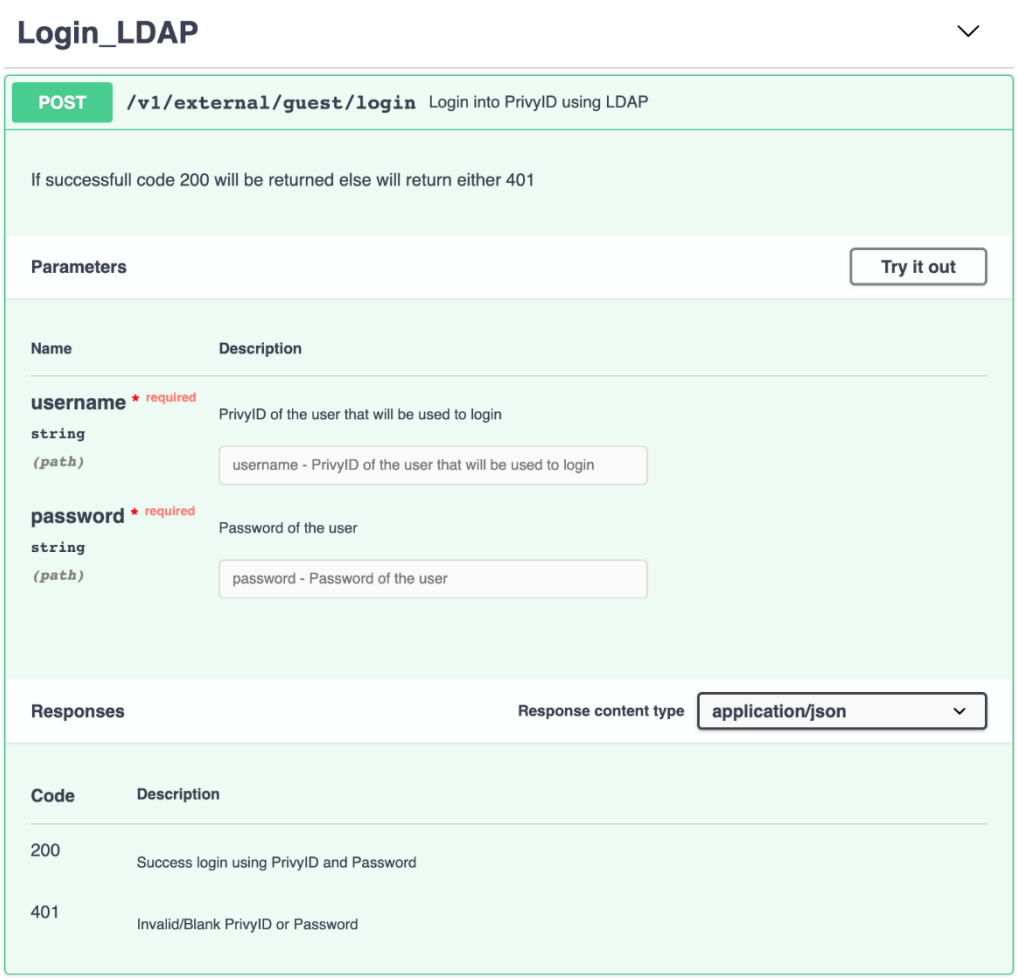
```
// swagger:operation POST /v1/external/guest/login Login_LDAP guestLogin
// ---
// summary: Login into PrivyID using LDAP
// description: If successfull code 200 will be returned else will return either 401
// parameters:
// - name: username
//   in: path
//   description: PrivyID of the user that will be used to login
//   type: string
//   required: true
// - name: password
//   in: path
//   description: Password of the user
//   type: string
//   required: true
// responses:
//   "200":
//     description: Success login using PrivyID and Password
//   "401":
//     description: Invalid/Blank PrivyID or Password
```

Gambar 3.2 (Contoh pemasangan)

Gambar 3.2 adalah merupakan *comment* yang dibuat didalam *function login*, dari *comment* tersebut *swagger* akan mengkonversinya menjadi bentuk JSON seperti berikut:

Gambar 3.3 (Hasil perubahan comment menjadi JSON)

Gambar 3.3 merupakan hasil konversi dari *comment* menjadi format JSON. Setelah diubah formatnya menjadi JSON, maka *swagger* akan memvisualisasikannya menjadi dokumentasi yang lebih mudah dipahami oleh bagian *front-end*, berikut contohnya:



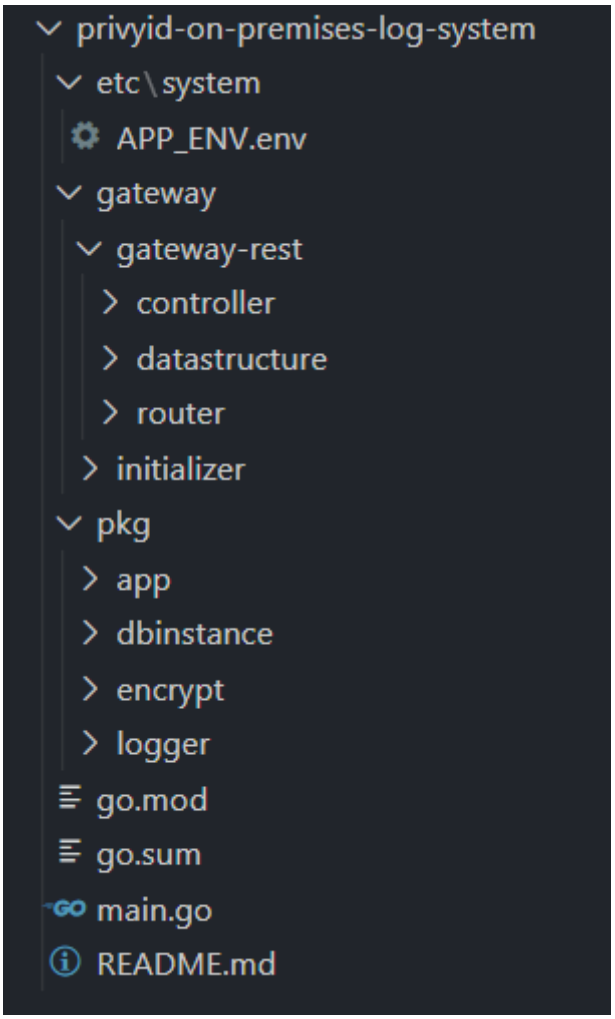
Gambar 3.4 (Bentuk visualisasi swagger)

Gambar 3.4 merupakan bentuk akhir dari dokumentasi menggunakan *swagger* yaitu *user interface* interaktif untuk dibaca dan digunakan oleh bagian *front-end*. Waktu yang dibutuhkan untuk mendokumentasikan semua API yang ada adalah 3 minggu.

3.3.10. Pembuatan *Centralized Log System*

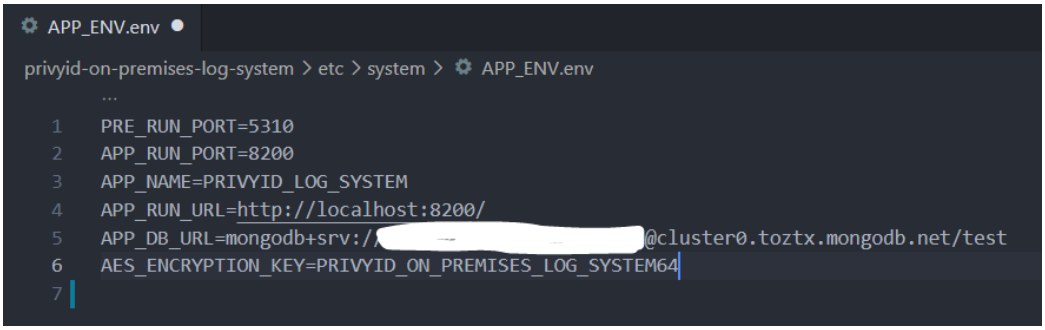
Centralized log system adalah sentralisasi dari *logging* yang ada pada produk *Privacy Middleware*. Karena *centralized log system* adalah sistem yang dibuat dari awal, maka tahap *development* akan dimulai dari awal.

Pada tahap awal, dilakukan pembuatan boilerplate. Boilerplate bertujuan untuk mempersiapkan hal-hal yang dibutuhkankan sebelum masuk kedalam tahap development, seperti membuat struktur folder sesuai *Model-View-Controller*, mempersiapkan *environment variable*, membuat *folder tree* agar dapat diakses dari folder lainnya, dan membuat handler untuk *Cross-Origin Resource Sharing* (CORS).



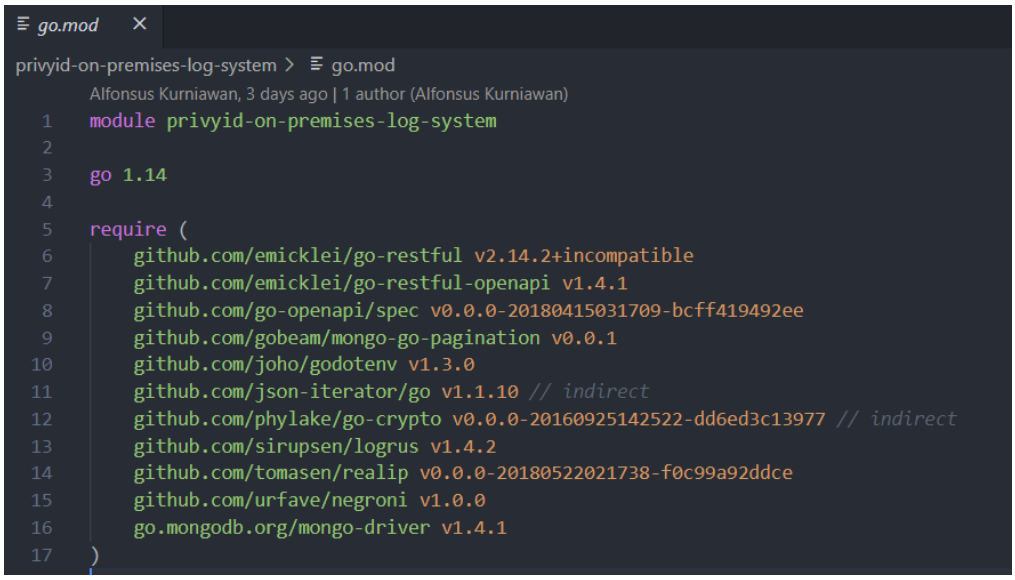
Gambar 3.5 (Folder Tree)

Gambar 3.5 merupakan struktur *folder* yang telah dibuat mengikuti format MVC yang dianjurkan oleh *senior* dan *lead* di dalam tim.



Gambar 3.6 (Environment Variable)

Gambar 3.6 merupakan *environment variable* yang disimpan ke dalam *file* .env untuk di *load* kedalam program.



Gambar 3.7 (Go Module)

Gambar 3.7 merupakan *module* dari *GoLang*, yang nantinya jika ingin mengakses *function* yang berada didalam *folder* yang berbeda, pengaksesan akan diawali dengan *module*.

```
cors := restful.CrossOriginResourceSharing{
    AllowedHeaders: []string{"Content-Type", "Accept"},
    AllowedMethods: []string{"GET", "POST", "PUT", "DELETE"},
    CookiesAllowed: false,
    Container:      restful.DefaultContainer}
restful.DefaultContainer.Filter(cors.Filter)
```

Gambar 3.8 (Cors Handler)

Gambar 3.8 merupakan *handler* untuk CORS, agar data-data yang dikirimkan melalui *RestAPI* dapat diakses oleh pihak *front-end*.

Ketika semua telah disiapkan mulai dari *boilerplate*, struktur *folder*, *module* dan *environment variabel* telah dibuat, maka tahap *development* dapat dimulai.

Data dari *log* bisa dikatakan sebagai data yang bersifat rahasia, maka dari itu perlu adanya *encryption* untuk merahasiakan data tersebut.

```go
package encrypt

import (
    "bytes"
    "crypto/aes"
    "crypto/cipher"
    "encoding/hex"
)

// Encrypt is used to encrypt the Fileprefix
func Encrypt(key []byte, logdata string, iv []byte, blockSize int) string {
    blogdata := PKCS5Padding([]byte(logdata), blockSize, len(logdata))
    block, _ := aes.NewCipher(key)
    ciphertext := make([]byte, len(blogdata))
    mode := cipher.NewCBCEncrypter(block, iv)
    mode.CryptBlocks(ciphertext, blogdata)
    return hex.EncodeToString(ciphertext)
}

func PKCS5Padding(ciphertext []byte, blockSize int, after int) []byte {
    padding := (blockSize - len(ciphertext)%blockSize)
    padtext := bytes.Repeat([]byte{byte(padding)}, padding)
    return append(ciphertext, padtext...)
}
```

Gambar 3.9 (Enkripsi)

Gambar 3.9 merupakan *function* untuk membuat *key* baru untuk enkripsi. *Key* tersebut di buat menggunakan konsep enkripsi *AES256*.

Setelah mempersiapkan *environment variable*, kita perlu me-*load* file yang menyimpan *environment* tersebut agar dapat digunakan.

```
//load environment variables from specific location
err := app.LoadEnv([]string{
    "APP_ENV",
}, "/etc/system")

if err != nil {
    log.Fatal("Unable to Set app Environment. err: ", err)
}
```

Gambar 3.10 (Load environment variable)

Gambar 3.10 merupakan *block code* yang berada pada *function main*, agar *environment variable* yang telah dibuat dapat diakses dari manapun di dalam program, jika *environment variable* gagal untuk di-*load* ke dalam program maka program akan berhenti dan mengeluarkan pesan *error*.

```go
package app

import (
    "os"
    "path/filepath"
    "strings"

    "github.com/joho/godotenv"
)

func LoadEnv(envVarKeys []string, fileBasePath string) error {
    for _, k := range envVarKeys {
        str := os.Getenv(k)

        if str == "" {
            continue
        }

        r := strings.NewReader(str)
        envMap, err := godotenv.Parse(r)
        if err != nil {
            return err
        }

        for ik, iv := range envMap {
            if _, exists := os.LookupEnv(ik); exists {
                os.Setenv(ik, iv)
            }
        }
    }
```

Gambar 3.11 (Isi dari function LoadEnv)

Gambar 3.11 merupakan isi dari function LoadEnv yang digunakan agar *variabel environment* dapat diakses didalam program, *library* yang digunakan untuk me-*load environment variable* adalah *godot* .env yang dibuat oleh Joho.

```
apiPort := os.Getenv("APP_RUN_PORT")
var router *restful.WebService
if apiPort == "" {
    fmt.Println("Use Pre Configuration")
    apiPort = os.Getenv("PRE_RUN_PORT")
    router = r.PreRunningConfig()
} else {
    fmt.Println("Running in Operation")
    router = r.RunningRoute()
}
```

Gambar 3.12 (Inisiasi Router)

Gambar 3.12 merupakan block yang digunakan untuk melakukan load semua endpoint berdasarkan environment variable APP_RUN_PORT. Jika sedang menggunakan port untuk pre-configuration maka akan menjalankan PreRunningConfig, namun jika akan menggunakan Operation akan menjalankan RunningRoute.

```go
package router

import (
    "net/http"
    c "privyid-on-premises-log-system/gateway/gateway-rest/controller"
    ds "privyid-on-premises-log-system/gateway/gateway-rest/datastructure"

    restful "github.com/emicklei/go-restful"
    restfulspec "github.com/emicklei/go-restful-openapi"
)

//PreRunningConfig is routing for first time setup or configuration route
func PreRunningConfig() *restful.WebService {
    service := new(restful.WebService)

    return service
}

//RunningRoute routing for apps running route
func RunningRoute() *restful.WebService {
    service := new(restful.WebService)
    service.
        Path("/log").
        Consumes(restful.MIME_XML, restful.MIME_JSON).
        Produces(restful.MIME_XML, restful.MIME_JSON)

    tags := []string{"Log API"}
```

Gambar 3.13 (Setup router)

Gambar 3.13 merupakan *block* awal untuk melakukan *assign* API *route* kedalam *router*. Untuk melakukan *routing* menggunakan *library go-restful* yang dibuat oleh Emicklei.

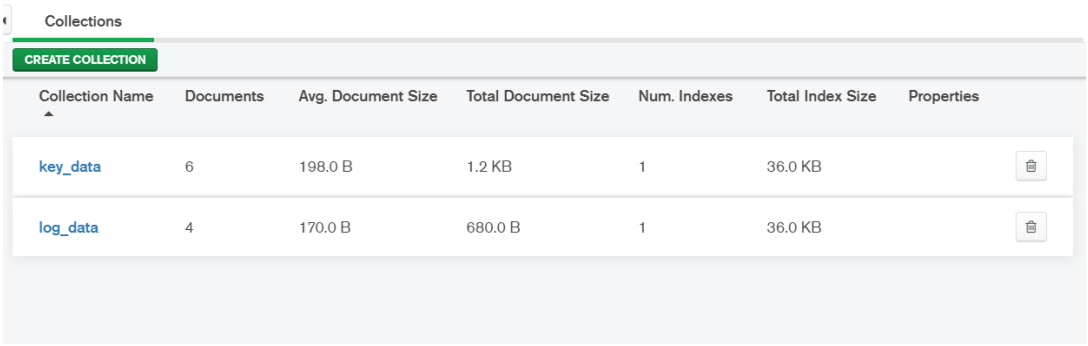
```go
service.Route(service.GET("/getlogdetail/{logID}").
    To(c.GetDetailLog).
    Metadata(restfulspec.KeyOpenAPITags, tags).
    Doc("Get detail of the log data").
    Param(service.PathParameter("logID", "identifier of log data").DataType("string")).
    Returns(401, "Invalid logID", nil).
    Returns(402, "Blank logID", nil))
```

Gambar 3.14 (Contoh registrasi route dan controller)

Gambar 3.14 merupakan salah satu contoh dari *block* kodingan cara melakukan *assign controller* kedalam *router*. Dengan

menggunakan *library go-restful-openapi*, maka dokumentasi API dapat dilakukan bersamaan saat melakukan *assign controller* ke dalam *router*. Ketika alamat "*/getlogdetail/{logID}*" diakses, maka *controller* yang akan dijalankan adalah "*GetDetailLog*".

Database yang digunakan adalah *MongoDB*, maka dari itu tidak perlu melakukan *setup* untuk membuat relasi antar *table* karena *MongoDB* merupakan *Document Oriented NoSQL Database*. Maka dari itu, tidak ada normalisasi kolom pada tahap ini, tapi langsung mengelompokkan tipe data apa yang akan masuk ke dalam *collection*. Untuk mengakses *database* tersebut, diperlukan *driver* dari *MongoDB* bernama *mongo-driver*, untuk membuka koneksi ke *cloud database* dari *MongoDB* dan mengakses *collection* yang dituju.



Gambar 3.15 (List collection dalam database MongoDB)

Gambar 3.15 adalah *list collection* yang ada pada *database Centralized Log System*. *Collection* tersebut berisikan tidak menyimpan data layaknya *database* relasional, tetapi menggunakan *document based*.

Gambar 3.16 (Contoh penyimpanan dokumen didalam collection)

Gambar 3.16 adalah contoh penyimpanan data yang ada di dalam *collection MongoDB*, setiap isi dokumen bisa berbeda-beda dan tidak ada *value* khusus yang akan selalu ada kecuali terdapat *value* yang disimpan kedalam *index* sebagai *constraint*.

```go
package mongodb

import (
    "context"
    "log"
    "os"

    "go.mongodb.org/mongo-driver/mongo"
    "go.mongodb.org/mongo-driver/mongo/options"
)

// StartMongoDB used to get connection to the database
func StartMongoDB() *mongo.Client {

    mongoCon := os.Getenv("APP_DB_URL")
    ctx := context.TODO()
    clientOptions := options.Client().ApplyURI(mongoCon)
    client, err := mongo.Connect(ctx, clientOptions)

    if err != nil {
        log.Fatalln(err)
    }

    err = client.Ping(ctx, nil)
    if err != nil {
        log.Fatalln(err)
    }

    log.Println("Connection openned to MongoDB")
    return client
}
```

Gambar 3.17 (Membuka koneksi ke MongoDB)

Gambar 3.17 merupakan *block code* yang digunakan untuk membuka akses kedalam database *MongoDB*. Namun belum bisa mengakses *collection* yang diinginkan.

```go
package mongodb

import (
    "go.mongodb.org/mongo-driver/mongo"
)

//GetCollection is used to get the collection in the database
func GetLogCollection() *mongo.Collection {
    client := StartMongoDB()
    return client.Database("privyid_log_system").Collection("log_data")
}

func GetKeyCollection() *mongo.Collection {
    client := StartMongoDB()
    return client.Database("privyid_log_system").Collection("key_data")
}
```

Gambar 3.18 (Mengakses collection didalam database)

Gambar 3.18 merupakan *block code* yang digunakan untuk mengakses *collection* yang ingin dituju dengan menggunakan client yang telah dibuka sebelumnya didalam *function* "*StartMongoDB()*".

Data model adalah bentuk *struct* yang dibuat untuk menerima *request* dari *body* atau ketika akan melakukan penyimpanan ke dalam *database MongoDB.*

```
type LogData struct {
    Code       int    `json:"code" bson:"code"`
    RequestID  string `json:"requestid" bson:"requestid"`
    Action     string `json:"action" bson:"action"`
    Payload    string `json:"payload" bson:"payload"`
    Fileprefix string `json:"fileprefix" bson:"fileprefix"`
    Error      string `json:"error,omitempty" bson:"error,omitempty"`
}
```

Gambar 3.19 (Contoh model data)

Gambar 3.19 merupakan salah satu contoh data model yang digunakan untuk menerima *request* yang kemudian disimpan ke dalam *MongoDB* dalam format *BSON*.

Controller adalah *function* yang dijalankan ketika salah satu dari *route* yang telah di *assign* di dalam *router* diakses. Dalam *Centralized Log System* terdapat 10 *endpoints*.

```go
package controller

import (
    "log"
    "net/http"
    str "privyid-on-premises-log-system/gateway/gateway-rest/datastructure"

    restful "github.com/emicklei/go-restful"
)

//Ping is a function used to test connection
func Ping(request *restful.Request, response *restful.Response) {
    // Build response to be parsed back
    m := str.Response{Code: 200, Message: "You are connected"}

    // Parse the response in json format
    response.WriteHeaderAndJson(http.StatusOK, m, restful.MIME_JSON)

    // Write the log to server
    log.Print("Helloww you are using the Ping API")

}
```

Gambar 3.20 (Controller ping)

Gambar 3.20 adalah *Ping controller* yang berfungsi untuk melakukan uji coba akses API yang tidak memerlukan *request* apapun

```go
package controller

import (
    "context"
    "encoding/base64"
    "log"
    "net/http"
    ds "privyid-on-premises-log-system/gateway/gateway-rest/datastructure"
    con "privyid-on-premises-log-system/pkg/dbinstance"
    encdec "privyid-on-premises-log-system/pkg/encrypt"
    "time"

    restful "github.com/emicklei/go-restful"
)

// CreateKey is the fuction called for routing /createkey
func CreateKey(request *restful.Request, response *restful.Response) {
    key := encdec.GenerateKey()
    keyString := string(key)
    keyBase64 := base64.StdEncoding.EncodeToString(key)
    currentTime, _ := time.Parse("2006-01-02 15:04:05", time.Now().Format("2006-01-02 15:04:05"))

    keyRequest := new(ds.KeyRequest)
    err := request.ReadEntity(&keyRequest)
    if err != nil {
        log.Println("Error request data format")
        resp := ds.Response{Code: 400, Message: "The data cannot be read properly (Wrong data type)"}
        response.WriteHeaderAndJson(http.StatusInternalServerError, resp, restful.MIME_JSON)
        return
    }

    keyData := new(ds.KeyStruct)
    keyData.Name = keyRequest.Name
    keyData.KeyStatus = keyRequest.Status
    keyData.ClientKey = keyString
```

Gambar 3.21 (Create key controller (1))

```go
    keyData.KeyStatus = keyRequest.Status
    keyData.ClientKey = keyString
    keyData.SecretKey = keyBase64
    keyData.UpdatedAt = currentTime
    keyData.CreatedAt = currentTime

    mongoCollection := con.GetKeyCollection()
    ctx, _ := context.WithTimeout(context.Background(), 15*time.Second)

    _, insertErr := mongoCollection.InsertOne(ctx, keyData)

    if insertErr != nil {
        log.Println("InsertOne ERROR:", insertErr)
        errResp := ds.Response{Code: 401, Message: "Failed creating new keyid"}
        response.WriteHeaderAndJson(http.StatusBadRequest, errResp, restful.MIME_JSON)
        return
    } else {
        m := ds.Response{Code: 200, Message: "Success creating new key id"}
        response.WriteHeaderAndJson(http.StatusOK, m, restful.MIME_JSON)
        log.Println("Success creating new key id")
    }
}
```

Gambar 3.22 (Create key controller (2))

Gambar 3.21 dan gambar 3.22 merupakan *controller* untuk membuat *key* baru untuk setiap *user* menggunakan prinsip AES256.

```go
package controller

import (
    "context"
    "log"
    "net/http"
    ds "privyid-on-premises-log-system/gateway/gateway-rest/datastructure"
    con "privyid-on-premises-log-system/pkg/dbinstance"
    "time"

    restful "github.com/emicklei/go-restful"
    "go.mongodb.org/mongo-driver/bson"
    "go.mongodb.org/mongo-driver/bson/primitive"
)

func DeleteKey(request *restful.Request, response *restful.Response) {
    keyID := request.PathParameter("keyID")
    if keyID == "" {
        log.Println("Blank keyID")
        resp := ds.Response{Code: 400, Message: "Blank keyID"}
        response.WriteHeaderAndJson(http.StatusInternalServerError, resp, restful.MIME_JSON)
        return
    }

    mongoCollection := con.GetKeyCollection()
    ctx, _ := context.WithTimeout(context.Background(), 15*time.Second)
    idToHex, _ := primitive.ObjectIDFromHex(keyID)
    filter := bson.D{{"_id", idToHex}}

    res, _ := mongoCollection.DeleteOne(ctx, filter)

    if res.DeletedCount == 0 {
        errResp := ds.Response{Code: 401, Message: "Invalid Key ID"}
        response.WriteHeaderAndJson(http.StatusBadRequest, errResp, restful.MIME_JSON)
        log.Println("Invalid Key ID")
```

Gambar 3.24 (Controller delete key (1))

```go
    if res.DeletedCount == 0 {
        errResp := ds.Response{Code: 401, Message: "Invalid Key ID"}
        response.WriteHeaderAndJson(http.StatusBadRequest, errResp, restful.MIME_JSON)
        log.Println("Invalid Key ID")
    } else {
        m := ds.Response{Code: 200, Message: "Success deleting key"}
        response.WriteHeaderAndJson(http.StatusOK, m, restful.MIME_JSON)
        log.Printf("Success delete key for (key: %s)\n", keyID)
    }

}
```

Gambar 3.23 (Controller delete key (2))

Gambar 3.23 dan gambar 3.24 merupakan *controller* yang *endpoint* yang akan menghapus *key* untuk *user* yang spesifik berdasarkan *id* dari *user* tersebut.

```go
package controller

import (
    "context"
    "log"
    "net/http"
    ds "privyid-on-premises-log-system/gateway/gateway-rest/datastructure"
    con "privyid-on-premises-log-system/pkg/dbinstance"
    "time"

    restful "github.com/emicklei/go-restful"
    . "github.com/gobeam/mongo-go-pagination"
    "go.mongodb.org/mongo-driver/bson"
    "go.mongodb.org/mongo-driver/bson/primitive"
)

func GetDetails(request *restful.Request, response *restful.Response) {
    keyID := request.PathParameter("keyID")
    if len(keyID) <= 0 {
        resp := ds.Response{Code: 402, Message: "Blank keyID"}
        response.WriteHeaderAndJson(http.StatusInternalServerError, resp, restful.MIME_JSON)
        return
    }
    mongoCollection := con.GetKeyCollection()
    ctx, _ := context.WithTimeout(context.Background(), 15*time.Second)
    result := new(ds.KeyStruct)
    idToHex, _ := primitive.ObjectIDFromHex(keyID)
    filter := bson.D{{"_id", idToHex}}
    err := mongoCollection.FindOne(ctx, filter).Decode(&result)
    if err != nil {
        resp := ds.Response{Code: 401, Message: "Invalid keyID"}
        response.WriteHeaderAndJson(http.StatusInternalServerError, resp, restful.MIME_JSON)
        log.Println(err)
        return
    }
    response.WriteHeaderAndJson(http.StatusOK, result, restful.MIME_JSON)
}
```

Gambar 3.25 (Controller detail key)

Gambar 3.25 merupakan *controller* untuk mendapatkan *detail* data dari *key* yang telah dibuat dengan parameter *id* dari *key* tersebut.

```go
func GetKeyPagination(request *restful.Request, response *restful.Response) {
    paginationData := new(ds.PageParameters)
    errPageData := request.ReadEntity(&paginationData)
    var perPage int64
    var page int64
    if errPageData != nil {
        perPage = 10
        page = 1
    } else {
        perPage = int64(paginationData.PerPage)
        page = int64(paginationData.Page)
    }

    mongoCollection := con.GetKeyCollection()
    pageParam := new(ds.PageParameters)
    errParam := request.ReadEntity(&pageParam)
    if errParam != nil {
        log.Println("Error reading page parameter")
        resp := ds.Response{Code: 400, Message: "Error reading page parameter"}
        response.WriteHeaderAndJson(http.StatusBadRequest, resp, restful.MIME_JSON)
        return
    }

    projection := bson.D{
        {"_id", 1},
        {"name", 1},
        {"keystatus", 1},
        {"clientkey", 1},
        {"secretkey", 1},
        {"updatedat", 1},
        {"createdat", 1},
    }
    PaginatedData, err := New(mongoCollection).Limit(perPage).Page(page).Sort("name", pageParam.Order).S
```

Gambar 3.27 (List all key (1))

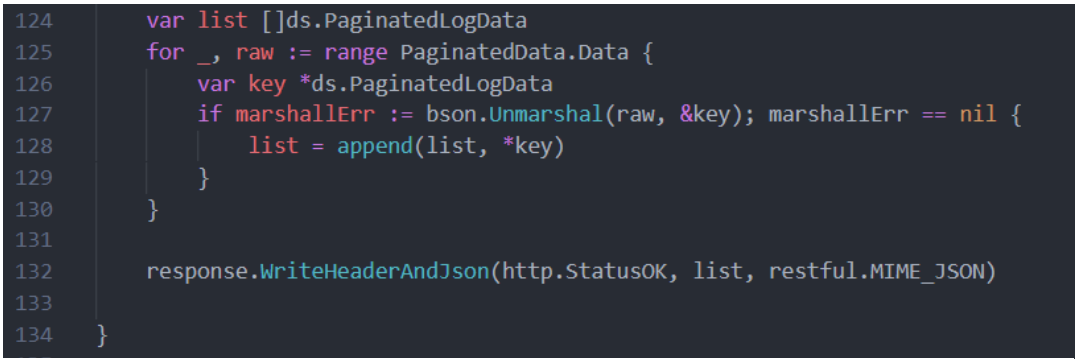
```go
    PaginatedData, err := New(mongoCollection).Limit(perPage).Page(page).Sort("name", pageParam.Order).Select(p
    if err != nil {
        log.Println(err)
        resp := ds.Response{Code: 401, Message: "Error getting list of pagination"}
        response.WriteHeaderAndJson(http.StatusBadRequest, resp, restful.MIME_JSON)
        return
    }

    var list []ds.PaginatedKeyData
    for _, raw := range PaginatedData.Data {
        var key *ds.PaginatedKeyData
        if marshallErr := bson.Unmarshal(raw, &key); marshallErr == nil {
            log.Println(key.Id)
            list = append(list, *key)
        }
    }

    response.WriteHeaderAndJson(http.StatusOK, list, restful.MIME_JSON)
}
```

Gambar 3.26 (List all key (2))

Gambar 3.26 dan gambar 3.27 adalah *controller* untuk mendapatkan *list* semua *key* yang pernah di-*generate* secara sistem.

```go
func GetLogPagination(request *restful.Request, response *restful.Response) {
    mongoCollection := con.GetLogCollection()
    paginationData := new(ds.PageParameters)
    errPageData := request.ReadEntity(&paginationData)

    var perPage int64
    var page int64
    if errPageData != nil {
        perPage = 10
        page = 1
    } else {
        perPage = int64(paginationData.PerPage)
        page = int64(paginationData.Page)
    }

    projection := bson.D{
        {"_id", 1},
        {"code", 1},
        {"datetime", 1},
        {"action", 1},
        {"payload", 1},
        {"fileprefix", 1},
        {"error", 1},
    }

    PaginatedData, err := New(mongoCollection).Limit(perPage).Page(page).Sort("name", paginationData.Order).Select
    if err != nil {
        log.Println(err)
        resp := ds.Response{Code: 401, Message: "Error getting list of pagination"}
        response.WriteHeaderAndJson(http.StatusBadRequest, resp, restful.MIME_JSON)
        return
    }
```

Gambar 3.29 (Controller list log (1))



```go
    var list []ds.PaginatedLogData
    for _, raw := range PaginatedData.Data {
        var key *ds.PaginatedLogData
        if marshallErr := bson.Unmarshal(raw, &key); marshallErr == nil {
            list = append(list, *key)
        }
    }

    response.WriteHeaderAndJson(http.StatusOK, list, restful.MIME_JSON)

}
```

Gambar 3.28 (Controller list log (2))

Gambar 3.28 dan gambar 3.29 adalah *controller* untuk menampilkan *log* dari spesifik *user*.

```
136 func GetDetailLog(request *restful.Request, response *restful.Response) {
137     logID := request.PathParameter("logID")
138     if len(logID) <= 0 {
139         resp := ds.Response{Code: 402, Message: "Blank logID"}
140         response.WriteHeaderAndJson(http.StatusInternalServerError, resp, restful.MIME_JSON)
141         log.Println("Blank logID")
142         return
143     }
144     mongoCollection := con.GetLogCollection()
145     ctx, _ := context.WithTimeout(context.Background(), 15*time.Second)
146     result := new(ds.Normalized)
147     idToHex, _ := primitive.ObjectIDFromHex(logID)
148     filter := bson.D{{"_id", idToHex}}
149     err := mongoCollection.FindOne(ctx, filter).Decode(&result)
150     if err != nil {
151         resp := ds.Response{Code: 401, Message: "Invalid logID"}
152         response.WriteHeaderAndJson(http.StatusInternalServerError, resp, restful.MIME_JSON)
153         log.Println(err)
154         return
155     }
156 
157     response.WriteHeaderAndJson(http.StatusOK, result, restful.MIME_JSON)
158 }
159
```

Gambar 3.30 (Controller detail log)

Gambar 3.30 merupakan *controller* untuk mendapatkan detail tentang *log* yang diambil berdasarkan *logid* yang dikirim melalui "*GetLogPagination"*.

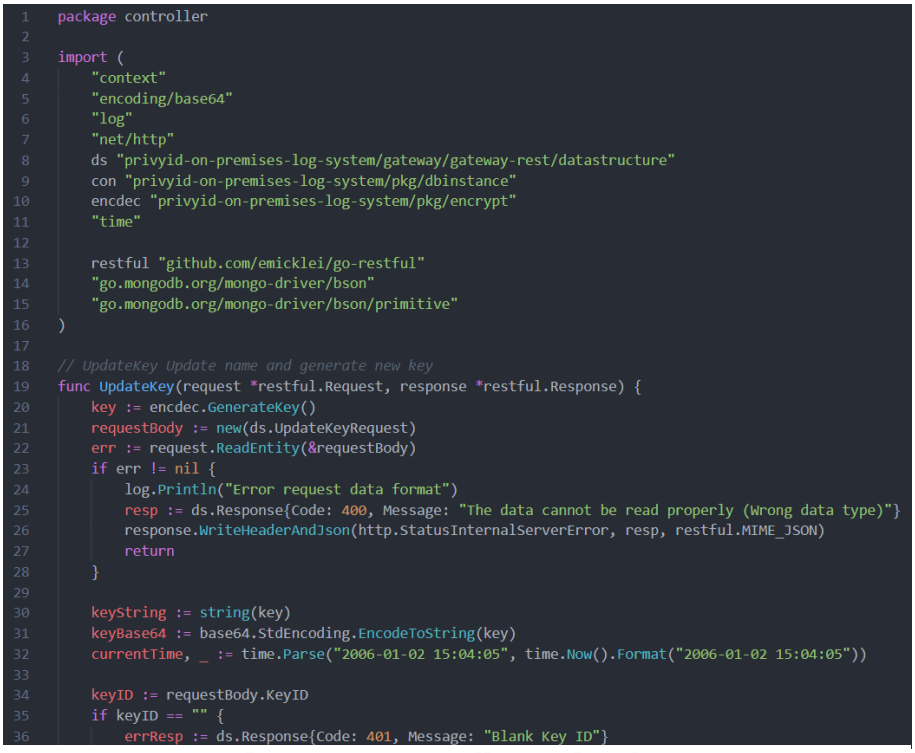
```go
package controller

import (
    "context"
    "crypto/aes"
    "log"
    "net/http"
    "os"
    ds "privyid-on-premises-log-system/gateway/gateway-rest/datastructure"
    con "privyid-on-premises-log-system/pkg/dbinstance"
    encdec "privyid-on-premises-log-system/pkg/encrypt"
    "time"

    restful "github.com/emicklei/go-restful"
)

//SendData is the function called when routing to /log/send
func SendData(request *restful.Request, response *restful.Response) {

    logData := new(ds.LogData)
    err := request.ReadEntity(&logData)

    if err != nil {
        resp := ds.Response{Code: 400, Message: "The data cannot be read properly (Wrong data type)"}
        response.WriteHeaderAndJson(http.StatusInternalServerError, resp, restful.MIME_JSON)
        log.Println("Invalid request data format")
        return
    }

    if logData.RequestID == "" {
        resp := ds.Response{Code: 401, Message: "Request id is empty"}
        response.WriteHeaderAndJson(http.StatusInternalServerError, resp, restful.MIME_JSON)
        log.Println("Request id is empty")
        return
    }

```

Gambar 3.32 (Controller save log (1))

```go
        return
    }

    mongoCollection := con.GetLogCollection()
    ctx, _ := context.WithTimeout(context.Background(), 15*time.Second)
    key := encdec.GenerateKey()
    iv := encdec.GenerateIV()

    logData.Fileprefix = encdec.Encrypt(key, logData.Fileprefix, iv, aes.BlockSize)
    normalized := ds.Normalize(logData, key)

    _, insertErr := mongoCollection.InsertOne(ctx, normalized)
    if insertErr != nil {
        log.Println("InsertOne ERROR:", insertErr)
        os.Exit(1) // safely exit script on error
    }

    resp := ds.Response{Code: 200, Message: "Success inserting log data"}
    response.WriteHeaderAndJson(http.StatusBadRequest, resp, restful.MIME_JSON)
}
```

Gambar 3.31 (Controller save log (2))

Gambar 3.31 dan gambar 3.32 merupakan *controller* untuk menyimpan *log* yang yang dilakukan oleh *user*.



```go
package controller

import (
    "context"
    "encoding/base64"
    "log"
    "net/http"
    ds "privyid-on-premises-log-system/gateway/gateway-rest/datastructure"
    con "privyid-on-premises-log-system/pkg/dbinstance"
    encdec "privyid-on-premises-log-system/pkg/encrypt"
    "time"

    restful "github.com/emicklei/go-restful"
    "go.mongodb.org/mongo-driver/bson"
    "go.mongodb.org/mongo-driver/bson/primitive"
)

// UpdateKey Update name and generate new key
func UpdateKey(request *restful.Request, response *restful.Response) {
    key := encdec.GenerateKey()
    requestBody := new(ds.UpdateKeyRequest)
    err := request.ReadEntity(&requestBody)
    if err != nil {
        log.Println("Error request data format")
        resp := ds.Response{Code: 400, Message: "The data cannot be read properly (Wrong data type)"}
        response.WriteHeaderAndJson(http.StatusInternalServerError, resp, restful.MIME_JSON)
        return
    }

    keyString := string(key)
    keyBase64 := base64.StdEncoding.EncodeToString(key)
    currentTime, _ := time.Parse("2006-01-02 15:04:05", time.Now().Format("2006-01-02 15:04:05"))

    keyID := requestBody.KeyID
    if keyID == "" {
        errResp := ds.Response{Code: 401, Message: "Blank Key ID"}
```

Gambar 3.34 (Controller update key (1))

```go
    if keyID == "" {
        errResp := ds.Response{Code: 401, Message: "Blank Key ID"}
        response.WriteHeaderAndJson(http.StatusBadRequest, errResp, restful.MIME_JSON)
        return
    }

    name := requestBody.Name
    var update bson.M
    if name == "" {
        update = bson.M{"$set": bson.M{"updatedat": currentTime, "name": name, "clientkey": keyString, "secretkey": keyBase64}}
    } else {
        update = bson.M{"$set": bson.M{"updatedat": currentTime, "clientkey": keyString, "secretkey": keyBase64}}
    }

    mongoCollection := con.GetKeyCollection()
    ctx, _ := context.WithTimeout(context.Background(), 15*time.Second)
    idToHex, _ := primitive.ObjectIDFromHex(keyID)
    filter := bson.D{{"_id", idToHex}}

    res, _ := mongoCollection.UpdateOne(ctx, filter, update)

    if res.MatchedCount == 0 {
        errResp := ds.Response{Code: 402, Message: "Invalid Key ID"}
        response.WriteHeaderAndJson(http.StatusBadRequest, errResp, restful.MIME_JSON)
        log.Println("Invalid Key ID")
    } else {
        m := ds.Response{Code: 200, Message: "Success updating key and name for the current user"}
        response.WriteHeaderAndJson(http.StatusOK, m, restful.MIME_JSON)
        log.Printf("Success update new key for (key: %s, name: %s)\n", keyID, name)
    }

}
```

Gambar 3.33 (Controller update key (2))

Gambar 3.33 dan gambar 3.34 adalah *controller* untuk menmbuat pasangan *encryption key* yang baru untuk *user*.

```go
func UpdateStatusKey(request *restful.Request, response *restful.Response) {
    requestBody := new(ds.UpdateStatusKey)
    err := request.ReadEntity(&requestBody)
    if err != nil {
        log.Println("Error request data format")
        resp := ds.Response{Code: 401, Message: "The data cannot be read properly (Wrong data type)"}
        response.WriteHeaderAndJson(http.StatusBadRequest, resp, restful.MIME_JSON)
        return
    }

    if requestBody.KeyID == "" {
        log.Println("Blank keyID in body request")
        resp := ds.Response{Code: 402, Message: "Blank keyID"}
        response.WriteHeaderAndJson(http.StatusBadRequest, resp, restful.MIME_JSON)
        return
    }

    if requestBody.Status == 0 {
        log.Println("Blank status in body request")
        resp := ds.Response{Code: 403, Message: "Blank status"}
        response.WriteHeaderAndJson(http.StatusBadRequest, resp, restful.MIME_JSON)
        return
    }

    update := bson.M{"$set": bson.M{"keystatus": requestBody.Status}}
    keyID := requestBody.KeyID

    mongoCollection := con.GetKeyCollection()
    ctx, _ := context.WithTimeout(context.Background(), 15*time.Second)
    idToHex, _ := primitive.ObjectIDFromHex(keyID)
    filter := bson.D{{"_id", idToHex}}

    res, _ := mongoCollection.UpdateOne(ctx, filter, update)

    if res.MatchedCount == 0 {
        errResp := ds.Response{Code: 402, Message: "Invalid Key ID"}
```

Gambar 3.36 (Controller update status key (1))

```go
    if res.MatchedCount == 0 {
        errResp := ds.Response{Code: 402, Message: "Invalid Key ID"}
        response.WriteHeaderAndJson(http.StatusBadRequest, errResp, restful.MIME_JSON)
        log.Println("Invalid Key ID")
    } else {
        m := ds.Response{Code: 200, Message: "Success updating key and name for the current user"}
        response.WriteHeaderAndJson(http.StatusOK, m, restful.MIME_JSON)
        log.Printf("Success update new key for (key: %s, status: %d)\n", keyID, requestBody.Status)
    }
}
```

Gambar 3.35 (Controller update status key (2))

Gambar 3.35 dan gambar 3.36 adalah *controller* untuk mengubah status *key* dengan memberikan *request* berupa *id* dari *key* tersebut.

3.4. Kendala yang Dihadapi

Terdapat beberapa kendala yang dihadapi saat melakukan proses praktek kerja magang, yaitu:

- Sulitnya saat penyesuaian awal dalam menggunakan bahasa pemrograman *Go*, karena belum pernah menggunakan bahasa tersebut pada saat pembelajaran di kampus.
- Dikarenakan kerja magang dimulai saat di tengah proyek yang sedang berjalan, harus memahami fungsi dari setiap *function* yang telah dibuat juga memahami *business-process* dari produk *Privacy Middleware* karena memiliki alur yang sedikit berbeda dengan produk PrivyID yang lainnya.
- Banyaknya *library open-source* dari *Go* yang akan digunakan dalam produk *Privacy Middleware* dan *Centralized Log System* sehingga membutuhkan waktu lebih untuk memahami konsep cara kerja dari masing-masing *library*, karena *library* yang digunakan adalah *library open-source*.
- *Version control* untuk mengatur versi dari *code* yang telah dibuat menggunakan *GitLab* yang sebelumnya belum pernah digunakan saat pembelajaran di kampus.
- Pembuatan *boilerplate* merupakan *template* awal dalam membangun *code-base* menggunakan konsep *Model-View-*

Controller (MVC) yang masih asing karena sebelumnya belum pernah digunakan saat pembelajaran di kampus.

- Dokumentasi *swagger* saat tidak menggunakan *gorestful* memiliki disiplin penulisan dalam malakukan dokumentasi, sehingga membutuhkan waktu lebih untuk memahami cara kerja dari *swagger* sendiri.

3.5. Solusi atas Kendala

Pada kendala tentang bahasa pemrograman baru, solusi yang dilakukan adalah untuk mencoba untuk menggunakannya secara langsung untuk dapat memahami cara pemakaiannya, juga membaca dokumentasi dan forum agar dapat lebih memahami karakteristik dan penggunaan dari bahasa pemrogaraman *Go*.

Pada kendala yang berhubungan dengan masuknya mahasiswa ditengah-tengah *project*, solusi yang dilakukan dengan berkomunikasi dengan *Tech Lead* dan *Product Manager* mengenai alur bisnis dari produk *Privacy Middleware* dan melakukan eksplorasi mandiri atas setiap *function* yang ada.

Pada kendala *library open-source* yang digunakan selama magang, solusi yang dilakukan adalah mencoba untuk mengimplementasikan *library* tersebut kedalam program sederhana yang kemudian ketika sudah mulai memahmi cara penggunaannya, diimplementasikan kedalam proyek.

Pada kendala *version control*, solusi yang dilakukan dengan mempelajari cara menggunakan *version control* khususnya *git* dengan memahami konsep cara kerja dari *git* dan memahami *basic comment* atau istilah yang ada pada *git*.

Pada kendala pembuatan *boilerplate* yang mengikuti format MVC adalah meminta *review* dan *feedback* atas *boilerplate* yang telah dibuat kepada *Tech Lead* dan *Senior*.

Pada kendala dokumentasi *swagger* solusinya adalah melakukan *trial and error* dalam mempelajari *swagger*, karena penulisan spasi saja akan membuat dokumentasi tidak terbaca.