**Soal**

1. Buat Simple CRUD RESTful API menggunakan Bahasa Pemograman yang kamu bisa dan Database yang kamu bisa. (Golang dan NoSQL nilai Plus)

Contoh : CRUD Banner, CRUD Siswa, atau CRUD Guru

1. Buatlah simple Session Login dimana session tersebut disimpan pada Redis dengan waktu 30s (TTL)

**Jawaban**

1. **Golang & MongoDB**

package main  
  
import (  
 "context"  
 "encoding/json"  
 "log"  
 "net/http"  
  
 "github.com/gorilla/mux"  
 "go.mongodb.org/mongo-driver/bson"  
 "go.mongodb.org/mongo-driver/bson/primitive"  
 "go.mongodb.org/mongo-driver/mongo"  
 "go.mongodb.org/mongo-driver/mongo/options"  
)  
  
var ctx = context.Background()  
  
type Student struct {  
 Id primitive.ObjectID `json:"id" bson:"\_id,omitempty"`  
 Name string `json:"name" bson:"Name,omitempty"`  
 Grade int `json:"grade" bson:"Grade,omitempty"`  
}  
  
type BasicResponse struct {  
 Code int `json:"code"`  
 Message string `json:"message"`  
}  
  
func main(){  
 serverAPI := mux.NewRouter()  
  
 serverAPI.HandleFunc("/students", getStudents).Methods("GET")  
 serverAPI.HandleFunc("/students/{id}", getStudent).Methods("GET")  
 serverAPI.HandleFunc("/students", createStudent).Methods("POST")  
 serverAPI.HandleFunc("/students/{id}", updateStudent).Methods("PUT")  
 serverAPI.HandleFunc("/students/{id}", deleteStudent).Methods("DELETE")  
  
 log.Println("Server :8000")  
 log.Fatal(http.ListenAndServe(":8000", serverAPI))  
}  
/\* START API HANDLER \*/  
func getStudents(w http.ResponseWriter, r \*http.Request) {  
 w.Header().Set("Content-Type", "application/json")  
 resultData, err := allData()  
 if err != nil{  
 w.WriteHeader(400)  
 json.NewEncoder(w).Encode(BasicResponse{Code:1003, Message:"Error when execute query to database"})  
 return  
 }  
  
 json.NewEncoder(w).Encode(resultData)  
}  
  
func getStudent(w http.ResponseWriter, r \*http.Request) {  
 w.Header().Set("Content-Type", "application/json")  
  
 var params = mux.Vars(r)  
  
 id, err := primitive.ObjectIDFromHex(params["id"])  
 if err != nil{  
 log.Println(err.Error())  
 w.WriteHeader(400)  
 json.NewEncoder(w).Encode(BasicResponse{Code:1001, Message:"Your id isn't valid"})  
 return  
 }  
  
 resultData := findData(id)  
 if resultData == (Student{}){  
 w.WriteHeader(400)  
 json.NewEncoder(w).Encode(BasicResponse{Code:1002, Message:"Data not found"})  
 return  
 }  
  
 json.NewEncoder(w).Encode(resultData)  
}  
  
func createStudent(w http.ResponseWriter, r \*http.Request){  
 w.Header().Set("Content-Type", "application/json")  
  
 var student Student  
  
 // we decode our body request params  
 err := json.NewDecoder(r.Body).Decode(&student)  
 if err != nil {  
 log.Println(err.Error())  
 w.WriteHeader(400)  
 json.NewEncoder(w).Encode(BasicResponse{Code:1004, Message:"Your body request isn't valid"})  
 return  
 }  
  
 result := insertData(student)  
 if result != nil {  
 w.WriteHeader(400)  
 json.NewEncoder(w).Encode(BasicResponse{Code:1003, Message:"Error when execute query to database"})  
 return  
 }  
  
 json.NewEncoder(w).Encode(BasicResponse{Code:2001, Message:"Success"})  
}  
  
func updateStudent(w http.ResponseWriter, r \*http.Request){  
 w.Header().Set("Content-Type", "application/json")  
  
 var params = mux.Vars(r)  
 var student Student  
  
 id, err := primitive.ObjectIDFromHex(params["id"])  
 if err != nil{  
 log.Println(err.Error())  
 w.WriteHeader(400)  
 json.NewEncoder(w).Encode(BasicResponse{Code:1001, Message:"Your id isn't valid"})  
 return  
 }  
 err = json.NewDecoder(r.Body).Decode(&student)  
 if err != nil {  
 log.Println(err.Error())  
 w.WriteHeader(400)  
 json.NewEncoder(w).Encode(BasicResponse{Code:1004, Message:"Your body request isn't valid"})  
 return  
 }  
 err = updateData(id, student)  
 if err != nil {  
 log.Println(err.Error())  
 w.WriteHeader(400)  
 json.NewEncoder(w).Encode(BasicResponse{Code:1003, Message:"Error when execute query to database"})  
 return  
 }  
  
 json.NewEncoder(w).Encode(BasicResponse{Code:2001, Message:"Success"})  
}  
  
func deleteStudent(w http.ResponseWriter, r \*http.Request){  
 w.Header().Set("Content-Type", "application/json")  
  
 var params = mux.Vars(r)  
  
 id, err := primitive.ObjectIDFromHex(params["id"])  
 if err != nil{  
 log.Println(err.Error())  
 w.WriteHeader(400)  
 json.NewEncoder(w).Encode(BasicResponse{Code:1001, Message:"Your id isn't valid"})  
 return  
 }  
  
 err = removeData(id)  
 if err != nil{  
 log.Println(err.Error())  
 w.WriteHeader(400)  
 json.NewEncoder(w).Encode(BasicResponse{Code:1003, Message:"Error when execute query to database"})  
 return  
 }  
  
 json.NewEncoder(w).Encode(BasicResponse{Code:2001, Message:"Success"})  
}  
/\* END API HANDLER \*/  
  
/\* START DB FUNCTION \*/  
func connect() (\*mongo.Database, error) {  
 clientOptions := options.Client()  
 clientOptions.ApplyURI("mongodb://localhost:27017")  
 client, err := mongo.NewClient(clientOptions)  
 if err != nil {  
 return nil, err  
 }  
  
 err = client.Connect(ctx)  
 if err != nil {  
 return nil, err  
 }  
  
 return client.Database("kdigital"), nil  
}  
  
func insertData(data Student) error {  
 db, err := connect()  
 if err != nil {  
 log.Fatal(err.Error())  
 return err  
 }  
  
 \_, err = db.Collection("Student").InsertOne(ctx, data)  
 if err != nil {  
 log.Fatal(err.Error())  
 return err  
 }  
  
 return nil  
}  
  
func findData(id primitive.ObjectID) Student{  
 var db, err = connect()  
 if err != nil {  
 log.Fatal(err.Error())  
 }  
  
 result := db.Collection("Student").FindOne(context.Background(), bson.M{"\_id": id})  
 var findData Student  
 err = result.Decode(&findData)  
 if err != nil {  
 log.Println(err.Error())  
 }  
 return findData  
}  
  
func allData() ([]Student, error) {  
 var db, err = connect()  
 if err != nil {  
 log.Fatal(err.Error())  
 }  
  
  
 csr, err := db.Collection("Student").Find(context.Background(), bson.M{})  
 if err != nil {  
 log.Println(err.Error())  
 return nil, err  
 }  
 students := make([]Student, 0)  
 for csr.Next(ctx){  
  
 var row Student  
 err := csr.Decode(&row)  
 if err != nil {  
 log.Println(err.Error())  
 return nil, err  
 }  
  
 students = append(students, row)  
 }  
 return students, nil  
}  
  
func removeData(id primitive.ObjectID) error {  
 db, err := connect()  
 if err != nil {  
 return err  
 }  
  
 \_, err = db.Collection("Student").DeleteOne(ctx, bson.M{"\_id": id})  
 if err != nil {  
 return err  
 }  
 return nil  
}  
  
func updateData(id primitive.ObjectID, data Student) error {  
 db, err := connect()  
 if err != nil {  
 return err  
 }  
  
 \_, err = db.Collection("Student").UpdateOne(ctx, bson.M{"\_id": id}, bson.M{"$set": data})  
 if err != nil {  
 return err  
 }  
  
 return nil  
}  
/\* END DB FUNCTION \*/

**2. Golang & Redis**

package main  
  
import (  
 "encoding/json"  
 "fmt"  
 "github.com/gomodule/redigo/redis"  
 "log"  
 "math/rand"  
 "net/http"  
 "time"  
)  
  
// Store the redis connection as a package level variable  
var cache redis.Conn  
var users = map[string]string{  
 "asdar": "asdar",  
 "admin": "admin",  
}  
type Credentials struct {  
 Password string `json:"password"`  
 Username string `json:"username"`  
}  
  
func main() {  
 initRedis()  
 http.HandleFunc("/login", Login)  
 http.HandleFunc("/home", Home)  
  
 log.Println("Server :8000")  
 log.Fatal(http.ListenAndServe(":8000", nil))  
}  
  
func initRedis() {  
 conn, err := redis.DialURL("redis://localhost")  
 if err != nil {  
 panic(err)  
 }  
 cache = conn  
}  
  
  
  
func RandomString(n int) string {  
 const *letterBytes* = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"  
 b := make([]byte, n)  
 for i := range b {  
 b[i] = *letterBytes*[rand.Intn(len(*letterBytes*))]  
 }  
 return string(b)  
}  
  
func Login(w http.ResponseWriter, r \*http.Request) {  
 var creds Credentials  
 err := json.NewDecoder(r.Body).Decode(&creds)  
 if err != nil {  
 w.WriteHeader(http.*StatusBadRequest*)  
 return  
 }  
  
 expectedPassword, ok := users[creds.Username]  
  
 if !ok || expectedPassword != creds.Password {  
 w.WriteHeader(http.*StatusUnauthorized*)  
 return  
 }  
  
  
 sessionToken := RandomString(10)  
 \_, err = cache.Do("SETEX", sessionToken, "30", creds.Username)  
 if err != nil {  
 w.WriteHeader(http.*StatusInternalServerError*)  
 return  
 }  
  
 http.SetCookie(w, &http.Cookie{  
 Name: "session\_token",  
 Value: sessionToken,  
 Expires: time.Now().Add(120 \* time.*Second*),  
 })  
}  
  
func Home(w http.ResponseWriter, r \*http.Request) {  
 c, err := r.Cookie("session\_token")  
 if err != nil {  
 if err == http.ErrNoCookie {  
 w.WriteHeader(http.*StatusUnauthorized*)  
 return  
 }  
 w.WriteHeader(http.*StatusBadRequest*)  
 return  
 }  
 sessionToken := c.Value  
  
 response, err := cache.Do("GET", sessionToken)  
 if err != nil {  
 w.WriteHeader(http.*StatusInternalServerError*)  
 return  
 }  
 if response == nil {  
 w.WriteHeader(http.*StatusUnauthorized*)  
 return  
 }  
 w.Write([]byte(fmt.Sprintf("Welcome %s!", response)))  
}