**Laporan Hasil Project Backend Praktikum**



Nama : Muhamad Ghandi Nur Setiawan

Nim : 434221014

Kelas : C-1

**Universitas Airlangga**

**Surabaya**

**2024**

**Hasil Project Backend Praktikum**

Tugas :

1. Buatlah API untuk :
2. Select 1 user berdasarkan ID nya

Controller userController

*// Get User by ID*

func GetUserOne(c \*fiber.Ctx) error {

    ctx, cancel := context.WithTimeout(context.Background(), 10\*time.Second)

    defer cancel()

    id := c.Params("id")

    userID, err := primitive.ObjectIDFromHex(id)

    if err != nil {

        return c.Status(http.StatusBadRequest).JSON(fiber.Map{"error": "Invalid ID"})

    }

    var user models.User

    err = userCollection.FindOne(ctx, bson.M{"\_id": userID}).Decode(&user)

    if err != nil {

        return c.Status(http.StatusNotFound).JSON(fiber.Map{"error": "User not found"})

    }

    return c.Status(http.StatusOK).JSON(user)

}

Route

*// get user by id*

    Users.Get("/getUser/:id", controllers.GetUserOne)

1. Update data user berdasarkan ID nya

Controller UserController

*// Update User by ID*

func UpdateUser(c \*fiber.Ctx) error {

    ctx, cancel := context.WithTimeout(context.Background(), 10\*time.Second)

    defer cancel()

    id := c.Params("id")

    userID, err := primitive.ObjectIDFromHex(id)

    if err != nil {

        return c.Status(http.StatusBadRequest).JSON(fiber.Map{"error": "Invalid ID"})

    }

*// Menerima data pengguna yang baru dari request body*

    var user models.User

    if err := c.BodyParser(&user); err != nil {

        return c.Status(http.StatusBadRequest).JSON(fiber.Map{"error": err.Error()})

    }

*// Membuat update data*

    update := bson.M{

        "username":      user.Username,

        "nm\_user":       user.Nm\_user,

        "pass":          user.Pass,

        "email":         user.Email,

        "role\_aktif":    user.Role\_aktif,

        "jenis\_kelamin": user.Jenis\_kelamin,

        "photo":         user.Photo,

        "phone":         user.Phone,

        "pass\_2":       user.Pass\_2,

    }

*// Melakukan update*

    result, err := userCollection.UpdateOne(ctx, bson.M{"\_id": userID}, bson.M{"$set": update})

    if err != nil {

        return c.Status(http.StatusInternalServerError).JSON(fiber.Map{"error": err.Error()})

    }

*// Jika tidak ada dokumen yang terpengaruh, pengguna tidak ditemukan*

    if result.MatchedCount == 0 {

        return c.Status(http.StatusNotFound).JSON(fiber.Map{"error": "User not found"})

    }

*// Mengambil pengguna yang diperbarui untuk dikembalikan*

    var updatedUser models.User

    err = userCollection.FindOne(ctx, bson.M{"\_id": userID}).Decode(&updatedUser)

    if err != nil {

        return c.Status(http.StatusInternalServerError).JSON(fiber.Map{"error": err.Error()})

    }

    return c.Status(http.StatusOK).JSON(updatedUser)

}

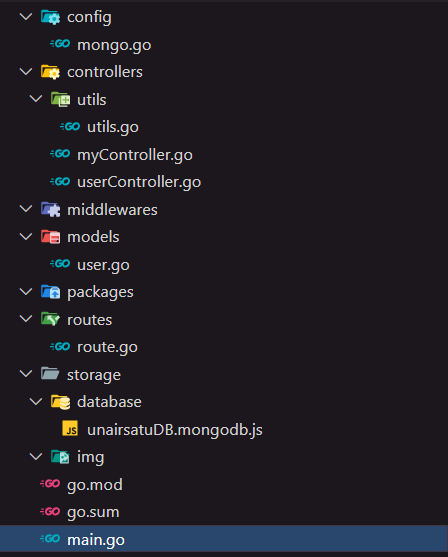
Route

*// update user by id*

    Users.Put("/updateUser/:id", controllers.UpdateUser)

1. Test dan screenshot hasilnya. Yang anda screenshot

Struktur project :



1. Code anda

Controller :

package controllers

import (

    "context"

    "net/http"

    "time"

    "github.com/gofiber/fiber/v2"

    "go.mongodb.org/mongo-driver/bson"

    "go.mongodb.org/mongo-driver/bson/primitive"

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

    "project-crud/config"

    "project-crud/controllers/utils"

    "project-crud/models"

)

var userCollection \*mongo.Collection = config.GetCollection("users")

*// Create User*

func CreateUser(c \*fiber.Ctx) error {

    ctx, cancel := context.WithTimeout(context.Background(), 10\*time.Second)

    defer cancel()

    var user models.User

    if err := c.BodyParser(&user); err != nil {

        return c.Status(http.StatusBadRequest).JSON(fiber.Map{"error": err.Error()})

    }

*// Parse id\_jenis\_user dari string ke ObjectID*

    idJenisUser, err := primitive.ObjectIDFromHex(user.Id\_jenis\_user.Hex())

    if err != nil {

        return c.Status(http.StatusBadRequest).JSON(fiber.Map{"error": "Invalid id\_jenis\_user format"})

    }

    user.Id\_jenis\_user = idJenisUser

    loc, err := time.LoadLocation("Asia/Jakarta")

    if err != nil {

        return c.Status(http.StatusInternalServerError).JSON(fiber.Map{"error": err.Error()})

    }

    user.Created\_at = primitive.NewDateTimeFromTime(time.Now().In(loc))

*// Generate token acak*

    token, err := utils.GenerateRandomString(32)

    if err != nil {

        return c.Status(http.StatusInternalServerError).JSON(fiber.Map{"error": "Failed to generate token"})

    }

    newUser := models.User{

**ID**:            primitive.NewObjectID(),

**Username**:      user.Username,

**Nm\_user**:       user.Nm\_user,

**Pass**:          user.Pass,

**Email**:         user.Email,

**Role\_aktif**:    user.Role\_aktif,

**Created\_at**:    primitive.NewDateTimeFromTime(time.Now()),

**Jenis\_kelamin**: user.Jenis\_kelamin,

**Photo**:         user.Photo,

**Phone**:         user.Phone,

**Token**:         token,

**Id\_jenis\_user**: user.Id\_jenis\_user,

**Pass\_2**:        user.Pass\_2,

    }

    \_, errIns := userCollection.InsertOne(ctx, newUser)

    if errIns != nil {

        return c.Status(http.StatusInternalServerError).JSON(fiber.Map{"error": errIns.Error()})

    }

    return c.Status(http.StatusCreated).JSON(newUser)

}

*// Get All Users*

func GetUsers(c \*fiber.Ctx) error {

    ctx, cancel := context.WithTimeout(context.Background(), 10\*time.Second)

    defer cancel()

    var users []models.User

    cursor, err := userCollection.Find(ctx, bson.M{})

    if err != nil {

        return c.Status(http.StatusInternalServerError).JSON(fiber.Map{"error": err.Error()})

    }

    if err = cursor.All(ctx, &users); err != nil {

        return c.Status(http.StatusInternalServerError).JSON(fiber.Map{"error": err.Error()})

    }

    return c.Status(http.StatusOK).JSON(users)

}

*// Get User by ID*

func GetUserOne(c \*fiber.Ctx) error {

    ctx, cancel := context.WithTimeout(context.Background(), 10\*time.Second)

    defer cancel()

    id := c.Params("id")

    userID, err := primitive.ObjectIDFromHex(id)

    if err != nil {

        return c.Status(http.StatusBadRequest).JSON(fiber.Map{"error": "Invalid ID"})

    }

    var user models.User

    err = userCollection.FindOne(ctx, bson.M{"\_id": userID}).Decode(&user)

    if err != nil {

        return c.Status(http.StatusNotFound).JSON(fiber.Map{"error": "User not found"})

    }

    return c.Status(http.StatusOK).JSON(user)

}

*// Update User by ID*

func UpdateUser(c \*fiber.Ctx) error {

    ctx, cancel := context.WithTimeout(context.Background(), 10\*time.Second)

    defer cancel()

    id := c.Params("id")

    userID, err := primitive.ObjectIDFromHex(id)

    if err != nil {

        return c.Status(http.StatusBadRequest).JSON(fiber.Map{"error": "Invalid ID"})

    }

*// Menerima data pengguna yang baru dari request body*

    var user models.User

    if err := c.BodyParser(&user); err != nil {

        return c.Status(http.StatusBadRequest).JSON(fiber.Map{"error": err.Error()})

    }

*// Membuat update data*

    update := bson.M{

        "username":      user.Username,

        "nm\_user":       user.Nm\_user,

        "pass":          user.Pass,

        "email":         user.Email,

        "role\_aktif":    user.Role\_aktif,

        "jenis\_kelamin": user.Jenis\_kelamin,

        "photo":         user.Photo,

        "phone":         user.Phone,

        "pass\_2":       user.Pass\_2,

    }

*// Melakukan update*

    result, err := userCollection.UpdateOne(ctx, bson.M{"\_id": userID}, bson.M{"$set": update})

    if err != nil {

        return c.Status(http.StatusInternalServerError).JSON(fiber.Map{"error": err.Error()})

    }

*// Jika tidak ada dokumen yang terpengaruh, pengguna tidak ditemukan*

    if result.MatchedCount == 0 {

        return c.Status(http.StatusNotFound).JSON(fiber.Map{"error": "User not found"})

    }

*// Mengambil pengguna yang diperbarui untuk dikembalikan*

    var updatedUser models.User

    err = userCollection.FindOne(ctx, bson.M{"\_id": userID}).Decode(&updatedUser)

    if err != nil {

        return c.Status(http.StatusInternalServerError).JSON(fiber.Map{"error": err.Error()})

    }

    return c.Status(http.StatusOK).JSON(updatedUser)

}

Model :

package models

import (

    "go.mongodb.org/mongo-driver/bson/primitive"

)

*// User represents a user in MongoDB*

type User struct {

**ID**            primitive.ObjectID `bson:"\_id,omitempty"`

**Username**      string             `json:"username"`

**Nm\_user**       string             `json:"nm\_user"`

**Pass**          string             `json:"pass"`

**Email**         string             `json:"email"`

**Role\_aktif**    int                `json:"role\_aktif"`

**Created\_at**    primitive.DateTime `json:"created\_at"`

**Jenis\_kelamin** int                `json:"jenis\_kelamin"`

**Photo**         string             `json:"photo"`

**Phone**         string             `json:"phone"`

**Token**         string             `json:"token"`

**Id\_jenis\_user** primitive.ObjectID `json:"id\_jenis\_user"`

**Pass\_2**        string             `json:"pass\_2"`

}

Utils

package utils

import (

    "crypto/rand"

    "encoding/hex"

)

*// Fungsi untuk menghasilkan string acak dengan panjang tertentu*

func GenerateRandomString(length int) (string, error) {

    bytes := make([]byte, length/2)

    if \_, err := rand.Read(bytes); err != nil {

        return "", err

    }

    return hex.EncodeToString(bytes), nil

}

Main :

package main

import (

    "project-crud/routes"

    "github.com/gofiber/fiber/v2"

)

func main() {

    app := fiber.New()

    routes.RouteApp(app)

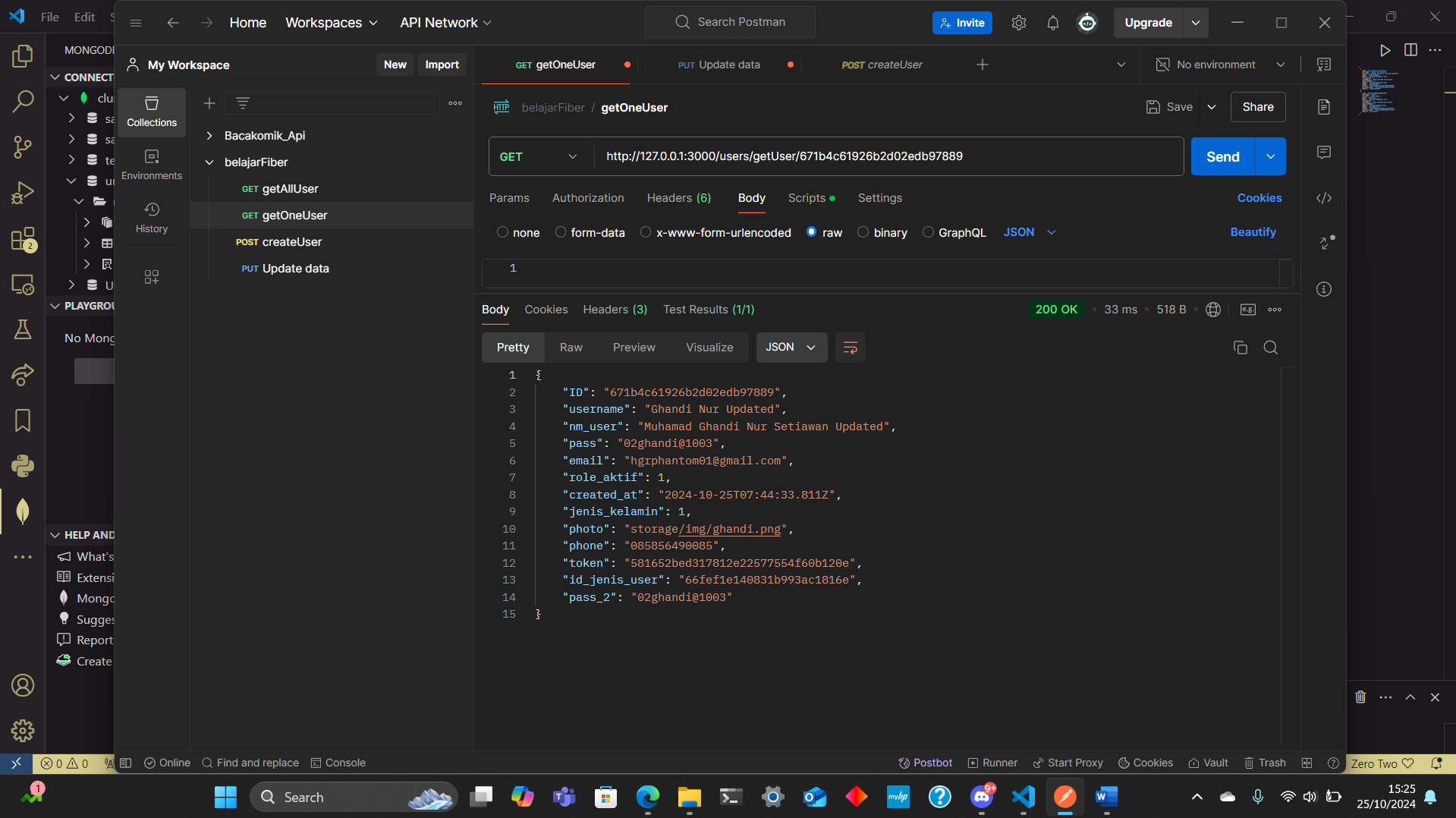
    app.Listen(":3000")

}

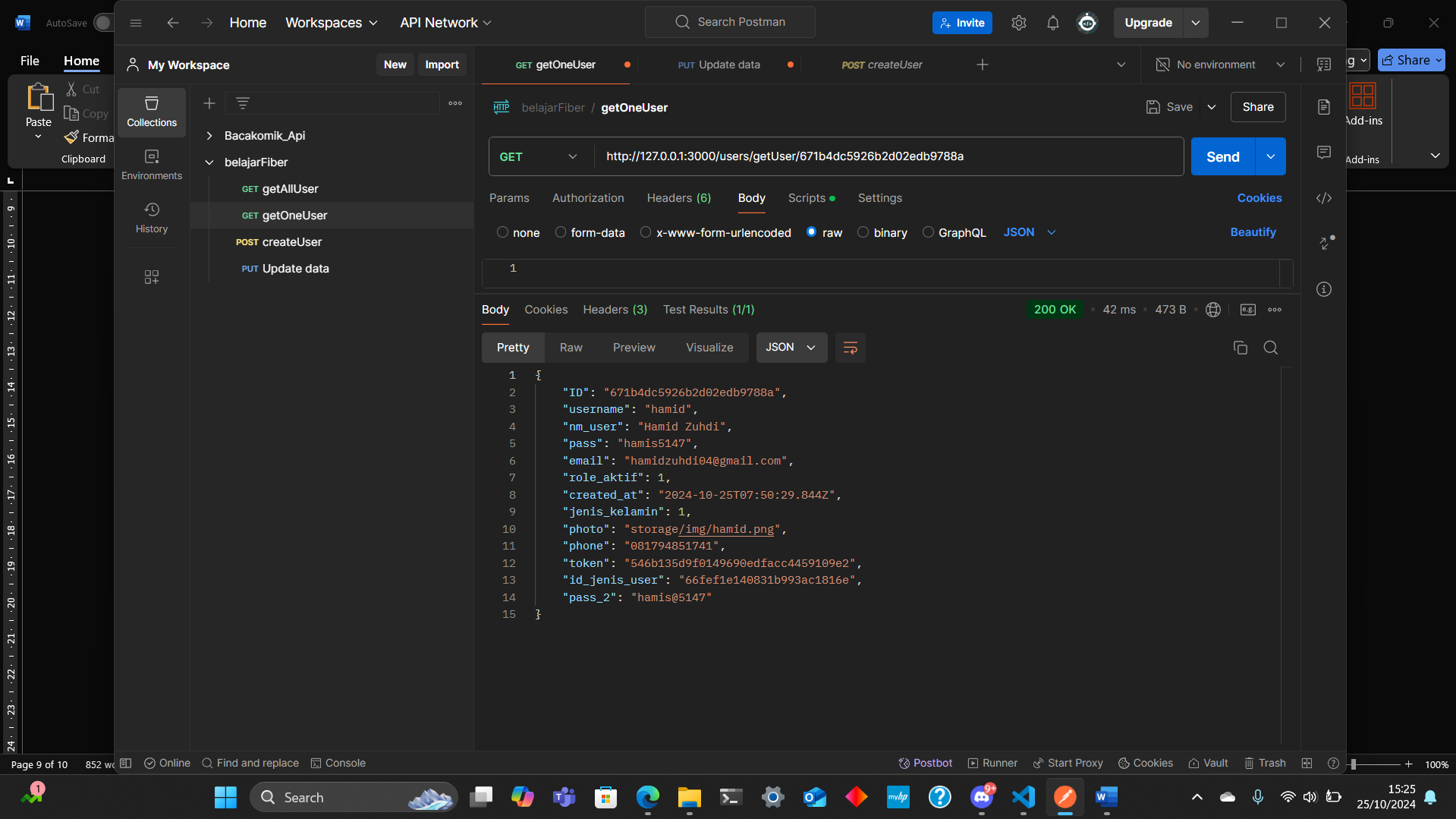
1. Request pada postman

Menampilkan user by id

* User 1 :

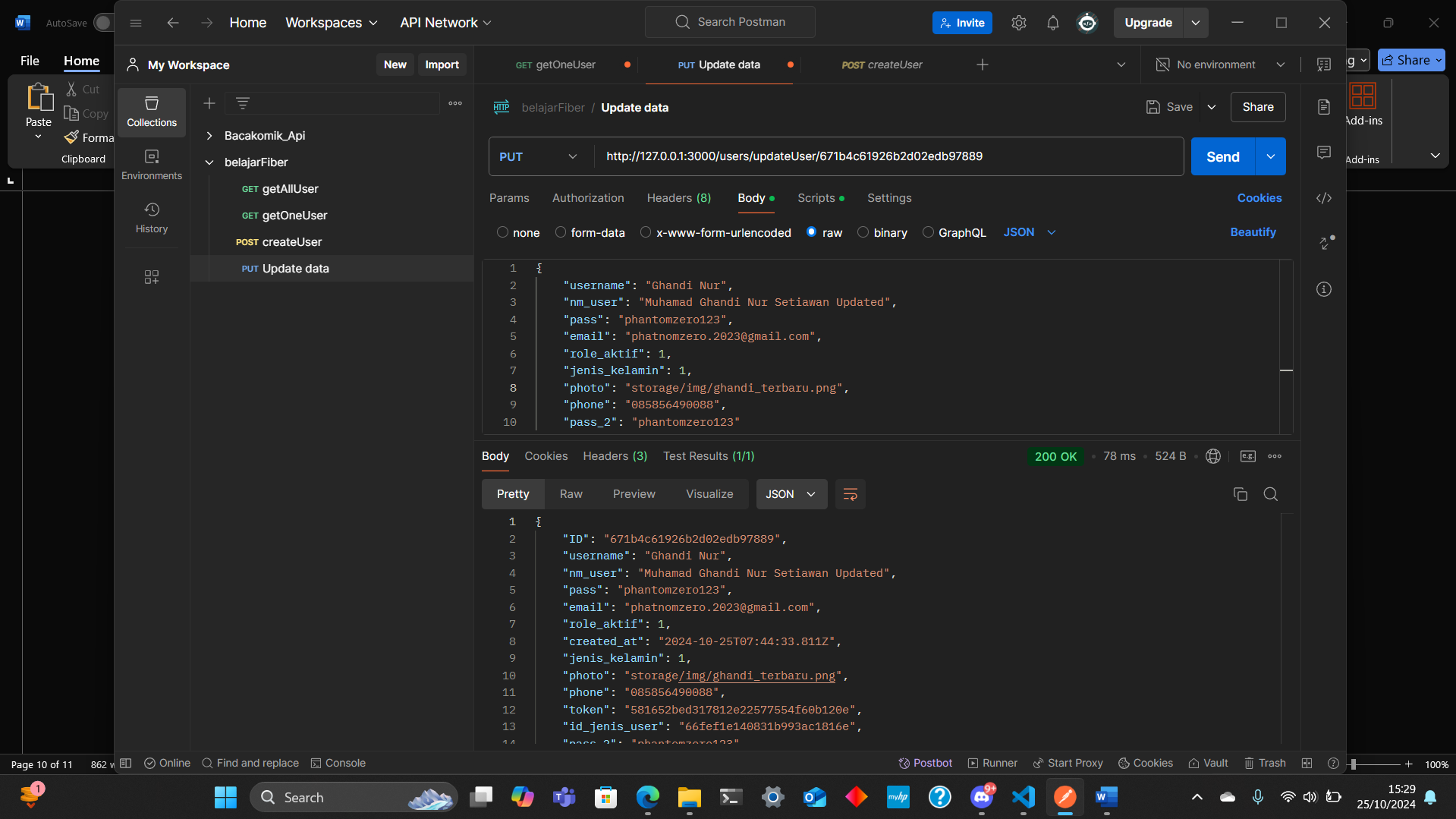


* User 2 :

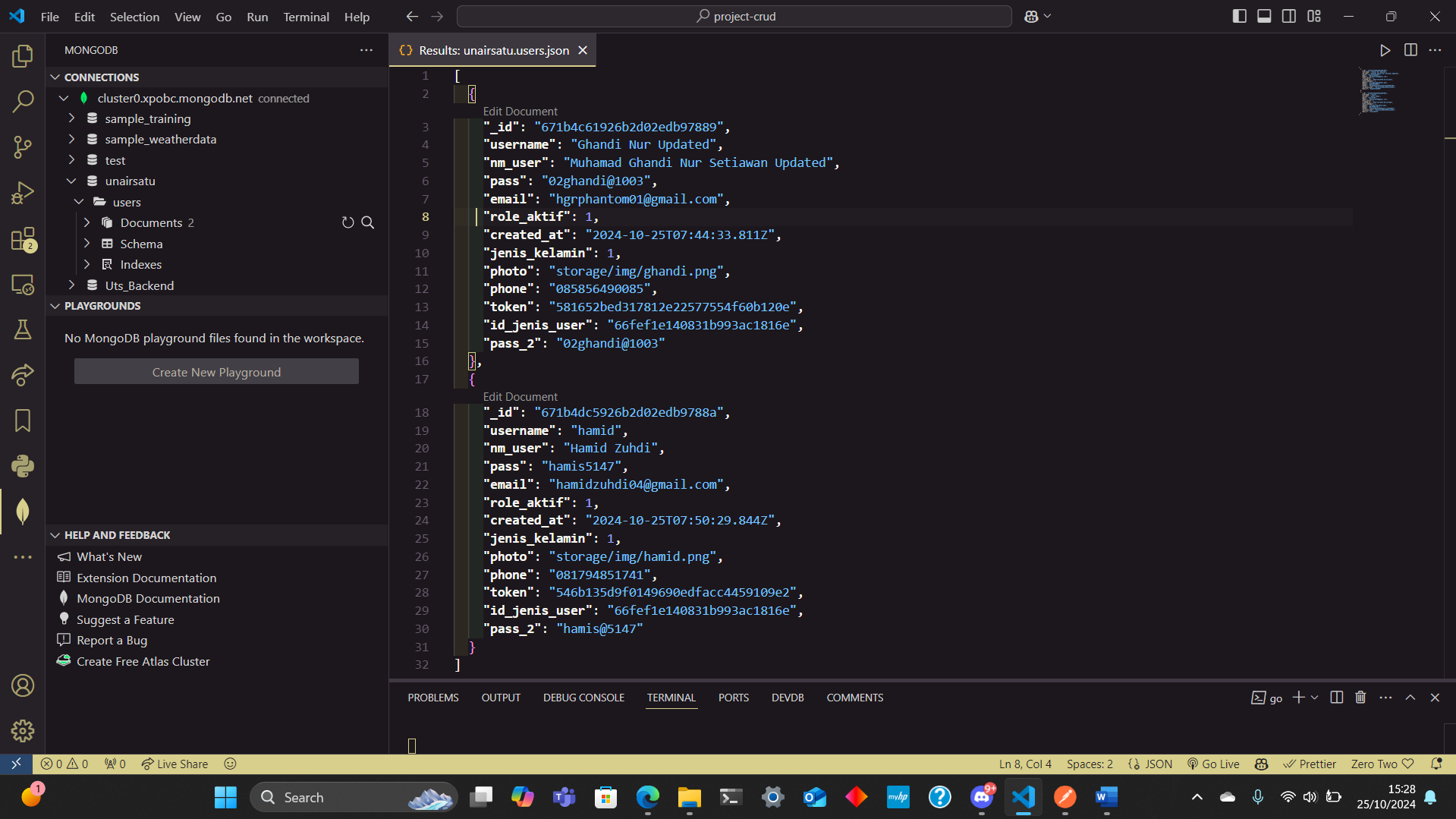


Update user by id

Update pada User 1 :



1. Collection pada mongoDB sebelum dan sesudah API dipanggil

Collection MongoDB Setelah Input :   


Collection MongoDB Setelah Update :

Update pada user 1 :

