Netcentric lab 9

Nguyen Manh Viet Khoi
ITCSIU21081

main.go

```go
package main

import (
   "fmt"
   "log"
   "math/rand"
   "net/http"
   "os"
   "strconv"
   "time"

   "github.com/jaswdr/faker"

   "github.com/gin-gonic/gin"
   "github.com/joho/godotenv"
   "gorm.io/driver/postgres"
   "gorm.io/gorm"
)

var db *gorm.DB

type User struct {
   ID        uint   `json:"id" gorm:"primaryKey"`
   Username  string `json:"username"`
   Firstname string `json:"firstname"`
   Lastname  string `json:"lastname"`
   Email     string `json:"email"`
   Avatar    string `json:"avatar"`
   Phone     string `json:"phone"`
   Dob       string `json:"dob"`
```

```go
    Country   string `json:"country"`
    City      string `json:"city"`
    Street    string `json:"street"`
    Address   string `json:"address"`
}

func main() {
    err := godotenv.Load()
    if err != nil {
        log.Fatal("Error loading .env file")
    }

    host := os.Getenv("DB_HOST")
    port := os.Getenv("DB_PORT")
    user := os.Getenv("DB_USER")
    password := os.Getenv("DB_PASSWORD")
    dbname := os.Getenv("DB_NAME")

    dsn := fmt.Sprintf("host=%s user=%s password=%s dbname=%s
port=%s sslmode=require TimeZone=Asia/Shanghai",
        host, user, password, dbname, port)
    db, err = gorm.Open(postgres.Open(dsn), &gorm.Config{})
    if err != nil {
        log.Fatal("Failed to connect to database:", err)
    }

    // Auto migrate the User model to create the users table
    db.AutoMigrate(&User{})

    // Create a new Gin router
    router := gin.Default()

    // Define routes and their handlers
    // Base URL
```

```go
    // localhost:8080/v1/
    {
        v1 := router.Group("/v1")
        v1.GET("/", func(c *gin.Context) {
            c.JSON(http.StatusOK, gin.H{"message": "Welcome to
the API"})
        })
        v1.POST("/users", createUser)
        v1.GET("/users", getUsers)
        v1.GET("/users/:id", getUser)
        v1.PUT("/users/:id", updateUser)
        v1.DELETE("/users/:id", deleteUser)
        v1.GET("/users/username/:username", findUserByUsername)
        v1.GET("/users/firstname/:firstname",
findUserByFirstname)
        v1.GET("/users/lastname/:lastname", findUserByLastname)
        // create fake data to the database
        v1.GET("/users/fake/:count", func(c *gin.Context) {
            count, _ := strconv.Atoi(c.Param("count"))
            insertData(db, count)
            c.JSON(http.StatusOK, gin.H{"message": "Fake data
created"})
        })

    }

    // Run the server
    router.Run(":8080")
}

func createUser(c *gin.Context) {
    var user User
    if err := c.BindJSON(&user); err != nil {
```

```
        c.JSON(http.StatusBadRequest, gin.H{"error": "Invalid
input"})
        return
    }

    if result := db.Create(&user); result.Error != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"error":
"Failed to create user"})
        return
    }
    c.JSON(http.StatusOK, gin.H{"message": "User created",
"user": user})
}

func getUsers(c *gin.Context) {
    var users []User
    if result := db.Find(&users); result.Error != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"error":
"Failed to retrieve users"})
        return
    }
    c.JSON(http.StatusOK, users)
}

func getUser(c *gin.Context) {
    id := c.Param("id")
    var user User
    if result := db.First(&user, id); result.Error != nil {
        if result.Error == gorm.ErrRecordNotFound {
            c.JSON(http.StatusNotFound, gin.H{"error": "User not
found"})
            return
        }
```

```
        c.JSON(http.StatusInternalServerError, gin.H{"error":
"Failed to retrieve user"})
        return
    }
    c.JSON(http.StatusOK, user)
}

func updateUser(c *gin.Context) {
    id := c.Param("id")
    var user User
    if err := c.BindJSON(&user); err != nil {
        c.JSON(http.StatusBadRequest, gin.H{"error": "Invalid
input"})
        return
    }

    if result := db.Model(&User{}).Where("id = ?",
id).Updates(user); result.Error != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"error":
"Failed to update user"})
        return
    }
    c.JSON(http.StatusOK, gin.H{"message": "User updated"})
}

func deleteUser(c *gin.Context) {
    id := c.Param("id")
    if result := db.Delete(&User{}, id); result.Error != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"error":
"Failed to delete user"})
        return
    }
    c.JSON(http.StatusOK, gin.H{"message": "User deleted"})
}
```

```go
// create a function find user base on username in the database
func findUserByUsername(c *gin.Context) {
   username := c.Param("username")
   var user User
   if result := db.Where("username = ?", username).First(&user);
result.Error != nil {
       if result.Error == gorm.ErrRecordNotFound {
           c.JSON(http.StatusNotFound, gin.H{"error": "User not
found"})
           return
       }
       c.JSON(http.StatusInternalServerError, gin.H{"error":
"Failed to retrieve user"})
       return
   }
   c.JSON(http.StatusOK, user)
}

// create a function to find user by firstname
func findUserByFirstname(c *gin.Context) {
   firstname := c.Param("firstname")
   var user User
   if result := db.Where("firstname = ?",
firstname).First(&user); result.Error != nil {
       if result.Error == gorm.ErrRecordNotFound {
           c.JSON(http.StatusNotFound, gin.H{"error": "User not
found"})
           return
       }
       c.JSON(http.StatusInternalServerError, gin.H{"error":
"Failed to retrieve user"})
       return
   }
```

```
    c.JSON(http.StatusOK, user)
}

// create a function to find user by lastname
func findUserByLastname(c *gin.Context) {
    lastname := c.Param("lastname")
    var user User
    if result := db.Where("lastname = ?", lastname).First(&user);
result.Error != nil {
        if result.Error == gorm.ErrRecordNotFound {
            c.JSON(http.StatusNotFound, gin.H{"error": "User not
found"})
            return
        }
        c.JSON(http.StatusInternalServerError, gin.H{"error":
"Failed to retrieve user"})
        return
    }
    c.JSON(http.StatusOK, user)
}

// INSERT INTO users (username, firstname, lastname, email,
avatar, phone, dob, country, city, street, address) VALUES
('john_doe', 'John', 'Doe', '
func insertData(db *gorm.DB, count int) {
    rand.New(rand.NewSource(time.Now().UnixNano()))

    for i := 0; i < count; i++ {
        fake := faker.New()
        rand.New(rand.NewSource(time.Now().UnixNano()))
        now := time.Now()
        user := User{
            Username:  fake.Person().FirstName() +
strconv.Itoa(rand.Intn(1000)),
```

```
            Firstname: fake.Person().FirstName(),
            Lastname:  fake.Person().LastName(),
            Email:     fake.Internet().Email(),
            Avatar:    fake.Internet().URL(),
            Phone:     fake.Phone().Number(),
            Dob:       now.Format(time.RFC3339),
            Country:   fake.Address().Country(),
            City:      fake.Address().City(),
            Street:    fake.Address().StreetName(),
            Address:   fake.Address().Address(),
        }
        db.Create(&user)
    }
}
```

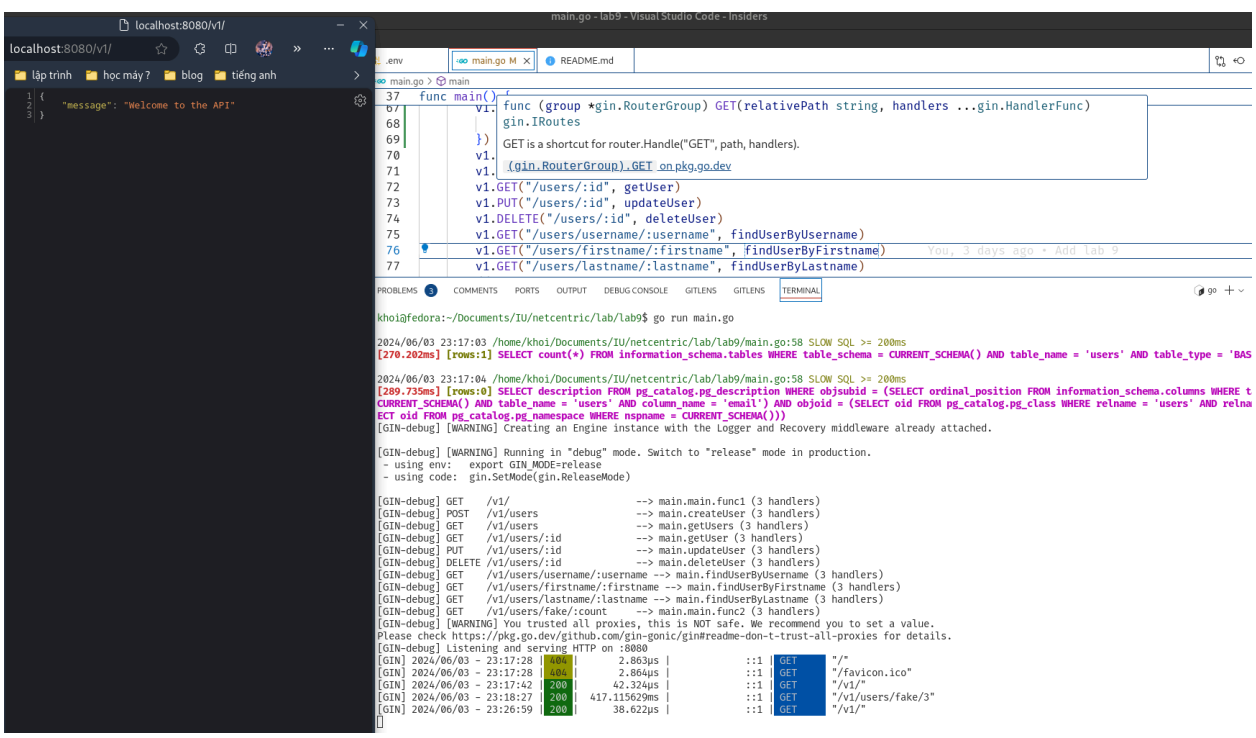
- This is a simple RESTful API server written in Go. It uses Gin framework for handling HTTP requests and GORM for interacting with a supabase (postgreSQL database online). The server provides endpoints for creating, reading, updating and deleting users as well, finding username, firstname or lastname. Beside that i also provide an endpoint to create fake data for testing.
- Here's a brief overview of the main components:
    - User struct: This is the data model for a user. It includes fields for the user's ID, username, first name, last name, email, avatar, phone number, date of birth, country, city, street, and address.
    - main function: This function loads environment variables from a .env file, connects to the database, sets up the Gin router, defines the routes and their handlers, and starts the server.
    - createUser, getUsers, getUser, updateUser, deleteUser functions: These are the handlers for the /users

endpoints. They interact with the database to create, read, update, and delete users.

- findUserByUsername, findUserByFirstname, findUserByLastname functions: These are the handlers for the /users/username/:username, /users/firstname/:firstname, and /users/lastname/:lastname endpoints. They find users by username, first name, or last name.
- insertData function: This function generates fake user data and inserts it into the database. It uses the github.com/jaswdr/faker package to generate the fake data.

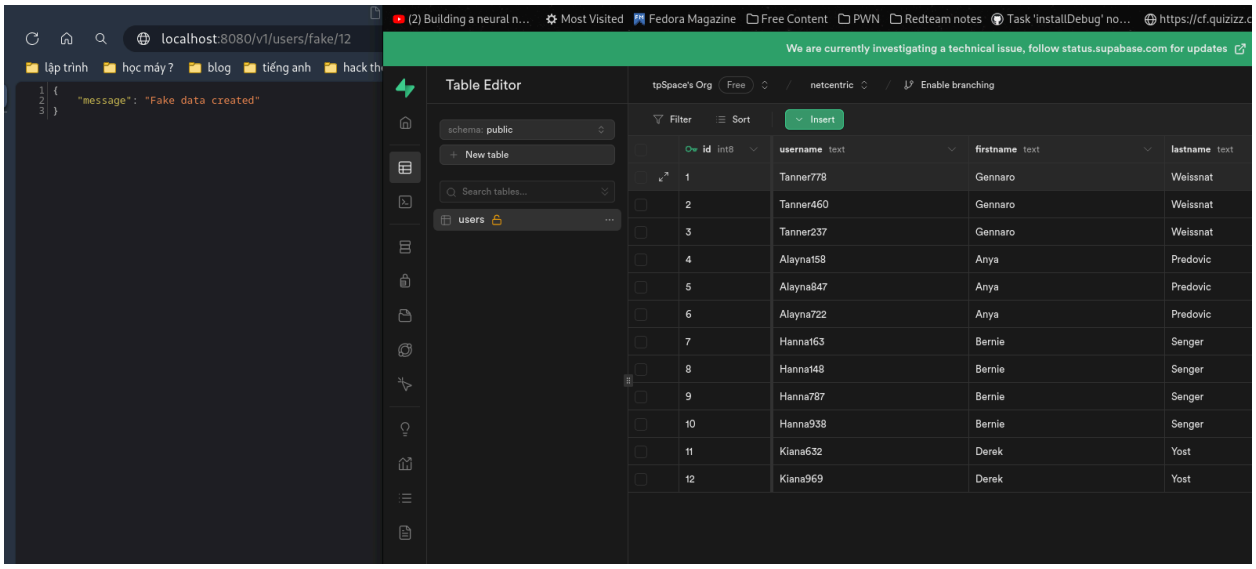
The server listens on port 8080 and provides versioned API endpoints under the /v1 path.
Base url localhost:8080/v1/

Results
http://localhost:8080/v1/



http://localhost:8080/v1/users/fake/12

- Generate fake data



http://localhost:8080/v1/users

- View all users information

localhost:8080/v1/users

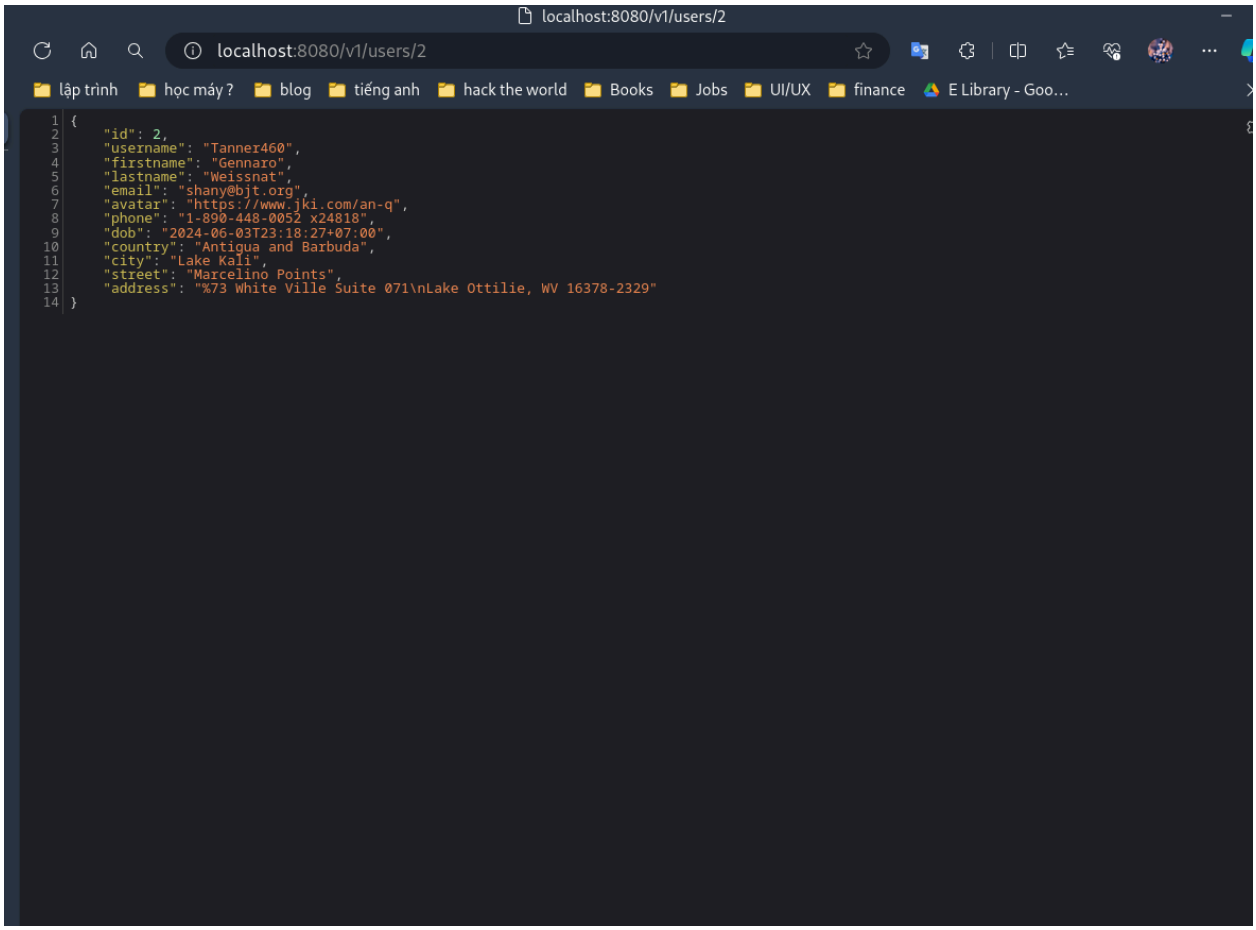
localhost:8080/v1/users

lập trình | học máy ? | blog | tiếng anh | hack the world | Books | Jobs | UI/UX | finance | E Library - Goo...


```
[
    {
        "id": 1,
        "username": "Tanner778",
        "firstname": "Gennaro",
        "lastname": "Weissnat",
        "email": "shany@bjt.org",
        "avatar": "https://www.jki.com/an-q",
        "phone": "1-890-448-0052 x24818",
        "dob": "2024-06-03T23:18:27+07:00",
        "country": "Antigua and Barbuda",
        "city": "Lake Kali",
        "street": "Marcelino Points",
        "address": "%73 White Ville Suite 071\nLake Ottilie, WV 16378-2329"
    },
    {
        "id": 2,
        "username": "Tanner460",
        "firstname": "Gennaro",
        "lastname": "Weissnat",
        "email": "shany@bjt.org",
        "avatar": "https://www.jki.com/an-q",
        "phone": "1-890-448-0052 x24818",
        "dob": "2024-06-03T23:18:27+07:00",
        "country": "Antigua and Barbuda",
        "city": "Lake Kali",
        "street": "Marcelino Points",
        "address": "%73 White Ville Suite 071\nLake Ottilie, WV 16378-2329"
    },
    {
        "id": 3,
        "username": "Tanner237",
        "firstname": "Gennaro",
        "lastname": "Weissnat",
        "email": "shany@bjt.org",
        "avatar": "https://www.jki.com/an-q",
        "phone": "1-890-448-0052 x24818",
        "dob": "2024-06-03T23:18:27+07:00",
        "country": "Antigua and Barbuda",
        "city": "Lake Kali",
        "street": "Marcelino Points",
        "address": "%73 White Ville Suite 071\nLake Ottilie, WV 16378-2329"
    },
    {
        "id": 4,
        "username": "Alayna158",
        "firstname": "Anya",
        "lastname": "Predovic",
        "email": "koss@ubz.com",
        "avatar": "http://www.hkz.com/mrif-lb",
        "phone": "840-999-7740 x883",
        "dob": "2024-06-03T23:28:26+07:00",
        "country": "Antigua and Barbuda",
        "city": "Rebekaside",
        "street": "Andreanne Crossroad",
        "address": "%6072 Jackson Lane\nDonaldchester, WY 93368-7442"
    },
    {
```

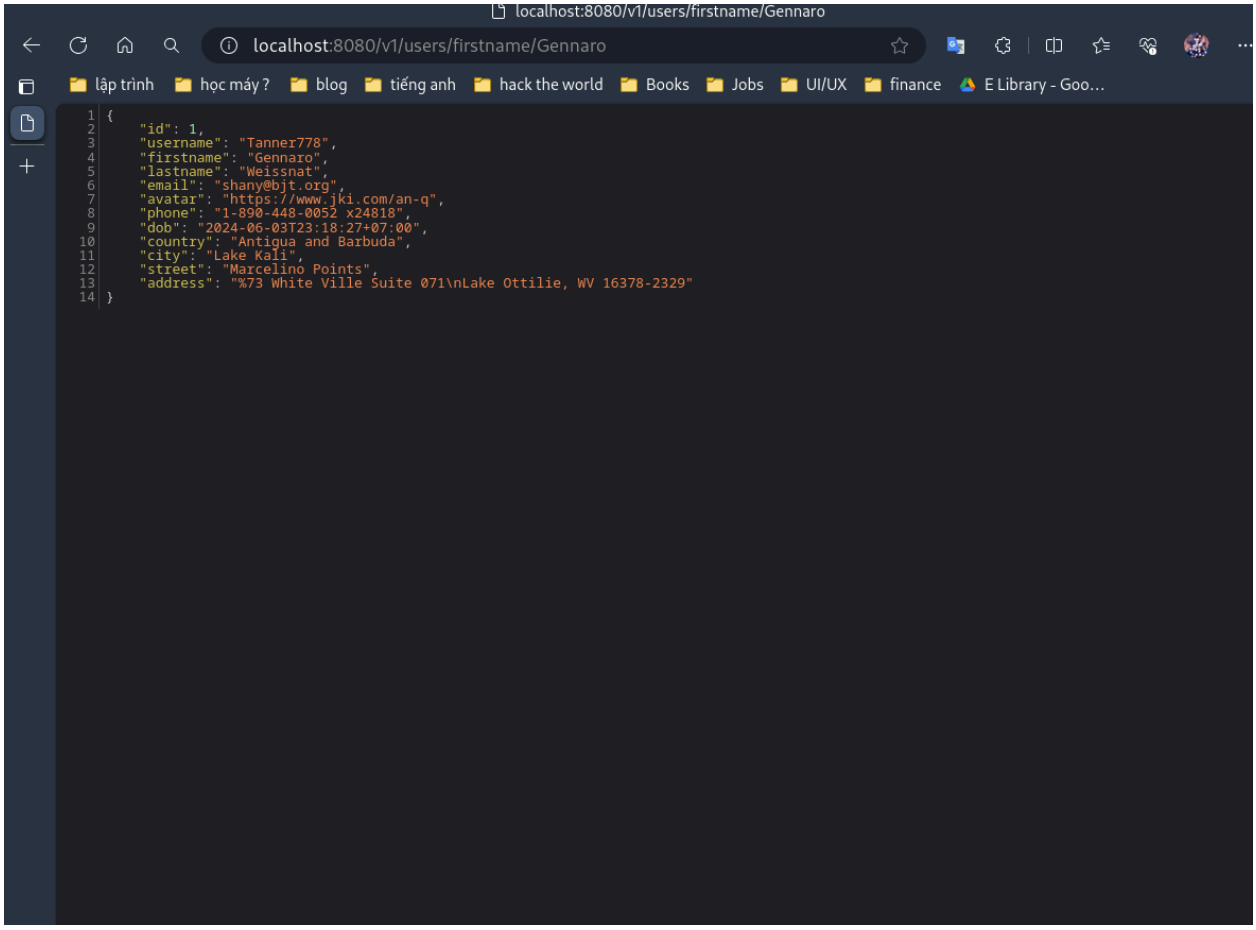
http://localhost:8080/v1/users/2

- Get users information base on id



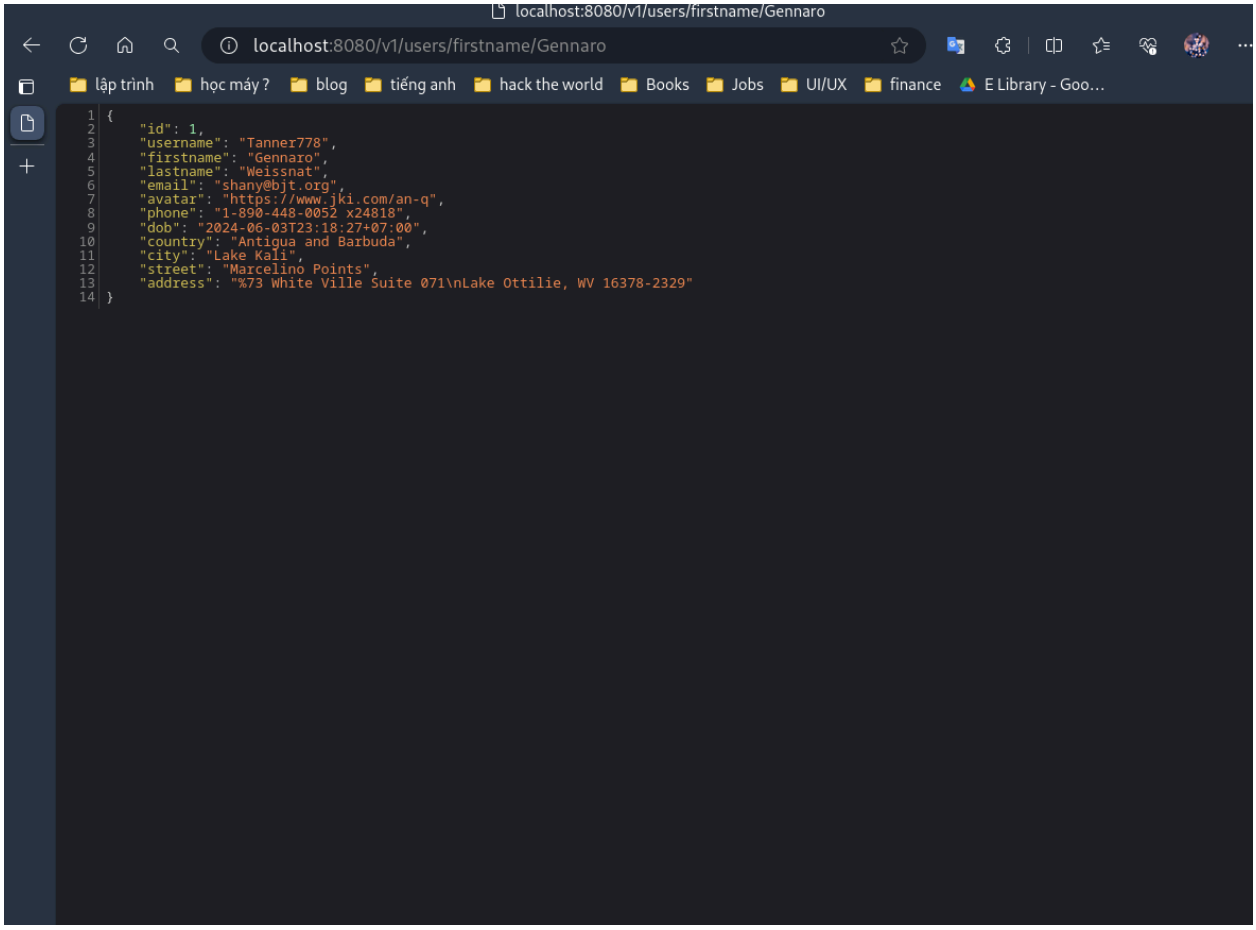
http://localhost:8080/v1/users/username/Tanner460

- Find user base on username



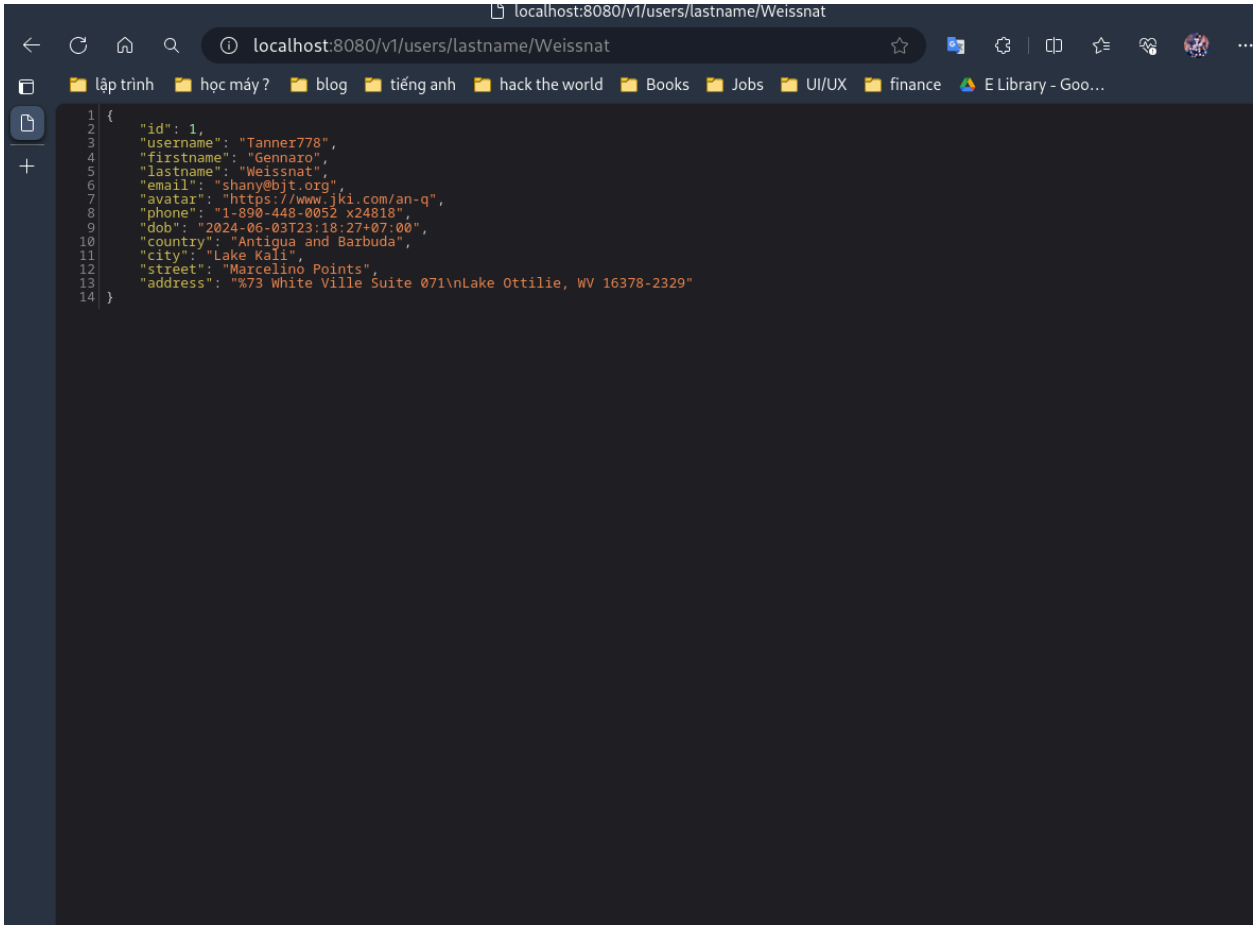
http://localhost:8080/v1/users/firstname/Gennaro

- Get firstname



http://localhost:8080/v1/users/lastname/Weissnat

- Find last name



- Post, Delete and put work well but i can’t install postman therefore no capture for evidence.