

# Netcentric lab 9

Nguyen Manh Viet Khoi  
ITCSIU21081

main.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"`
```

```

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

```

```

// 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"})
}

```

```

// 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](https://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

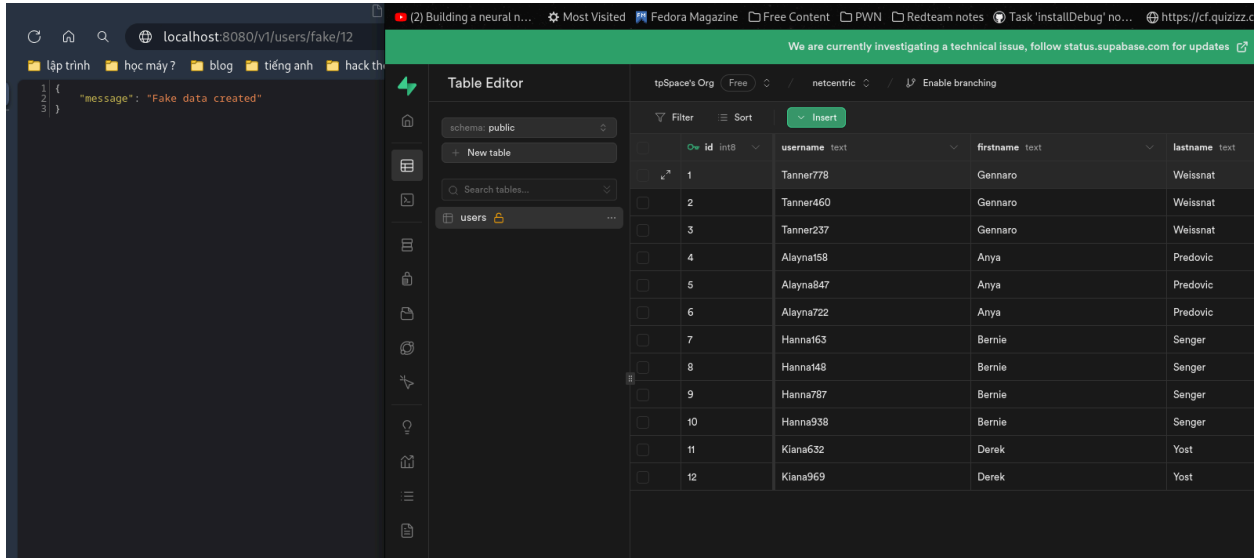
`http://localhost:8080/v1/`

```
func main() {
    v1 := func(group *gin.RouterGroup) GET(relativePath string, handlers ...gin.HandlerFunc) gin.IRoutes {
        GET is a shortcut for router.Handle("GET", path, handlers).
        v1 := (gin.RouterGroup).GET on pkg.go.dev
        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)
    }
    r := gin.Default()
    r.GET("/", func(c *gin.Context) {
        c.JSON(200, gin.H{
            "message": "Welcome to the API"
        })
    })
    r.Run()
}
```

```
2024/06/03 23:17:03 /home/khoi/Documents/IU/netcentric/lab/lab9/main.go:58 SLOW SQL >= 200ms
[278.202ms] [rows:1] SELECT count(*) FROM information_schema.tables WHERE table_schema = CURRENT_SCHEMA() AND table_name = 'users' AND table_type = 'BAS
2024/06/03 23:17:04 /home/khoi/Documents/IU/netcentric/lab/lab9/main.go:58 SLOW SQL >= 200ms
[289.735ms] [rows:0] SELECT description FROM pg_catalog.pg_description WHERE objsubid = (SELECT ordinal_position FROM information_schema.columns WHERE t
CURRENT_SCHEMA() AND table_name = 'users' AND column_name = 'email') AND objoid = (SELECT oid FROM pg_catalog.pg_class WHERE relname = 'users' AND relna
ECT oid FROM pg_catalog.pg_namespace WHERE namespace = CURRENT_SCHEMA()))
[GIN-debug] [WARNING] Creating an Engine instance with the Logger and Recovery middleware already attached.
[GIN-debug] [WARNING] Running in "debug" mode. Switch to "release" mode in production.
- using env: export GIN_MODE=release
- using code: gin.SetMode(gin.ReleaseMode)
[GIN-debug] GET    /v1/             --> main.main.func1 (3 handlers)
[GIN-debug] POST   /v1/users        --> main.createUser (3 handlers)
[GIN-debug] GET    /v1/users        --> main.getUsers (3 handlers)
[GIN-debug] GET    /v1/users/:id    --> main.getUser (3 handlers)
[GIN-debug] PUT    /v1/users/:id    --> main.updateUser (3 handlers)
[GIN-debug] DELETE /v1/users/:id    --> main.deleteUser (3 handlers)
[GIN-debug] GET    /v1/users/username/:username --> main.findUserByUsername (3 handlers)
[GIN-debug] GET    /v1/users/firstname/:firstname --> main.findUserByFirstname (3 handlers)
[GIN-debug] GET    /v1/users/lastname/:lastname --> main.findUserByLastname (3 handlers)
[GIN-debug] GET    /v1/users/fake/:count --> main.main.func2 (3 handlers)
[GIN-debug] [WARNING] You trusted all proxies, this is NOT safe. We recommend you to set a value.
Please check https://pkg.go.dev/github.com/gin-gonic/gin#readme-don-t-trust-all-proxies for details.
[GIN-debug] Listening and serving HTTP on :8080
[GIN] 2024/06/03 - 23:17:28 | 200 | 2.863µs | GET | "/"
[GIN] 2024/06/03 - 23:17:28 | 404 | 2.864µs | GET | "/favicon.ico"
[GIN] 2024/06/03 - 23:17:42 | 200 | 42.324µs | GET | "/v1/"
[GIN] 2024/06/03 - 23:18:27 | 200 | 417.115629ms | GET | "/v1/users/fake/3"
[GIN] 2024/06/03 - 23:26:59 | 200 | 38.622µs | GET | "/v1/"
```

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

- Generate fake data



The screenshot displays a web application interface. On the left, a REST client shows a successful POST request to `localhost:8080/v1/users/fake/12` with a JSON body `{ "message": "Fake data created" }`. On the right, the Supabase Table Editor is open, showing a table named `users` with columns `id`, `username`, `firstname`, and `lastname`. The table contains 12 rows of generated data.

id	username	firstname	lastname
1	Tanner778	Gennaro	Weissnat
2	Tanner460	Gennaro	Weissnat
3	Tanner237	Gennaro	Weissnat
4	Alayna158	Anyia	Predovic
5	Alayna847	Anyia	Predovic
6	Alayna722	Anyia	Predovic
7	Hanna163	Bernie	Senger
8	Hanna148	Bernie	Senger
9	Hanna787	Bernie	Senger
10	Hanna938	Bernie	Senger
11	Kiana632	Derek	Yost
12	Kiana969	Derek	Yost

<http://localhost:8080/v1/users>

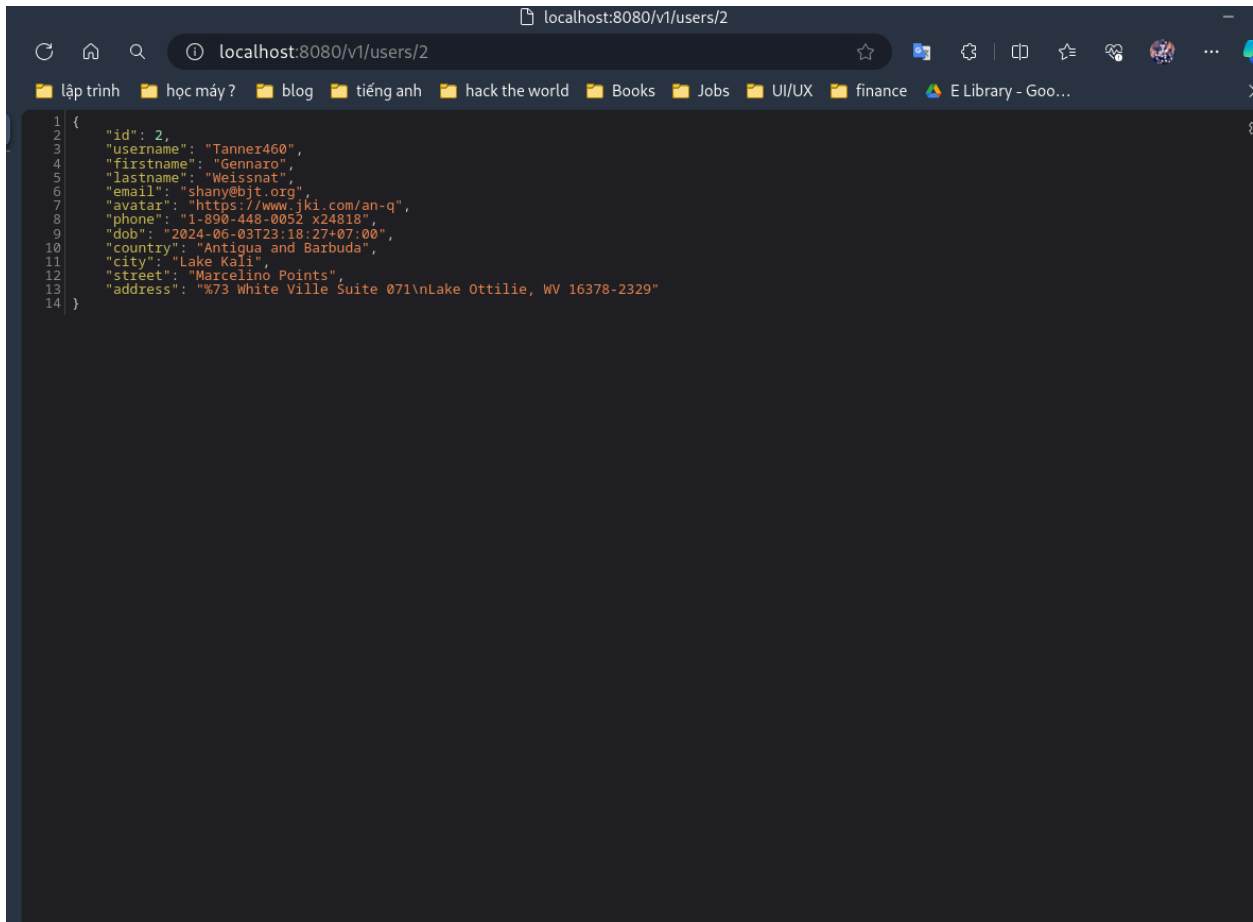
- View all users information

```
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...

1 [
2   {
3     "id": 1,
4     "username": "Tanner778",
5     "firstname": "Gennaro",
6     "lastname": "Weissnat",
7     "email": "shanyebjt.org",
8     "avatar": "https://www.jki.com/an-q",
9     "phone": "1-890-448-0052 x24818",
10    "dob": "2024-06-03T23:18:27+07:00",
11    "country": "Antigua and Barbuda",
12    "city": "Lake Kali",
13    "street": "Marcelino Points",
14    "address": "%73 White Ville Suite 071\\nLake Ottilie, WV 16378-2329"
15  },
16  {
17    "id": 2,
18    "username": "Tanner460",
19    "firstname": "Gennaro",
20    "lastname": "Weissnat",
21    "email": "shanyebjt.org",
22    "avatar": "https://www.jki.com/an-q",
23    "phone": "1-890-448-0052 x24818",
24    "dob": "2024-06-03T23:18:27+07:00",
25    "country": "Antigua and Barbuda",
26    "city": "Lake Kali",
27    "street": "Marcelino Points",
28    "address": "%73 White Ville Suite 071\\nLake Ottilie, WV 16378-2329"
29  },
30  {
31    "id": 3,
32    "username": "Tanner237",
33    "firstname": "Gennaro",
34    "lastname": "Weissnat",
35    "email": "shanyebjt.org",
36    "avatar": "https://www.jki.com/an-q",
37    "phone": "1-890-448-0052 x24818",
38    "dob": "2024-06-03T23:18:27+07:00",
39    "country": "Antigua and Barbuda",
40    "city": "Lake Kali",
41    "street": "Marcelino Points",
42    "address": "%73 White Ville Suite 071\\nLake Ottilie, WV 16378-2329"
43  },
44  {
45    "id": 4,
46    "username": "Alayna158",
47    "firstname": "Anyia",
48    "lastname": "Predovic",
49    "email": "koss@ubz.com",
50    "avatar": "http://www.hkz.com/mrif-lb",
51    "phone": "840-999-7740 x883",
52    "dob": "2024-06-03T23:28:26+07:00",
53    "country": "Antigua and Barbuda",
54    "city": "Rebekaside",
55    "street": "Andreanne Crossroad",
56    "address": "%072 Jackson Lane\\nDonaldchester, WV 93368-7442"
57  },
58  }
```

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

- Get users information base on id

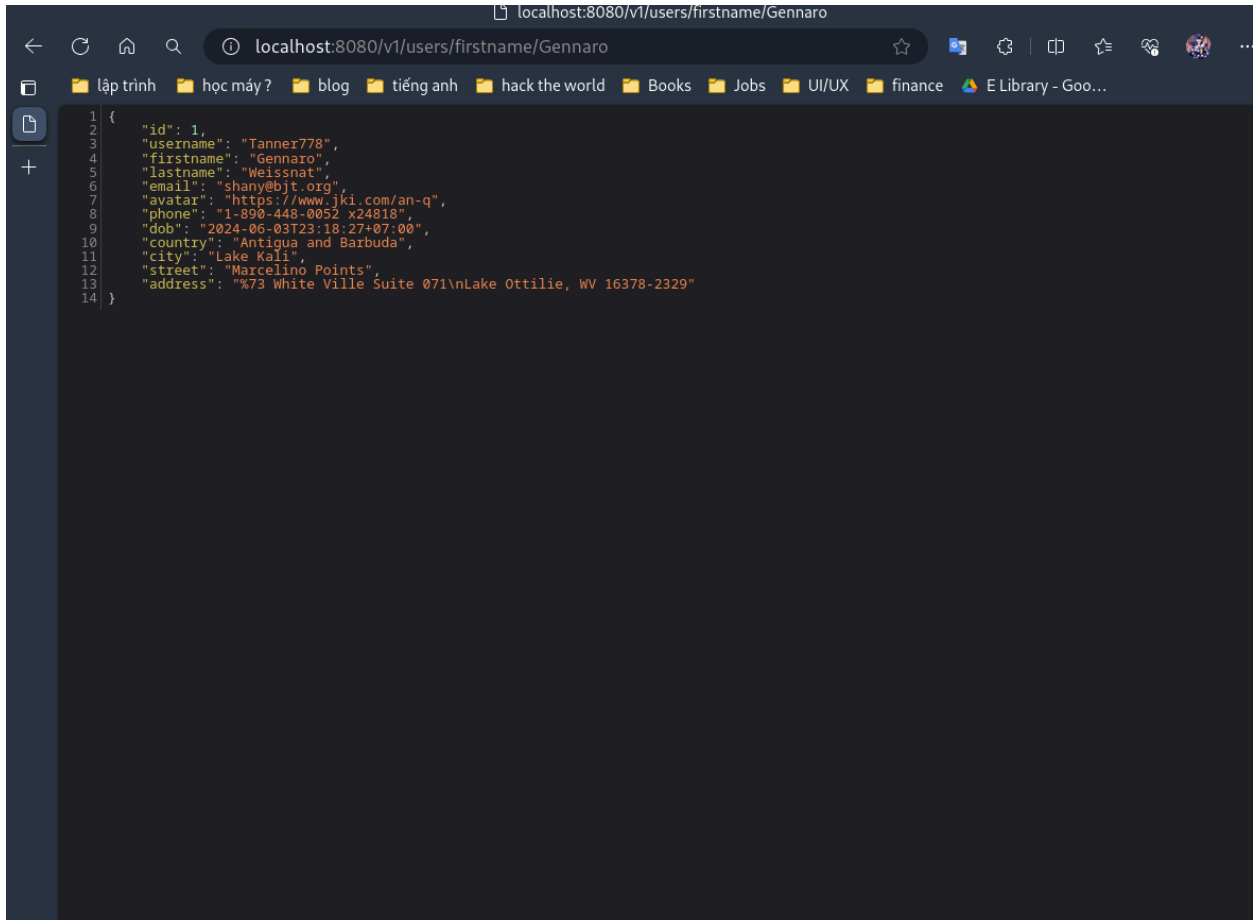


A screenshot of a web browser window displaying a JSON response from the API endpoint `localhost:8080/v1/users/2`. The browser's address bar shows the URL. Below the address bar, a dark-themed code editor displays the JSON data. The JSON object contains user information for ID 2, including username, first and last names, email, avatar, phone, date of birth, country, city, street, and a full address.

```
1 {
2   "id": 2,
3   "username": "Tanner460",
4   "firstname": "Gennaro",
5   "lastname": "Weissnat",
6   "email": "shany@bjt.org",
7   "avatar": "https://www.jki.com/an-q",
8   "phone": "1-890-448-0052 x24818",
9   "dob": "2024-06-03T23:18:27+07:00",
10  "country": "Antigua and Barbuda",
11  "city": "Lake Kali",
12  "street": "Marcelino Points",
13  "address": "%73 White Ville Suite 071\\nLake Ottilie, WV 16378-2329"
14 }
```

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

- Find user base on username

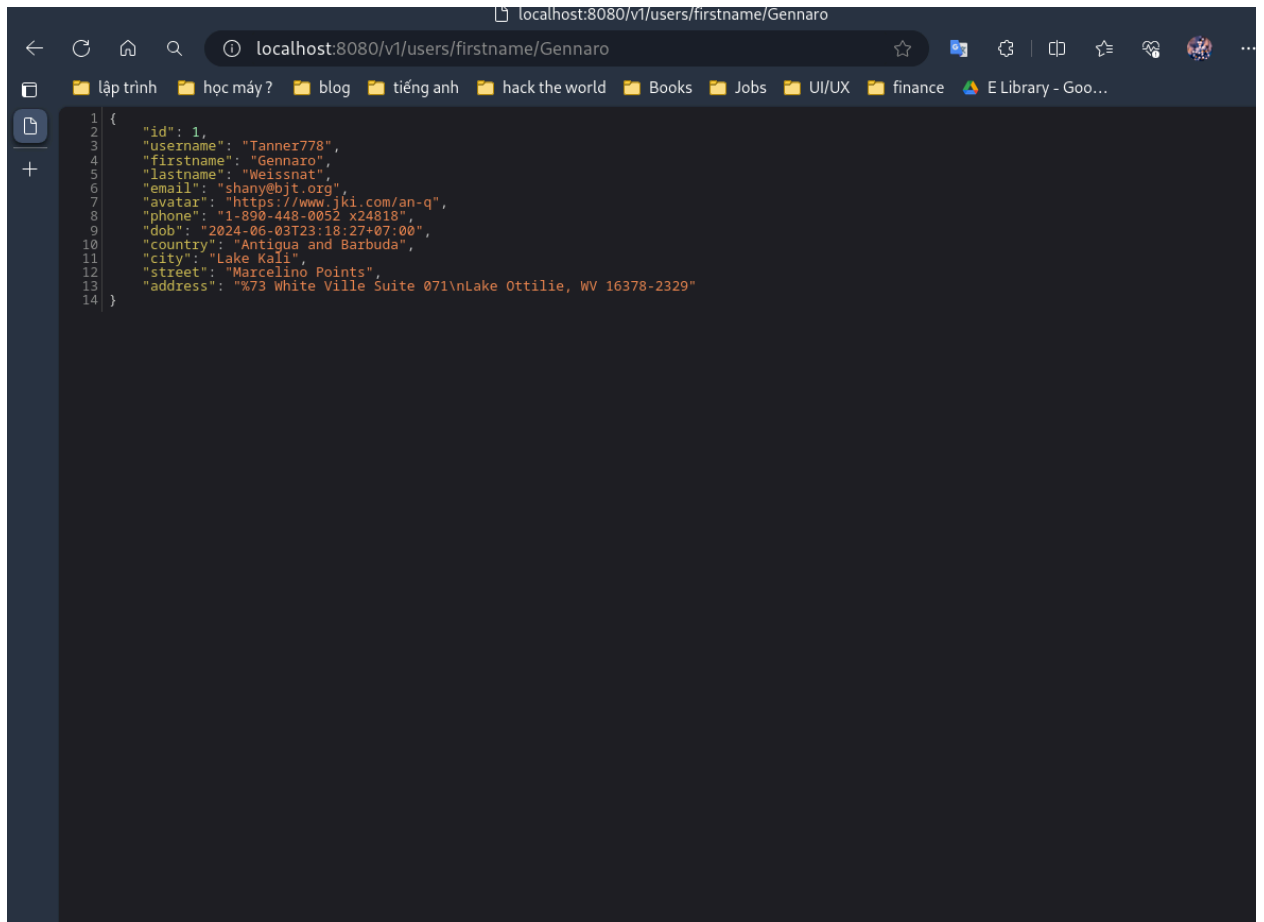


The screenshot shows a web browser window with the address bar displaying `localhost:8080/v1/users/firstname/Gennaro`. The browser's address bar and tabs are visible at the top. Below the browser window, a REST client interface displays a JSON response for the request. The JSON object contains the following fields:

```
1 {
2   "id": 1,
3   "username": "Tanner778",
4   "firstname": "Gennaro",
5   "lastname": "Weissnat",
6   "email": "shany@bjt.org",
7   "avatar": "https://www.jki.com/an-q",
8   "phone": "1-890-448-0052 x24818",
9   "dob": "2024-06-03T23:18:27+07:00",
10  "country": "Antigua and Barbuda",
11  "city": "Lake Kali",
12  "street": "Marcelino Points",
13  "address": "%73 White Ville Suite 071\\nLake Ottilie, WV 16378-2329"
14 }
```

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

- Get firstname

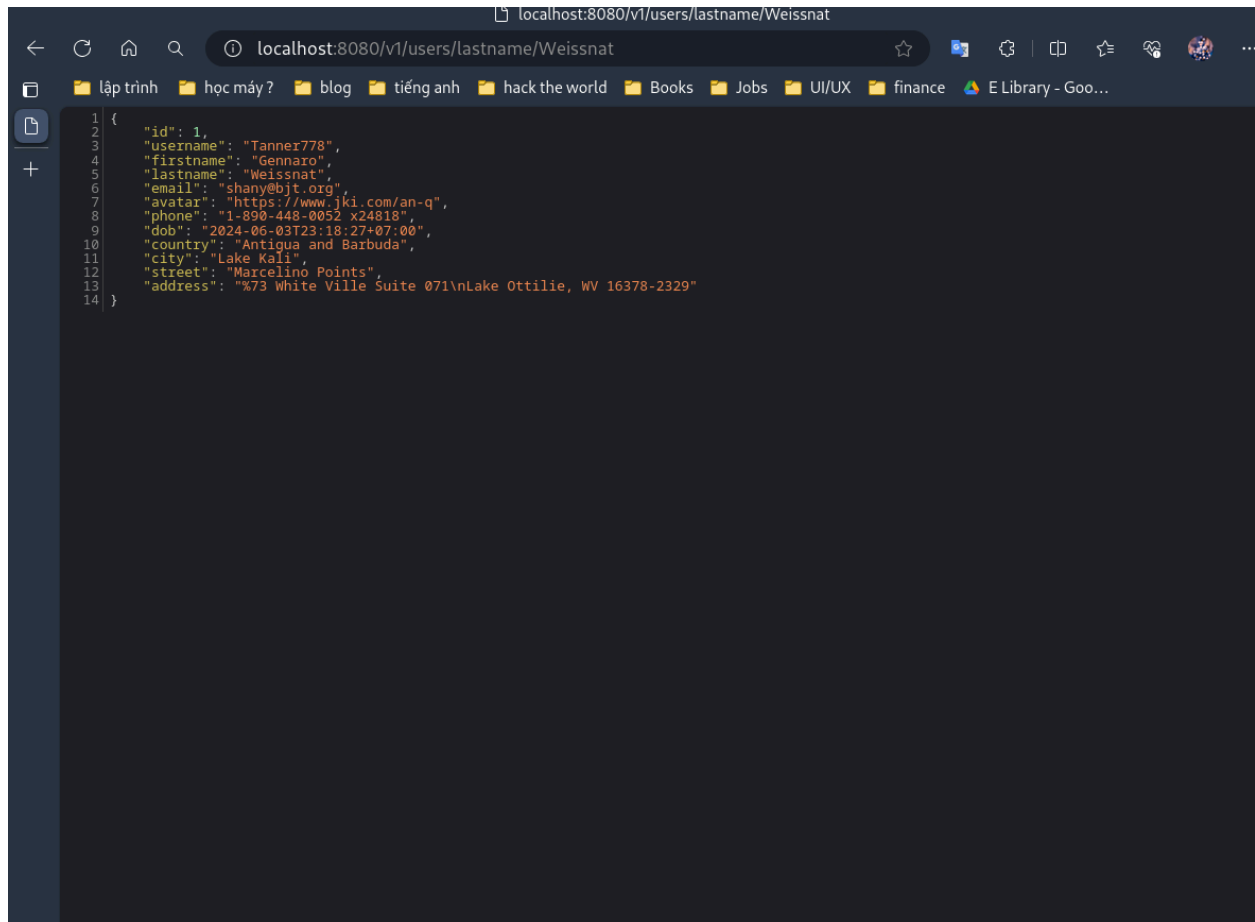


The screenshot shows a web browser window with the address bar displaying `localhost:8080/v1/users/firstname/Gennaro`. The browser's address bar also shows a search icon and a list of bookmarks including "lập trình", "học máy?", "blog", "tiếng anh", "hack the world", "Books", "Jobs", "UI/UX", "finance", and "E Library - Goo...". The main content area of the browser displays a JSON response from a REST client, with line numbers 1 through 14 visible on the left. The JSON object contains the following fields:

```
1 {
2   "id": 1,
3   "username": "Tanner778",
4   "firstname": "Gennaro",
5   "lastname": "Weissnat",
6   "email": "shany@bjt.org",
7   "avatar": "https://www.jki.com/an-q",
8   "phone": "1-890-448-0052 x24818",
9   "dob": "2024-06-03T23:18:27+07:00",
10  "country": "Antigua and Barbuda",
11  "city": "Lake Kali",
12  "street": "Marcelinó Points",
13  "address": "%73 White Ville Suite 071\\nLake Otilie, WV 16378-2329"
14 }
```

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

- Find last name



The screenshot shows a web browser window with the address bar displaying `localhost:8080/v1/users/lastname/Weissnat`. The browser's address bar also shows a search icon and a list of bookmarks including "lập trình", "học máy?", "blog", "tiếng anh", "hack the world", "Books", "Jobs", "UI/UX", "finance", and "E Library - Goo...". The main content area of the browser displays a JSON response from the API, which is a user object. The JSON is formatted with line numbers 1 through 14 on the left side of the editor. The JSON data includes fields for id, username, firstname, lastname, email, avatar, phone, dob, country, city, street, and address.

```
1 {  
2   "id": 1,  
3   "username": "Tanner778",  
4   "firstname": "Gennaro",  
5   "lastname": "Weissnat",  
6   "email": "shany@bjt.org",  
7   "avatar": "https://www.jki.com/an-q",  
8   "phone": "1-890-448-0052 x24818",  
9   "dob": "2024-06-03T23:18:27+07:00",  
10  "country": "Antigua and Barbuda",  
11  "city": "Lake Kali",  
12  "street": "Marcelinó Points",  
13  "address": "%73 White Ville Suite 071\\nLake Otilie, WV 16378-2329"  
14 }
```

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

