2021

Go Microservices 1

COURSE LOGGER APPLICATION

KEITH ANG

Components Implemented

Overview

- Login Authentication REST API
    - Multiplexer setup

```
func main() {
    router := mux.NewRouter()
    router.HandleFunc("/users/v1/", users)
    router.HandleFunc("/users/v1/{username}/{admin}", login).Methods("GET", "PUT", "POST", "DELETE")
    router.HandleFunc("/keys/v1/{accesskey}", validate).Methods("GET")

    fmt.Println("Listening at port 2000")
    log.Fatal(http.ListenAndServe(":2000", router))
}
```

    -
    - Calling HTTP Methods on resources to validate logins, create logins, and provision / revoke access keys

```
if r.Method == "GET" { // Authenticate user login
    fmt.Println("Login Authentication (non-admin) called")
    usernameInput, ok := params["username"]
    if !ok { // No username input
        w.WriteHeader(http.StatusUnprocessableEntity)
        w.Write([]byte("422 - Username and/or password is invalid or blank."))
        return
    }

    pwCookie, err := r.Cookie(passwordHeader)
    if err != nil { // No password input
        w.WriteHeader(http.StatusUnprocessableEntity)
        w.Write([]byte("422 - Username and/or password is invalid or blank."))
        return
    }
    pwInput := pwCookie.Value
    ok, user := getUser(db, usernameInput, pwInput, admin)

    if !ok {
        // User not found
        w.WriteHeader(http.StatusUnprocessableEntity)
        w.Write([]byte("422 - Username and/or password is invalid or blank"))
        return
    } else {
        // Login success, return response with username and accesskey
        usernameCookie := &http.Cookie{
            Name:  usernameHeader,
            Value: user.Username}
        accessKeyCookie := &http.Cookie{
            Name:  accessKeyHeader,
            Value: user.AccessKey}
        http.SetCookie(w, usernameCookie)
        http.SetCookie(w, accessKeyCookie)
        w.WriteHeader(http.StatusOK)
        w.Write([]byte("200 - User Authenticated:" + usernameInput))
        fmt.Println("login success")
    }
```

    -

```
if r.Method == "PUT" { // provision/revoke access key
    // Search DB for whether access key exists
    result := db.QueryRow(fmt.Sprintf("SELECT * FROM login_db.login WHERE Username = '%s' LIMIT 1", params["username"]))
    if result.Err() != nil {
        w.WriteHeader(http.StatusNotFound)
        w.Write([]byte("404 - No user found"))
        return
    }

    resultScan := struct {
        Username  string
        Pw        string
        AccessKey string
    }{}

    result.Scan(&resultScan.Username, &resultScan.Pw, &resultScan.AccessKey)
    fmt.Println(resultScan)
    if resultScan.AccessKey == "nil" { // Check how driver handles null values
        //Generate UUID and assign as key. Print Status and outcome (assigned).
        accessKey := uuid.NewV4()

        edituserKey(db, params["username"], accessKey.String())
        w.WriteHeader(http.StatusAccepted)
        w.Write([]byte("202 - Access Key provisioned: " + params["username"]))
    } else {
        // Remove Key. Print Status and outcome (revoked)
        edituserKey(db, params["username"], "nil")
        w.WriteHeader(http.StatusAccepted)
        w.Write([]byte("202 - Access Key revoked: " + params["username"]))
    }
}
```

- Course Management REST API
  - Multiplexer setup

```
func main() {
    defer db.Close()

    router := mux.NewRouter()
    router.HandleFunc("/CMS/v1/", home)
    router.HandleFunc("/CMS/v1/courses", allcourses)
    router.HandleFunc("/CMS/v1/courses/{courseid}", course).Methods("GET", "PUT", "POST", "DELETE")

    fmt.Println("Listening at port 5000")
    log.Fatal(http.ListenAndServe(":5000", router))
}
```

  - Handler function to return all courses or courses following a search criteria, depending on query strings

```go
func allcourses(w http.ResponseWriter, r *http.Request) {
    kv := r.URL.Query()
    key := kv[accessKeyHeader][0]
    if !validate(key) {
        return
    }
    // returns the key/value pairs in the query string as a map object

    if len(kv) > 1 {
        subsetted := make(map[string]CourseInfo)
        for k, v := range kv {
            for code, course := range courses {
                switch k {
                case "Title":
                    if strings.Contains(course.Title, v[0]) {
                        subsetted[code] = course
                    }
                case "Instructor":
                    if strings.Contains(course.Instructor, v[0]) {
                        subsetted[code] = course
                    }
                case "School":
                    if strings.Contains(course.School, v[0]) {
                        subsetted[code] = course
                    }
                }
            }
        }
        w.WriteHeader(http.StatusOK)
        json.NewEncoder(w).Encode(subsetted)
    } else { // No search criteria received, return all courses
        w.WriteHeader(http.StatusOK)
        json.NewEncoder(w).Encode(courses)
    }
}
```

- Access Key validation from courseService by calling loginService

```go
func validate(accessKey string) bool {
    if accessKey == "nil" {
        fmt.Println("No valid Access Key found.")
        return false
    }
    validationURL := "http://localhost:2000/keys/v1/" + accessKey
    response, err := http.Get(validationURL)
    if err != nil {
        fmt.Printf("The HTTP request failed with error %s\n", err)
        return false
    } else {
        defer response.Body.Close()
        raw, _ := ioutil.ReadAll(response.Body)
        var result struct {
            Validated bool
        }
        json.Unmarshal(raw, &result)
        if result.Validated {
            fmt.Println("Access Key confirmed:", accessKey)
        } else {
            fmt.Println("No valid Access Key found.")
        }
        return result.Validated
    }
}
```

- Admin Client for Access Key Provisioning/Revoking and User Account Management
    - Connects to loginService API to manage which accounts will have access key needed to access courseService

```
------------------------------
Admin Login Management
------------------------------
[ 0 ] : Provision / Revoke Access Key
[ 1 ] : Delete User Account
Please input your option (0 to 1)
Enter -9 to exit/terminate.
0
Selected [ 0 ] :  Provision / Revoke Access Key

Existing usernames:
john
test
user1
Please input the username to provide/revoke an access key:
```

- User Client for Accessing Modifying Course records
    - CRUD implementation by calling HTTP methods on resources (course codes)

```
Selected [ 0 ] :  View All Courses

A221 - Emotion & Politics || Taught by: Neil M. || Faculty: Yale-NUS College
A113 - Perception || Taught by: Neil M. || Faculty: Philosophy
Access Key confirmed.

------------------------------
Course Management Service
------------------------------
[ 0 ] : View All Courses
[ 1 ] : Access Specific Course
[ 2 ] : Filter Courses by Criteria
[ 3 ] : Add New Course
[ 4 ] : Edit Existing Course
[ 5 ] : Delete Existing Course
Please input your option (0 to 5)
Enter -9 to exit/terminate.
```

Setup Guide

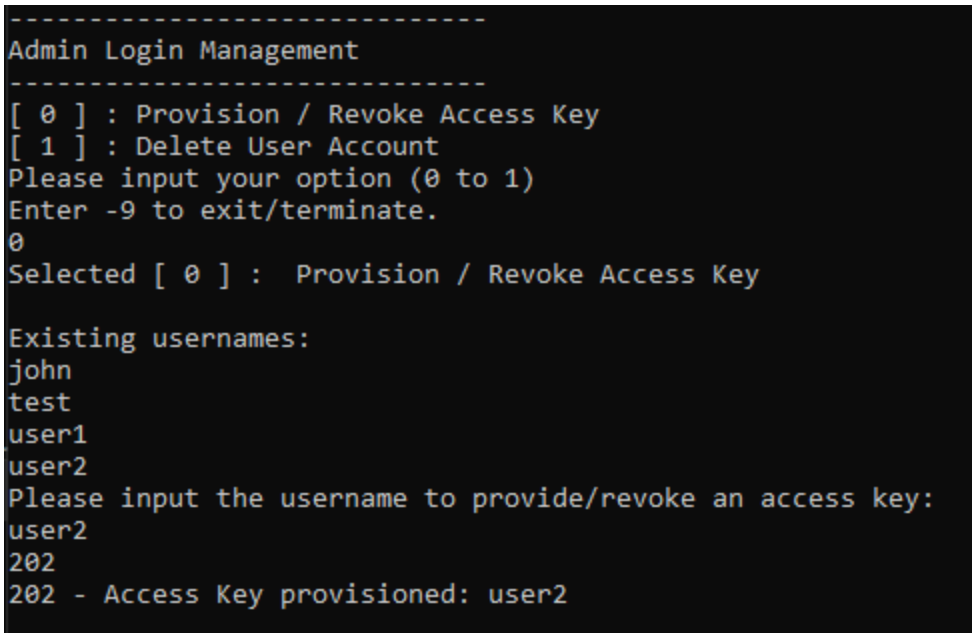
1) Database Setup
    a. Set MySQL root account password to 'veg-kluh!PRIW3hirt'
    b. Run included file, 'db_setup.sql' in MySQL Workbench on localhost (port 3306)
2) API Setup
    a. Open file 'courseService.exe' in the courseService folder
    b. Open file 'loginService.exe' in the loginService folder

Test Guide

1) Open client.exe and create a new account
    a.
```
------------------------------
Course Management Service
------------------------------
[ 0 ] : Login to Existing Account
[ 1 ] : Create New Account
Please input your option (0 to 1)
Enter -9 to exit/terminate.
1
Selected [ 1 ] :  Create New Account

New Account
Please input your username:
user2
Please input your password:
user2
201
201 - Account added : user2
No valid Access Key found.
```
2) Open adminClient.exe and provision an access key to that account

a.



3) Login on client.exe and interact with the CRUD operations for course data shown

a.