



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ

КАФЕДРА КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)

НАПРАВЛЕНИЕ ПОДГОТОВКИ 09.03.01 Компьютерные системы и сети

О Т Ч Е Т

по лабораторной работе № 11

Название: Аутентификация пользователей с помощью jwt-токена

Дисциплина: Языки Интернет-Программирования

Студент

ИУ6-31Б

(Группа)

(Подпись, дата)

К.Д. Коротаев

(И.О. Фамилия)

Преподаватель

(Подпись, дата)

И.О. Фамилия

(И.О. Фамилия)

Москва, 2024

Цель работы: получение первичных знаний в области авторизации и аутентификации в контексте веб-приложений

Задание:

1. Перекопировать код, полученный в ходе выполнения предыдущей лабораторной работы
2. Ознакомиться с теоретическими сведениями
3. Реализовать сервис Auth (регистрация пользователя и авторизация с выдачей jwt-токена)
4. Добавить в сервисы count, hello и query валидацию jwt-токена (если токен не валиден или отсутствует — возвращаем код 401)
5. Сделать отчёт и поместить его в директорию docs
6. Защитить лабораторную работу

Ход работы:

Код для сервиса Auth:

main.go

```
package main
```

```
import (  
    "net/http"  
    "time"
```

```
  
    "github.com/golang-jwt/jwt/v5"  
    echojwt "github.com/labstack/echo-jwt/v4"  
    "github.com/labstack/echo/v4"  
    "github.com/labstack/echo/v4/middleware"
```

```
)
```

```
// jwtCustomClaims are custom claims extending default ones.
```

```
// See https://github.com/golang-jwt/jwt for more examples
```

```
type jwtCustomClaims struct {  
    Name string `json:"name"`  
    Admin bool  `json:"admin"`  
    jwt.RegisteredClaims  
}
```

```
func login(c echo.Context) error {  
    username := c.FormValue("username")  
    password := c.FormValue("password")
```

```
  
    // Throws unauthorized error  
    if username != "admin" || password != "admin" {  
        return echo.ErrUnauthorized  
    }
```

```
  
    // Set custom claims  
    claims := &jwtCustomClaims{  
        "admin",  
        true,  
        jwt.RegisteredClaims{  
            ExpiresAt: jwt.NewNumericDate(time.Now().Add(time.Hour * 72)),  
        },  
    }
```

```
  
    // Create token with claims  
    token := jwt.NewWithClaims(jwt.SigningMethodHS256, claims)
```

```
  
    // Generate encoded token and send it as response.  
    t, err := token.SignedString([]byte("secret"))  
    if err != nil {
```

```

        return err
    }

    return c.JSON(http.StatusOK, echo.Map{
        "token": t,
    })
}

func accessible(c echo.Context) error {
    return c.String(http.StatusOK, "Accessible")
}

func restricted(c echo.Context) error {
    user := c.Get("user").(*jwt.Token)
    claims := user.Claims.(*jwtCustomClaims)
    name := claims.Name
    return c.String(http.StatusOK, "Welcome "+name+"!")
}

func main() {
    e := echo.New()

    // Middleware
    e.Use(middleware.Logger())
    e.Use(middleware.Recover())

    // Login route
    e.POST("/login", login)

    // Unauthenticated route
    e.GET("/", accessible)

    // Restricted group
    r := e.Group("/restricted")

    // Configure middleware with the custom claims type
    config := echojwt.Config{
        NewClaimsFunc: func(c echo.Context) jwt.Claims {
            return new(jwtCustomClaims)
        },
        SigningKey: []byte("secret"),
    }
    r.Use(echojwt.WithConfig(config))
    r.GET("", restricted)
    e.Logger.Fatal(e.Start(":1323"))
}

```

```

auth.go
package auth
import (
    "net/http"
    "time"
    "github.com/golang-jwt/jwt/v5"
    echojwt "github.com/labstack/echo-jwt/v4"
    "github.com/labstack/echo/v4"
    "github.com/labstack/echo/v4/middleware"
)
// jwtCustomClaims are custom claims extending default ones.
// See https://github.com/golang-jwt/jwt for more examples
type jwtCustomClaims struct {
    Name string `json:"name"`
    Admin bool `json:"admin"`
    jwt.RegisteredClaims
}
func login(c echo.Context) error {
    username := c.FormValue("username")
    password := c.FormValue("password")

    // Throws unauthorized error
    if username != "admin" || password != "admin" {
        return echo.ErrUnauthorized
    }

    // Set custom claims
    claims := &jwtCustomClaims{
        "admin",
        true,
        jwt.RegisteredClaims{
            ExpiresAt: jwt.NewNumericDate(time.Now().Add(time.Hour * 72)),
        },
    }

    // Create token with claims
    token := jwt.NewWithClaims(jwt.SigningMethodHS256, claims)
    // Generate encoded token and send it as response.
    t, err := token.SignedString([]byte("secret"))
    if err != nil {
        return err
    }

    return c.JSON(http.StatusOK, echo.Map{
        "token": t,
    })
}

```

```

}

func accessible(c echo.Context) error {
    return c.String(http.StatusOK, "Accessible")
}

func restricted(c echo.Context) error {
    user := c.Get("user").(*jwt.Token)
    claims := user.Claims.(*jwtCustomClaims)
    name := claims.Name
    return c.String(http.StatusOK, "Welcome "+name+"!")
}

func CheckToken(c echo.Context) error {
    user := c.Get("user").(*jwt.Token)
    claims := user.Claims.(*jwtCustomClaims)
    if !claims.Admin {
        return echo.ErrUnauthorized
    }
    return nil
}

func main() {
    e := echo.New()
    // Middleware
    e.Use(middleware.Logger())
    e.Use(middleware.Recover())

    // Login route
    e.POST("/login", login)

    // Unauthenticated route
    e.GET("/", accessible)

    // Restricted group
    r := e.Group("/restricted")

    // Configure middleware with the custom claims type
    config := echojwt.Config{
        NewClaimsFunc: func(c echo.Context) jwt.Claims {
            return new(jwtCustomClaims)
        },
        SigningKey: []byte("secret"),
    }
    r.Use(echojwt.WithConfig(config))
    r.GET("", restricted)
    e.Logger.Fatal(e.Start(":1323"))
}

```

Код сервисов идентичен 10 лабораторной работе, за исключением ниже описанных файлов.

Код для работы с сервисом Count:

```
api.go handler.go
C: > Users > admin > .ssh > web-11 > internal > count > api > api.go > NewServer

1 package api
2 import (
3     "fmt"
4     "github.com/golang-jwt/jwt/v5"
5     echojwt "github.com/labstack/echo-jwt/v4"
6     "github.com/labstack/echo/v4"
7     "github.com/labstack/echo/v4/middleware")
8 type Server struct {
9     maxSize int
10    server *echo.Echo
11    address string
12    uc Usecase}
13 type jwtCustomClaims struct {
14     Name string `json:"name"`
15     Admin bool `json:"admin"`
16     jwt.RegisteredClaims}
17 func NewServer(ip string, port int, maxSize int, uc Usecase) *Server {
18     api := Server{
19         maxSize: maxSize,
20         uc:      uc,
21     }
22     api.server = echo.New()
23     api.server.Use(middleware.Logger())
24     api.server.Use(middleware.Recover())
25     api.server.POST("/login", api.Login)
26     config := echojwt.Config{
27         NewClaimsFunc: func(c echo.Context) jwt.Claims {
28             return new(jwtCustomClaims)
29         },
30         SigningKey: []byte("secret"),
31     }
32     r := api.server.Group("/restricted")
33     r.Use(echojwt.WithConfig(config))
34     r.GET("/count", api.GetCount)
35     r.POST("/count", api.PostCount)
36     api.address = fmt.Sprintf("%s:%d", ip, port)
37     return &api}
38 func (api *Server) Run() {
39     api.server.Logger.Fatal(api.server.Start(api.address))}
```

```

api.go • handler.go •
C: > Users > admin > .ssh > web-11 > internal > count > api > handler.go > (*Server).Login
1 package api
2 import ("errors"
3         "net/http"
4         "time"
5         "github.com/ValeryBMSU/web-11/pkg/vars"
6         "github.com/golang-jwt/jwt/v5"
7         "github.com/labstack/echo/v4")
8 func (srv *Server) Login(e echo.Context) error {
9     username := e.FormValue("username")
10    password := e.FormValue("password")
11    // Throws unauthorized error
12    if username != "admin" || password != "admin" {return echo.ErrUnauthorized}
13    // Set custom claims
14    claims := &jwtCustomClaims{"admin", true, jwt.RegisteredClaims{ExpiresAt: jwt.NewNumericDate(time.Now().Add(time.Hour * 72)),},}
15    // Create token with claims
16    token := jwt.NewWithClaims(jwt.SigningMethodHS256, claims)
17    // Generate encoded token and send it as response.
18    t, err := token.SignedString([]byte("secret"))
19    if err != nil {return err}
20    return e.JSON(http.StatusOK, echo.Map{"token": t,})
21    // GetCount возвращает случайное приветствие пользователю
22    func (srv *Server) GetCount(e echo.Context) error {
23        msg, err := srv.uc.FetchCount()
24        if err != nil {return e.String(http.StatusInternalServerError, err.Error())}
25        return e.JSON(http.StatusOK, msg)}
26    // PostCount Помещает новый вариант приветствия в БД
27    func (srv *Server) PostCount(e echo.Context) error {
28        input := struct {Msg *int `json:"count"}`{}
29        err := e.Bind(&input)
30        if err != nil {return e.String(http.StatusInternalServerError, err.Error())}
31        if input.Msg == nil {return e.String(http.StatusBadRequest, "msg is empty")}
32        err = srv.uc.IncrementCount(*input.Msg)
33        if err != nil {
34            if errors.Is(err, vars.ErrAlreadyExist) {
35                return e.String(http.StatusConflict, err.Error())
36            }
37            return e.String(http.StatusInternalServerError, err.Error())
38        }
39        return e.String(http.StatusCreated, "OK")}

```


Код для работы с сервисом Hello:

```
api.go handler.go
C: > Users > admin > .ssh > web-11 > internal > hello > api > api.go > (*Server).Run

1 package api
2 import ("fmt"
3         "github.com/golang-jwt/jwt/v5"
4         echojwt "github.com/labstack/echo-jwt/v4"
5         "github.com/labstack/echo/v4"
6         "github.com/labstack/echo/v4/middleware")
7 type Server struct {
8     maxSize int
9     server  *echo.Echo
10    address string
11    uc Usecase}
12 type jwtCustomClaims struct {
13     Name string `json:"name"`
14     Admin bool  `json:"admin"`
15     jwt.RegisteredClaims}
16 func NewServer(ip string, port int, maxSize int, uc Usecase) *Server {
17     api := Server{
18         maxSize: maxSize,
19         uc:      uc,}
20     api.server = echo.New()
21     api.server.Use(middleware.Logger())
22     api.server.Use(middleware.Recover())
23     api.server.POST("/login", api.Login)
24     config := echojwt.Config{
25         NewClaimsFunc: func(c echo.Context) jwt.Claims {
26             return new(jwtCustomClaims)
27         },
28         SigningKey: []byte("secret"),
29     }
30     r := api.server.Group("/restricted")
31     r.Use(echojwt.WithConfig(config))
32     r.GET("/hello", api.GetHello)
33     r.POST("/hello", api.PostHello)
34     api.address = fmt.Sprintf("%s:%d", ip, port)
35     return &api
36 }
37 func (api *Server) Run() {
38     api.server.Logger.Fatal(api.server.Start(api.address))}
```

handler.go

C:\Users\admin> .ssh > web-11 > internal > hello > api > handler.go > (*Server).Login

```
1 package api
2 import ("errors"
3         "net/http"
4         "time"
5         "github.com/ValeryBMSTU/web-11/pkg/vars"
6         "github.com/golang-jwt/jwt/v5"
7         "github.com/labstack/echo/v4")
8 func (srv *Server) Login(e echo.Context) error {
9     username := e.FormValue("username")
10    password := e.FormValue("password")
11    // Throws unauthorized error
12    if username != "admin" || password != "admin" {return echo.ErrUnauthorized}
13    // Set custom claims
14    claims := &jwtCustomClaims{"admin", true, jwt.RegisteredClaims{ExpiresAt: jwt.NewNumericDate(time.Now().Add(time.Hour * 72)),},}
15    // Create token with claims
16    token := jwt.NewWithClaims(jwt.SigningMethodHS256, claims)
17    // Generate encoded token and send it as response.
18    t, err := token.SignedString([]byte("secret"))
19    if err != nil {return err}
20    return e.JSON(http.StatusOK, echo.Map{"token": t,})
21 func (srv *Server) GetHello(e echo.Context) error {
22     msg, err := srv.uc.FetchHelloMessage()
23     if err != nil {return e.String(http.StatusInternalServerError, err.Error())}
24     return e.JSON(http.StatusOK, msg)}
25 // PostHello Помещает новый вариант приветствия в БД
26 func (srv *Server) PostHello(e echo.Context) error {
27     input := struct {Msg *string `json:"msg"`}{}
28     err := e.Bind(&input)
29     if err != nil {return e.String(http.StatusInternalServerError, err.Error())}
30     if input.Msg == nil {
31         return e.String(http.StatusBadRequest, "msg is empty")}
32     if len([]rune(*input.Msg)) > srv.maxSize {
33         return e.String(http.StatusBadRequest, "hello message too large")}
34     err = srv.uc.SetHelloMessage(*input.Msg)
35     if err != nil {if errors.Is(err, vars.ErrAlreadyExist) {
36         return e.String(http.StatusConflict, err.Error())}
37         return e.String(http.StatusInternalServerError, err.Error())}
38     return e.String(http.StatusCreated, "OK")}
```

Код для работы с сервисом Query:

```
api.go
C: > Users > admin > .ssh > web-11 > internal > query > api > api.go > NewServer

1 package api
2 import ("fmt"
3         "github.com/golang-jwt/jwt/v5"
4         echojwt "github.com/labstack/echo-jwt/v4"
5         "github.com/labstack/echo/v4")
6 type Server struct {
7     maxSize int
8     server  *echo.Echo
9     address string
10    uc Usecase}
11 type jwtCustomClaims struct {
12     Name  string `json:"name"`
13     Admin bool   `json:"admin"`
14     jwt.RegisteredClaims
15 }
16 func NewServer(ip string, port int, maxSize int, uc Usecase) *Server {
17     api := Server{
18         maxSize: maxSize,
19         uc:      uc,
20     }
21     api.server = echo.New()
22     config := echojwt.Config{
23         NewClaimsFunc: func(c echo.Context) jwt.Claims {
24             return new(jwtCustomClaims)
25         },
26         SigningKey: []byte("secret"),
27     }
28     r := api.server.Group("/restricted")
29     r.Use(echojwt.WithConfig(config))
30     r.GET("/query", api.GetQuery)
31     r.POST("/query", api.PostQuery)
32     api.address = fmt.Sprintf("%s:%d", ip, port)
33     return &api
34 }
35 func (api *Server) Run() {
36     api.server.Logger.Fatal(api.server.Start(api.address))
37 }
```

handler.go

C:\Users> admin > .ssh > web-11 > internal > query > api > handler.go > ...

```
1 package api
2 import ("net/http"
3         "time"
4         "github.com/golang-jwt/jwt/v5"
5         "github.com/labstack/echo/v4")
6 func (srv *Server) Login(e echo.Context) error {
7     username := e.FormValue("username")
8     password := e.FormValue("password")
9     // Throws unauthorized error
10    if username != "admin" || password != "admin" {return echo.ErrUnauthorized}
11    // Set custom claims
12    claims := &jwtCustomClaims{"admin",true,jwt.RegisteredClaims{ExpiresAt: jwt.NewNumericDate(time.Now().Add(time.Hour * 72)),},}
13    // Create token with claims
14    token := jwt.NewWithClaims(jwt.SigningMethodHS256, claims)
15    // Generate encoded token and send it as response.
16    t, err := token.SignedString([]byte("secret"))
17    if err != nil {
18        return err
19    }
20    return e.JSON(http.StatusOK, echo.Map{"token": t,})
21 func (srv *Server) GetQuery(e echo.Context) error {
22     // get query name
23     name := e.QueryParam("name")
24     msg, err := srv.uc.FetchQuery(name)
25     if err != nil {
26         return e.String(http.StatusInternalServerError, err.Error())
27     }
28     return e.JSON(http.StatusOK, msg)}
29 func (srv *Server) PostQuery(e echo.Context) error {
30     // post query name
31     name := e.QueryParam("name")
32     err := srv.uc.InsertQuery(name)
33     if err != nil {
34         return e.String(http.StatusInternalServerError, err.Error())
35     }
36     return e.NoContent(http.StatusOK)
37 }
```

Тестирование к сервису Auth:

Получение токена

```
PS C:\Users\admin> $username = "admin"
PS C:\Users\admin> $password = "admin"
PS C:\Users\admin> $url = "http://localhost:1323/login"
PS C:\Users\admin> $response = Invoke-RestMethod -Uri $url -Method Post -Body @{username=$username; password=$password}
PS C:\Users\admin> $response.token
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1bm90cmVudCI6ImlhbnQ1IiwiaWF0IjoiMTYzOTQ1MzcxV1iOnza66OwqgyYmJ4BzU2bGJlCmllg
```

Тестирование к сервису Hello:

```
PS C:\Users\admin> Invoke-RestMethod -Uri "http://127.0.0.1:8081/restricted/hello" -Method Get -Headers @{Authorization = "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1bm90cmVudCI6ImlhbnQ1IiwiaWF0IjoiMTYzOTQ1MzcxV1iOnza66OwqgyYmJ4BzU2bGJlCmllg"}
Hello, negr!
PS C:\Users\admin> $Body = @{msg="Hello, aboba!"} | ConvertTo-Json
PS C:\Users\admin> $response = Invoke-RestMethod -Uri "http://127.0.0.1:8081/restricted/hello" -Method Post -Body $Body -ContentType "application/json" -Headers @{Authorization = "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1bm90cmVudCI6ImlhbnQ1IiwiaWF0IjoiMTYzOTQ1MzcxV1iOnza66OwqgyYmJ4BzU2bGJlCmllg"}
PS C:\Users\admin> $response
OK
PS C:\Users\admin> Invoke-RestMethod -Uri "http://127.0.0.1:8081/restricted/hello" -Method Get -Headers @{Authorization = "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1bm90cmVudCI6ImlhbnQ1IiwiaWF0IjoiMTYzOTQ1MzcxV1iOnza66OwqgyYmJ4BzU2bGJlCmllg"}
Hello, negr!
PS C:\Users\admin> Invoke-RestMethod -Uri "http://127.0.0.1:8081/restricted/hello" -Method Get -Headers @{Authorization = "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1bm90cmVudCI6ImlhbnQ1IiwiaWF0IjoiMTYzOTQ1MzcxV1iOnza66OwqgyYmJ4BzU2bGJlCmllg"}
Hello, aboba!
PS C:\Users\admin> Invoke-RestMethod -Uri "http://127.0.0.1:8081/restricted/hello" -Method Get -Headers @{Authorization = "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1bm90cmVudCI6ImlhbnQ1IiwiaWF0IjoiMTYzOTQ1MzcxV1iOnza66OwqgyYmJ4BzU2bGJlCmllg"}
Hello, World!
```

Тестирование к сервису Query:

```
PS C:\Users\admin> Invoke-RestMethod -Uri "http://127.0.0.1:8083/restricted/query?name=afawe" -Method Get -Headers @{Authorization = "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1bm90cmVudCI6ImlhbnQ1IiwiaWF0IjoiMTYzOTQ1MzcxV1iOnza66OwqgyYmJ4BzU2bGJlCmllg"}
Hello, afawe!
PS C:\Users\admin> Invoke-RestMethod -Uri "http://127.0.0.1:8083/restricted/query?name=afawe2" -Method Post -Headers @{Authorization = "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1bm90cmVudCI6ImlhbnQ1IiwiaWF0IjoiMTYzOTQ1MzcxV1iOnza66OwqgyYmJ4BzU2bGJlCmllg"}
PS C:\Users\admin> Invoke-RestMethod -Uri "http://127.0.0.1:8083/restricted/query?name=afawe2" -Method Get -Headers @{Authorization = "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1bm90cmVudCI6ImlhbnQ1IiwiaWF0IjoiMTYzOTQ1MzcxV1iOnza66OwqgyYmJ4BzU2bGJlCmllg"}
Hello, afawe2!
```

Тестирование к сервису Count:

```
PS C:\Users\admin> Invoke-RestMethod -Uri "http://127.0.0.1:8082/restricted/count" -Method Post -Body $Body -ContentType "application/json" -Headers @{Authorization = "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1bm90cmVudCI6ImlhbnQ1IiwiaWF0IjoiMTYzOTQ1MzcxV1iOnza66OwqgyYmJ4BzU2bGJlCmllg"}
OK
PS C:\Users\admin> Invoke-RestMethod -Uri "http://127.0.0.1:8082/restricted/count" -Method Get -Headers @{Authorization = "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1bm90cmVudCI6ImlhbnQ1IiwiaWF0IjoiMTYzOTQ1MzcxV1iOnza66OwqgyYmJ4BzU2bGJlCmllg"}
44
PS C:\Users\admin> $Body = @{count=14} | ConvertTo-Json
PS C:\Users\admin> Invoke-RestMethod -Uri "http://127.0.0.1:8082/restricted/count" -Method Post -Body $Body -ContentType "application/json" -Headers @{Authorization = "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1bm90cmVudCI6ImlhbnQ1IiwiaWF0IjoiMTYzOTQ1MzcxV1iOnza66OwqgyYmJ4BzU2bGJlCmllg"}
OK
PS C:\Users\admin> Invoke-RestMethod -Uri "http://127.0.0.1:8082/restricted/count" -Method Get -Headers @{Authorization = "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1bm90cmVudCI6ImlhbnQ1IiwiaWF0IjoiMTYzOTQ1MzcxV1iOnza66OwqgyYmJ4BzU2bGJlCmllg"}
58
```

Заключение: получены первичные знания в области авторизации и аутентификации в контексте веб-приложений