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 _____ К.Д. Коротаев
(Группа) (Подпись, дата) (И.О. Фамилия)

Преподаватель _____ И.О. Фамилия
(Подпись, дата) (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 валидент 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

```go
package api
import (
    "fmt"
    "github.com/golang-jwt/jwt/v5"
    echojwt "github.com/labstack/echo-jwt/v4"
    "github.com/labstack/echo/v4"
    "github.com/labstack/echo/v4/middleware")
type Server struct {
    maxSize int
    server  *echo.Echo
    address string
    uc Usecase}
type jwtCustomClaims struct {
    Name  string `json:"name"`
    Admin bool   `json:"admin"`
    jwt.RegisteredClaims}
func NewServer(ip string, port int, maxSize int, uc Usecase) *Server {
    api := Server{
        maxSize: maxSize,
        uc:      uc,
    }
    api.server = echo.New()
    api.server.Use(middleware.Logger())
    api.server.Use(middleware.Recover())
    api.server.POST("/login", api.Login)
    config := echojwt.Config{
        NewClaimsFunc: func(c echo.Context) jwt.Claims {
            return new(jwtCustomClaims)
        },
        SigningKey: []byte("secret"),}
    r := api.server.Group("/restricted")
    r.Use(echojwt.WithConfig(config))
    r.GET("/count", api.GetCount)
    r.POST("/count", api.PostCount)
    api.address = fmt.Sprintf("%s:%d", ip, port)
    return &api}
func (api *Server) Run() {
    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

```go
package api
import ("errors"
    "net/http"
    "time"
    "github.com/ValeryBMSTU/web-11/pkg/vars"
    "github.com/golang-jwt/jwt/v5"
    "github.com/labstack/echo/v4")
func (srv *Server) Login(e echo.Context) error {
    username := e.FormValue("username")
    password := e.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 e.JSON(http.StatusOK, echo.Map{"token": t,})}
// GetCount возвращает случайное приветствие пользователю
func (srv *Server) GetCount(e echo.Context) error {
    msg, err := srv.uc.FetchCount()
    if err != nil {return e.String(http.StatusInternalServerError, err.Error())}
    return e.JSON(http.StatusOK, msg)}
// PostCount Помещает новый вариант приветствия в БД
func (srv *Server) PostCount(e echo.Context) error {
    input := struct {Msg *int `json:"count"`}{}
    err := e.Bind(&input)
    if err != nil {return e.String(http.StatusInternalServerError, err.Error())}
    if input.Msg == nil {return e.String(http.StatusBadRequest, "msg is empty")}
    err = srv.uc.IncrementCount(*input.Msg)
    if err != nil {
        if errors.Is(err, vars.ErrAlreadyExist) {
            return e.String(http.StatusConflict, err.Error())}
        return e.String(http.StatusInternalServerError, err.Error())}
    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

```go
package api
import ("fmt"
    "github.com/golang-jwt/jwt/v5"
    echojwt "github.com/labstack/echo-jwt/v4"
    "github.com/labstack/echo/v4"
    "github.com/labstack/echo/v4/middleware")
type Server struct {
    maxSize int
    server  *echo.Echo
    address string
    uc Usecase}
type jwtCustomClaims struct {
    Name  string `json:"name"`
    Admin bool   `json:"admin"`
    jwt.RegisteredClaims}
func NewServer(ip string, port int, maxSize int, uc Usecase) *Server {
    api := Server{
        maxSize: maxSize,
        uc:      uc,}
    api.server = echo.New()
    api.server.Use(middleware.Logger())
    api.server.Use(middleware.Recover())
    api.server.POST("/login", api.Login)
    config := echojwt.Config{
        NewClaimsFunc: func(c echo.Context) jwt.Claims {
            return new(jwtCustomClaims)
        },
        SigningKey: []byte("secret"),
    }
    r := api.server.Group("/restricted")
    r.Use(echojwt.WithConfig(config))
    r.GET("/hello", api.GetHello)
    r.POST("/hello", api.PostHello)
    api.address = fmt.Sprintf("%s:%d", ip, port)
    return &api
}
func (api *Server) Run() {
    api.server.Logger.Fatal(api.server.Start(api.address))}
```

handler.go

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

```go
package api
import ("errors"
    "net/http"
    "time"
    "github.com/ValeryBMSTU/web-11/pkg/vars"
    "github.com/golang-jwt/jwt/v5"
    "github.com/labstack/echo/v4")
func (srv *Server) Login(e echo.Context) error {
    username := e.FormValue("username")
    password := e.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 e.JSON(http.StatusOK, echo.Map{"token": t,})}
func (srv *Server) GetHello(e echo.Context) error {
    msg, err := srv.uc.FetchHelloMessage()
    if err != nil {return e.String(http.StatusInternalServerError, err.Error())}
    return e.JSON(http.StatusOK, msg)}
// PostHello Помещает новый вариант приветствия в БД
func (srv *Server) PostHello(e echo.Context) error {
    input := struct {Msg *string `json:"msg"`}{}
    err := e.Bind(&input)
    if err != nil {return e.String(http.StatusInternalServerError, err.Error())}
    if input.Msg == nil {
        return e.String(http.StatusBadRequest, "msg is empty")}
    if len([]rune(*input.Msg)) > srv.maxSize {
        return e.String(http.StatusBadRequest, "hello message too large")}
    err = srv.uc.SetHelloMessage(*input.Msg)
    if err != nil {if errors.Is(err, vars.ErrAlreadyExist) {
            return e.String(http.StatusConflict, err.Error())}
        return e.String(http.StatusInternalServerError, err.Error())}
    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

```go
package api
import ("fmt"
    "github.com/golang-jwt/jwt/v5"
    echojwt "github.com/labstack/echo-jwt/v4"
    "github.com/labstack/echo/v4")
type Server struct {
    maxSize int
    server  *echo.Echo
    address string
    uc Usecase}
type jwtCustomClaims struct {
    Name  string `json:"name"`
    Admin bool   `json:"admin"`
    jwt.RegisteredClaims
}
func NewServer(ip string, port int, maxSize int, uc Usecase) *Server {
    api := Server{
        maxSize: maxSize,
        uc:      uc,}
    api.server = echo.New()
    config := echojwt.Config{
        NewClaimsFunc: func(c echo.Context) jwt.Claims {
            return new(jwtCustomClaims)
        },
        SigningKey: []byte("secret"),
    }
    r := api.server.Group("/restricted")
    r.Use(echojwt.WithConfig(config))
    r.GET("/query", api.GetQuery)
    r.POST("/query", api.PostQuery)
    api.address = fmt.Sprintf("%s:%d", ip, port)
    return &api
}
func (api *Server) Run() {
    api.server.Logger.Fatal(api.server.Start(api.address))
}
```

handler.go

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

```go
package api
import ("net/http"
    "time"
    "github.com/golang-jwt/jwt/v5"
    "github.com/labstack/echo/v4")
func (srv *Server) Login(e echo.Context) error {
    username := e.FormValue("username")
    password := e.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 e.JSON(http.StatusOK, echo.Map{"token": t,})}
func (srv *Server) GetQuery(e echo.Context) error {
    // get query name
    name := e.QueryParam("name")
    msg, err := srv.uc.FetchQuery(name)
    if err != nil {
        return e.String(http.StatusInternalServerError, err.Error())
    }
    return e.JSON(http.StatusOK, msg)}
func (srv *Server) PostQuery(e echo.Context) error {
    // post query name
    name := e.QueryParam("name")
    err := srv.uc.InsertQuery(name)
    if err != nil {
        return e.String(http.StatusInternalServerError, err.Error())
    }
    return e.NoContent(http.StatusOK)
}
```

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.eyJuYW1lIjoiYWRtaW4iLCJhZG1pbiI6dHJ1ZSwiZXhwIjoxNzMzODcxNDk5fQ.aA0vQd1y3cxV1iOnza66
OwqgyYMJ4BzU2bGJICaCllg
```

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.eyJuYW1lIjoiYWRtaW4iLCJhZG1pbiI6dHJ1ZSwiZXhwIjoxNzMzODcxNDk5fQ.aA0vQd1y3c
xV1iOnza66OwqgyYMJ4BzU2bGJICaCllg"}
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.eyJuYW1lIjoiYWRt
aW4iLCJhZG1pbiI6dHJ1ZSwiZXhwIjoxNzMzODcxNDk5fQ.aA0vQd1y3cxV1iOnza66OwqgyYMJ4BzU2bGJICaCllg"}
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.eyJuYW1lIjoiYWRtaW4iLCJhZG1pbiI6dHJ1ZSwiZXhwIjoxNzMzODcxNDk5fQ.aA0vQd1y3c
xV1iOnza66OwqgyYMJ4BzU2bGJICaCllg"}
Hello, negr!
PS C:\Users\admin> Invoke-RestMethod -Uri "http://127.0.0.1:8081/restricted/hello" -Method Get -Headers @{Authorization
= "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJuYW1lIjoiYWRtaW4iLCJhZG1pbiI6dHJ1ZSwiZXhwIjoxNzMzODcxNDk5fQ.aA0vQd1y3c
xV1iOnza66OwqgyYMJ4BzU2bGJICaCllg"}
Hello, aboba!
PS C:\Users\admin> Invoke-RestMethod -Uri "http://127.0.0.1:8081/restricted/hello" -Method Get -Headers @{Authorization
= "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJuYW1lIjoiYWRtaW4iLCJhZG1pbiI6dHJ1ZSwiZXhwIjoxNzMzODcxNDk5fQ.aA0vQd1y3c
xV1iOnza66OwqgyYMJ4BzU2bGJICaCllg"}
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 @{Aut
horization = "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJuYW1lIjoiYWRtaW4iLCJhZG1pbiI6dHJ1ZSwiZXhwIjoxNzMzODcxNDk5fQ
.aA0vQd1y3cxV1iOnza66OwqgyYMJ4BzU2bGJICaCllg"}
Hello, afawe!
PS C:\Users\admin> Invoke-RestMethod -Uri "http://127.0.0.1:8083/restricted/query?name=afawe2" -Method Post -Headers @{A
uthorization = "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJuYW1lIjoiYWRtaW4iLCJhZG1pbiI6dHJ1ZSwiZXhwIjoxNzMzODcxNDk5
fQ.aA0vQd1y3cxV1iOnza66OwqgyYMJ4BzU2bGJICaCllg"}

PS C:\Users\admin> Invoke-RestMethod -Uri "http://127.0.0.1:8083/restricted/query?name=afawe2" -Method Get -Headers @{Au
thorization = "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJuYW1lIjoiYWRtaW4iLCJhZG1pbiI6dHJ1ZSwiZXhwIjoxNzMzODcxNDk5f
Q.aA0vQd1y3cxV1iOnza66OwqgyYMJ4BzU2bGJICaCllg"}
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.eyJuYW1lIjoiYWRtaW4iLCJhZG1p
biI6dHJ1ZSwiZXhwIjoxNzMzODcxNDk5fQ.aA0vQd1y3cxV1iOnza66OwqgyYMJ4BzU2bGJICaCllg"}
OK
PS C:\Users\admin> Invoke-RestMethod -Uri "http://127.0.0.1:8082/restricted/count" -Method Get -Headers @{Authorization
= "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJuYW1lIjoiYWRtaW4iLCJhZG1pbiI6dHJ1ZSwiZXhwIjoxNzMzODcxNDk5fQ.aA0vQd1y3c
xV1iOnza66OwqgyYMJ4BzU2bGJICaCllg"}
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.eyJuYW1lIjoiYWRtaW4iLCJhZG1p
biI6dHJ1ZSwiZXhwIjoxNzMzODcxNDk5fQ.aA0vQd1y3cxV1iOnza66OwqgyYMJ4BzU2bGJICaCllg"}
OK
PS C:\Users\admin> Invoke-RestMethod -Uri "http://127.0.0.1:8082/restricted/count" -Method Get -Headers @{Authorization
= "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJuYW1lIjoiYWRtaW4iLCJhZG1pbiI6dHJ1ZSwiZXhwIjoxNzMzODcxNDk5fQ.aA0vQd1y3c
xV1iOnza66OwqgyYMJ4BzU2bGJICaCllg"}
58
```

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