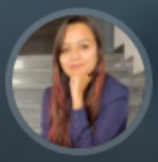
How to Implement Golang JWT Authentication and Authorization



Archita Nayak
Technical Writer

March 22, 2021 9 min read Last Updated on March 23, 2021

Table of Content

Introduction

Would you let anyone enter your house without knowing the person's identity? The answer would be – Obviously No! So, we have the same scenario with our web applications too. It's necessary to authenticate a user's identity before making requests using APIs. And this authentication takes place with the help of JWT .i.e., JSON Web Token. Now you might wonder what is JWT in Golang and JWT authentication. Don't panic if you are unaware of how to implement **Golang JWT authentication**. Here's a tutorial where I will make you understand how to implement Golang JWT Authentication and Authorization. So let's get started.

Exploring JSON Web Token

Under this section, we will comprehensively understand what is JWT, how does JSON Web token look like, and what JSON web token consists of.

What is a JSON Web Token?

A JWT token is a cryptographically signed token which the server generates and gives to the client. The client uses JWT for making various requests to the server.

The token can be signed using two algorithms: HMAC or SHA256.

SHA256 hashes the message without the need of any external input. It guarantees only message integrity.
HMAC needs a private key in order to hash the message. It guarantees message integrity and authentication.

How Does a JSON Web Token look like?

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyaWQiOiIxZGQ5MDEwYy00MzI4LTRoZjNiOWU2LTc3N2
```

The above token is invalid. It cannot be used for production.

What comprises a JSON Web Token?

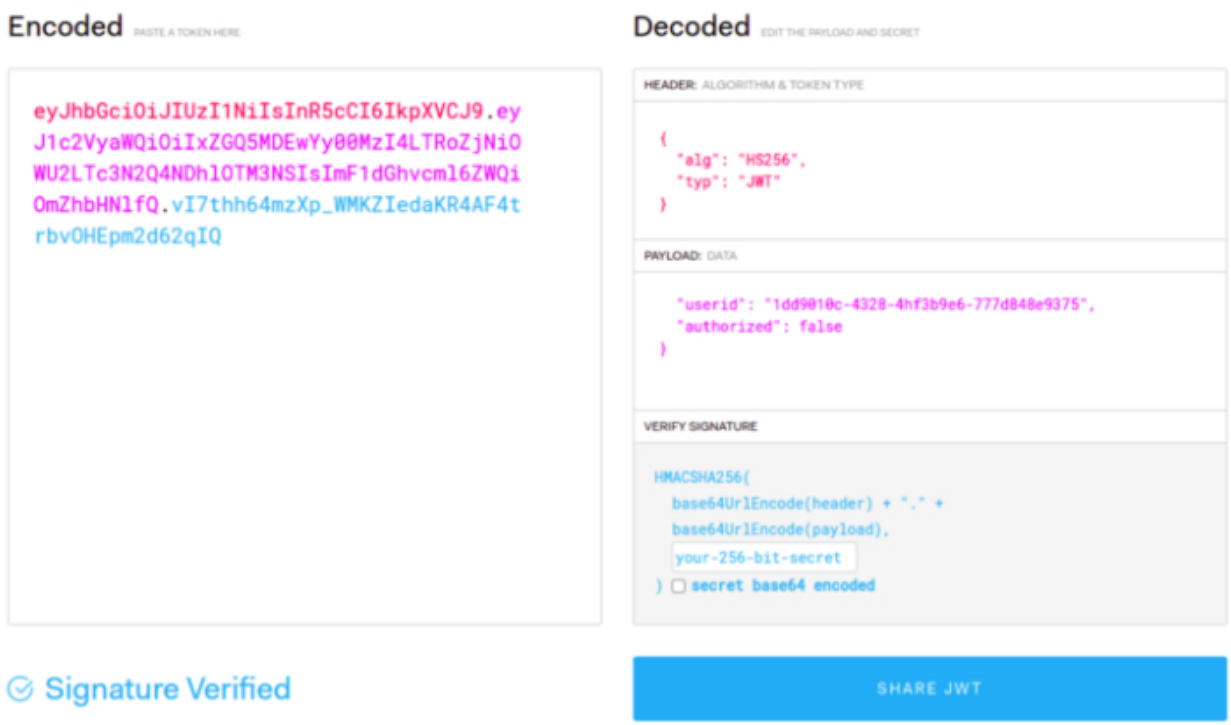
A JSON Web Token consists of three parts which are separated using .(dot) :

- **Header:** It indicates the token's type it is and which signing algorithm has been used.

- **Payload:** It consists of the claims. And claims comprise of application's data(email id, username, role), the expiration period of a token (Exp), and so on.

- **Signature:** It is generated using the secret (provided by the user), encoded header, and payload.

To test the token, you can go to https://jwt.io/.



We can set the expiration period for any JSON Web Token. Here in this application, we will consider Access Token and Refresh Token. Let's see the difference.

Access Token: An access token is used for authenticating the requests sent to the server. We add Follow these steps for Golang JWT Authentication and Authorization-

Create a directory

Create a directory called *jwt-practice*.

```
mkdir jwt-practice
cd jwt-practice
```

Initializing with *go.mod*

Initialize it with go.mod, for dependency management, using –

```
go mod init jwt-practice
```

Create a *main.go*

Create a *main.go* file in the root directory of the project. For simplicity, I will the entire code in

```
var router *mux.Router

func CreateRouter() {
        router = mux.NewRouter()
}

func InitializeRoute() {
        router.HandleFunc("/signup", SignUp).Methods("POST")
        router.HandleFunc("/signin", SignIn).Methods("POST")
}

func main() {
        CreateRouter()
        InitializeRoute()
}
```

Create some Structures

Let's get our hands on to create some structs.

```
type User struct {
        gorm.Model
        Name     string `json:"name"`
        Email    string `gorm:"unique" json:"email"`
        Password string `json:"password"`
        Role     string `json:"role"`
}

type Authentication struct {
        Email    string `json:"email"`
        Password string `json:"password"`
}
```

```
func main() {
        CreateRouter()
        InitializeRoute()
}
```

Create some Structures

Let's get our hands on to create some structs.

```
type User struct {
        gorm.Model
        Name     string `json:"name"`
        Email    string `gorm:"unique" json:"email"`
        Password string `json:"password"`
        Role     string `json:"role"`

        err = sqldb.Ping()
        if err != nil {
                log.Fatal("database connected")
        }

        fmt.Println("connected to database")
        return connection
}
func InitialMigration() {
        connection := GetDatabase()
        defer Closedatabase(connection)
        connection.AutoMigrate(User{})
}

func Closedatabase(connection *gorm.DB) {
        sqldb := connection.DB()
        sqldb.Close()
}
```

Sign Up process

The **SignUp** function opens the database connection, receives user data from the form, and checks if the user already exists in the database or not. If the user is already present in the database, it returns an error, otherwise hash the user password and creates a new database entry. Copy-paste the below-mentioned code in your file.

```
func SignUp(w http.ResponseWriter, r *http.Request) {
        connection := GetDatabase()
        defer Closedatabase(connection)

        var user User
```

```
        defer Closedatabase(connection)
        connection.AutoMigrate(User{})
}

func Closedatabase(connection *gorm.DB) {
        sqldb := connection.DB()
        sqldb.Close()
}
```

Sign Up process

The **SignUp** function opens the database connection, receives user data from the form, and checks if the user already exists in the database or not. If the user is already present in the database, it returns an error, otherwise hash the user password and creates a new database entry. Copy-paste the below-mentioned code in your file.

```
func SignUp(w http.ResponseWriter, r *http.Request) {
        connection := GetDatabase()
        defer Closedatabase(connection)

        var user User
        err := json.NewDecoder(r.Body).Decode(&user)
```

```
func GeneratehashPassword(password string) (string, error) {
        bytes, err := bcrypt.GenerateFromPassword([]byte(password), 14)
        return string(bytes), err
}
```

So, we are done with the fundamental setup in our *main.go*. It's time to start coding for the Authentication and Authorization part. But, before that let me brief you regarding the difference between the two processes.

Authentication vs Authorization

Authentication can be defined as validating the users of any particular application. And that's why it is said to be the crucial and foremost step in developing an application. It directly concerns security issues. Allowing someone to make a request to the server is a basic example of authentication.

Authorization is a process of where the user roles are being managed. It can be briefed as giving a user some specific permissions for accessing particular resources.

First, we will begin the process of authentication.

Generate JWT

Write the following function to create Golang JWT:

The *GenerateJWT()* function takes email and role as input. Creates a token by HS256 signing method and adds authorized email, role, and exp into claims. Claims are pieces of information added into tokens.

```
func GenerateJWT(email, role string) (string, error) {
```

Authentication can be defined as validating the users of any particular application. And that's why it is said to be the crucial and foremost step in developing an application. It directly concerns security issues. Allowing someone to make a request to the server is a basic example of authentication.

Authorization is a process of where the user roles are being managed. It can be briefed as giving a user some specific permissions for accessing particular resources.

First, we will begin the process of authentication.

Generate JWT

Write the following function to create Golang JWT:

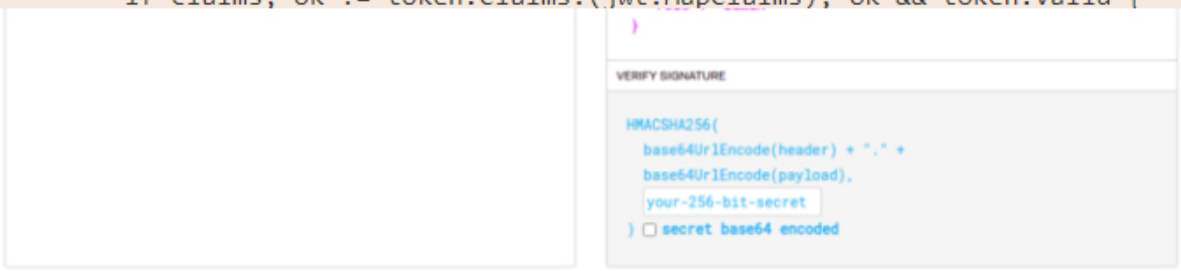
The *GenerateJWT()* function takes email and role as input. Creates a token by HS256 signing method and adds authorized email, role, and exp into claims. Claims are pieces of information added into tokens.

```
func GenerateJWT(email, role string) (string, error) {
        var authdetails Authentication
        err := json.NewDecoder(r.Body).Decode(&authdetails)
        if err != nil {
                var err Error
                err = SetError(err, "Error in reading body")
                w.Header().Set("Content-Type", "application/json")
                json.NewEncoder(w).Encode(err)
                return
        }

        var authuser User
        connection.Where("email = ?", authdetails.Email).First(&authuser)
        if authuser.Email == "" {
                var err Error
                err = SetError(err, "Username or Password is incorrect")
                w.Header().Set("Content-Type", "application/json")
                json.NewEncoder(w).Encode(err)
                return
        }

        check := CheckPasswordHash(authdetails.Password, authuser.Password)

        if !check {
                var err Error
                err = SetError(err, "Username or Password is incorrect")
                w.Header().Set("Content-Type", "application/json")
                json.NewEncoder(w).Encode(err)
                return
        }

        validToken, err := GenerateJWT(authuser.Email, authuser.Role)
        if err != nil {
                var err Error
```

```
        connection.Where("email = ?", authdetails.Email).First(&authuser)
        if authuser.Email == "" {
                var err Error
                err = SetError(err, "Username or Password is incorrect")
                w.Header().Set("Content-Type", "application/json")
```

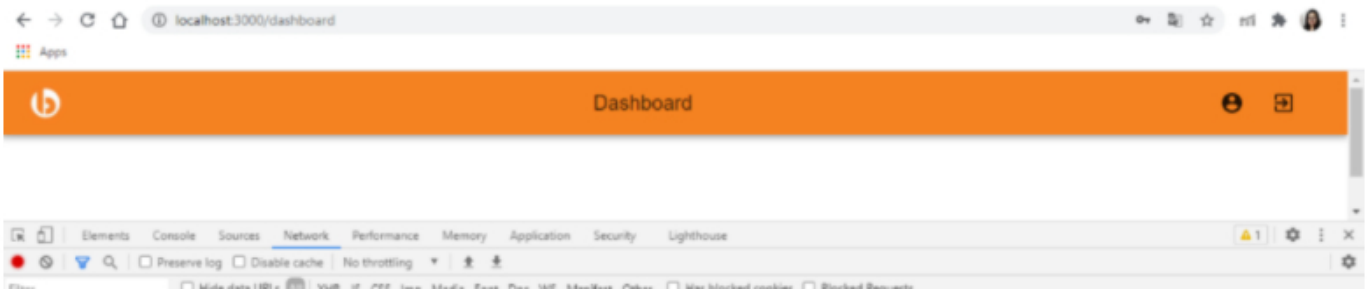
```
                json.NewEncoder(w).Encode(err)
                return
        }

        check := CheckPasswordHash(authdetails.Password, authuser.Password)

        if !check {
                var err Error
                err = SetError(err, "Username or Password is incorrect")
                w.Header().Set("Content-Type", "application/json")
                json.NewEncoder(w).Encode(err)
                return
        }

func IsAuthorized(handler http.HandlerFunc) http.HandlerFunc {
        return func(w http.ResponseWriter, r *http.Request) {

                if r.Header["Token"] == nil {
                        var err Error
                        err = SetError(err, "No Token Found")
                        json.NewEncoder(w).Encode(err)
                        return
                }

                var mySigningKey = []byte(secretkey)

                token, err := jwt.Parse(r.Header["Token"][0], func(token *jwt.Token) (i
                        if _, ok := token.Method.(*jwt.SigningMethodHMAC); !ok {
                                return nil, fmt.Errorf("There was an error in parsing")
                        }
                        return mySigningKey, nil
                })


                if err != nil {
                        var err Error
                        err = SetError(err, "Your Token has been expired")
                        json.NewEncoder(w).Encode(err)
                        return
                }

                if claims, ok := token.Claims.(jwt.MapClaims); ok && token.Valid {
                        if claims["role"] == "admin" {

                                r.Header.Set("Role", "admin")
                                handler.ServeHTTP(w, r)
                                return
```

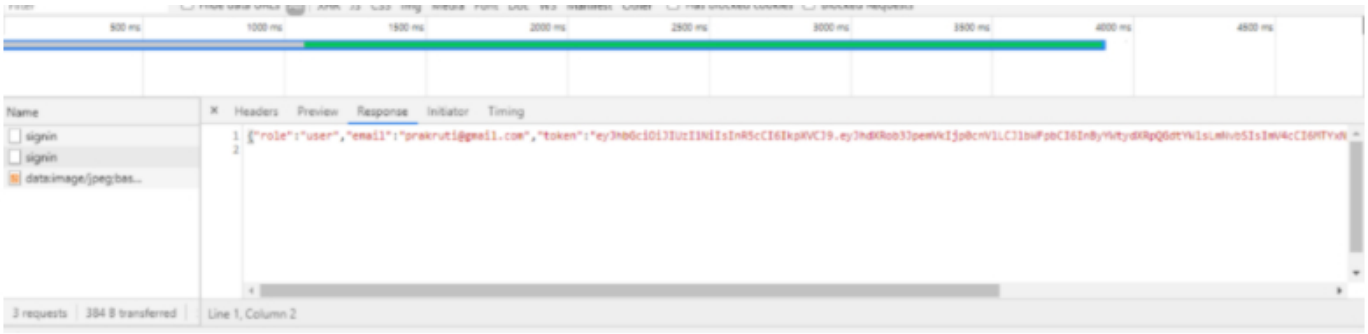
```
token, err := jwt.Parse(r.Header["Token"][0], func(token *jwt.Token) (i
        if _, ok := token.Method.(*jwt.SigningMethodHMAC); !ok {
                return nil, fmt.Errorf("There was an error in parsing")
        }
        return mySigningKey, nil
})


if err != nil {
        var err Error
        err = SetError(err, "Your Token has been expired")
        json.NewEncoder(w).Encode(err)
        return
}

if claims, ok := token.Claims.(jwt.MapClaims); ok && token.Valid {
```



Thus, you are done with generating the Golang JWT. Further, for your frontend side, you can store this token in your local storage and use it in different API requests. Refer to the below images-

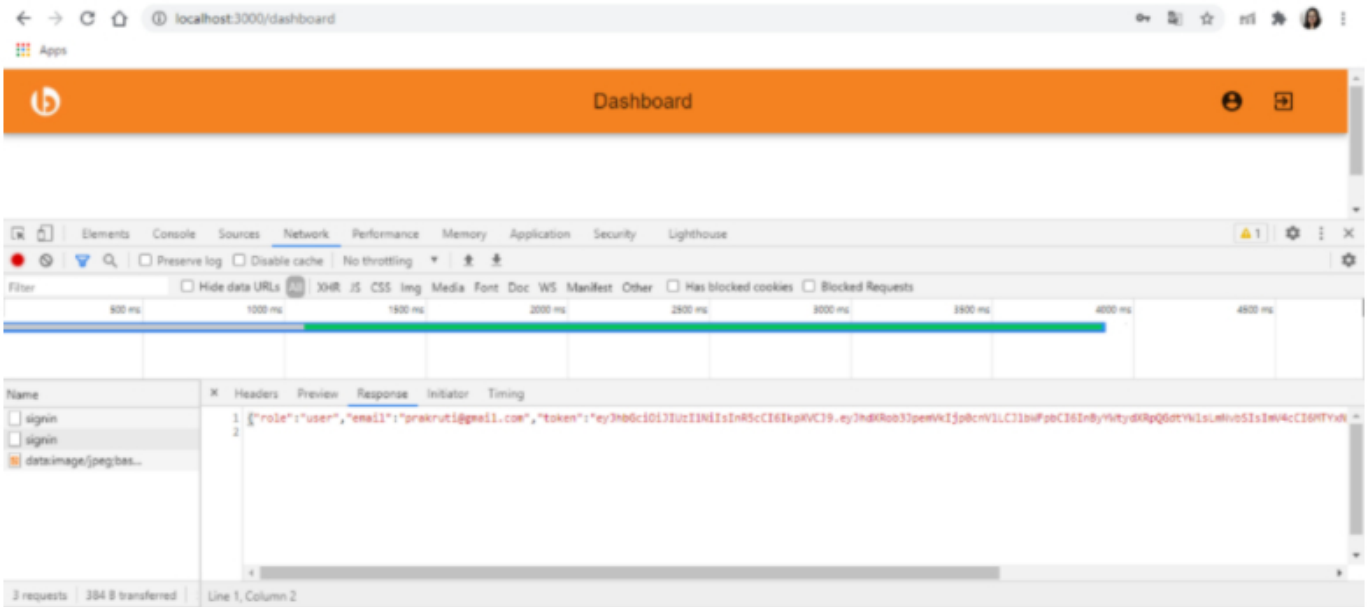
(1) Signed In successfully and receiving Golang JWT in the response. You can see the *"role": "user"* which satisfies the authorization part. It means that only specific resources will be accessible to the user role.





images-

(1) Signed In successfully and receiving Golang JWT in the response. You can see the *"role": "user"* which satisfies the authorization part. It means that only specific resources will be accessible to the user role.



touch to start small, scale-up, and go Agile.

Your Name | Email Address | Phone Number

Your Message

Inquire Now

Or

E-mail us : solutions@bacancy.com

Your Success Is Guaranteed !

Related Articles


Your Message

Inquire Now

Or

E-mail us : solutions@bacancy.com

Your Success Is Guaranteed !

Related Articles

May 7, 2021

Golang

Go Best Practices: Optimize Golang Code For Better Performance

By : Paridhi Wadhwani

Quick Summary: This article is for Go enthusiasts who address themselves as the Gophers. It is a collection of applicable Go best practices that lead...

10 Minute Read

Golang Code For Better Performance

By : Paridhi Wadhwani

Quick Summary: This article is for Go enthusiasts who address themselves as the Gophers. It is a collection of applicable Go best practices that lead...

10 Minute Read

Ahmedabad

Australia

351A Hampstead Rd, Northfield SA 5085

UAE

1608 Clover Bay, Business Bay, Dubai, UAE. PO Box 62049

Sweden

Junkergatan 4, 126 53 Hagersten

Get in Touch

Contact Number
+1 347 441 4161

Email Us
solutions@bacancy.com