HOW TO IMPLEMENT JWT IN GOLANG EASILY

get to know me better at: https://fauzancodes.id

Have you ever wondered how an application knows whether a user is logged in or not?

Take, for example, a dashboard page. How does the system decide if you're authorized to view it?

That's where **JWT**, or JSON Web Token, comes into play. **It's all about authentication**.

get to know me better at: https://fauzancodes.id

What Is JWT?

JWT (JSON Web Token) is a token designed to store data that helps us track if a user is logged in or not.

For example, it might store a user ID. This token is placed in the HTTP header, and when the server receives a request, it checks whether the token is present. If it exists, the server checks its validity—if it's valid, the user is logged in; if not, the user gets redirected to the login page. Simple, right?

But, hold on a second—do you think JWT is just some random string of characters? Think again. It’s not just a jumbled mess of letters and numbers.

JWT has a specific structure: **the header, payload, and signature**.

Structure of JWT

Header: This part contains the metadata, usually specifying the type of token (JWT) and the signing algorithm being used (HMAC SHA256, for example).

Payload: This section holds the actual data—the claims. The claims are statements about an entity (typically, the user) and any additional data. The payload can be anything, but it is typically used to store user info like IDs, roles, etc.

Signature: The signature is created by taking the encoded header and payload, then signing it using a secret key and the specified algorithm (like HMAC SHA256). The purpose of the signature is to ensure that the token hasn't been tampered with.

Let's take an example from the following jwt token

eyJhbGciOiAiSFMyNTYiLCAidHlwIjogIkpXVCJ9.eyJzdWIiOiAiMTIzNDU2Nzg5MCIsICJuYW1lIjogIkpvaG4gRG9lIiwgImlhdCI6IDE1MTYyMzkwMjJ9.x2lGe-57X6MvqOxij5FTptOB9niVh7YXqgsj-W4Uu0E

A bit confusing isn't it?

But try to look again, it consists of 3 parts separated by a dot (.).

Header

The header typically consists of two parts:

1. typ (type): This specifies that the token is a JWT.
2. alg (algorithm): This tells which signing algorithm is used, such as HMAC SHA256.

Example: {"alg": "HS256", "typ": "JWT"}

When encoded in base64, this becomes something like:

eyJhbGciOiAiSFMyNTYiLCAidHlwIjogIkpXVCJ9

Payload

This part contains the "claims." Claims are statements about an entity (typically the user) and any additional data.

Example: {"id": "1234567890", "name": "Mr. Blue"}

When base64 encoded, it looks something like this:

eyJzdWIiOiAiMTIzNDU2Nzg5MCIsICJuYW1lIjogIkpvaG4gRG9lIiwgImlhdCI6IDE1MTYyMzkwMjJ9

Signature

The signature is created by taking the encoded header and payload, concatenating them with a period (.), and then signing it with a **secret key** using the specified algorithm.

After applying the HMAC SHA256 algorithm with the secret key, you get a string like:

x2lGe-57X6MvqOxij5FTptOB9niVh7YXqgsj-W4Uu0E

Now, the final JWT token will look like this (it's a combination of the encoded **header, payload, and the signature, all separated by periods**):

eyJhbGciOiAiSFMyNTYiLCAidHlwIjogIkpXVCJ9.eyJzdWIiOiAiMTIzNDU2Nzg5MCIsICJuYW1lIjogIkpvaG4gRG9lIiwgImlhdCI6IDE1MTYyMzkwMjJ9.x2lGe-57X6MvqOxij5FTptOB9niVh7YXqgsj-W4Uu0E

How to Verify a JWT

When a request comes in with a JWT in the HTTP header, we don't just trust it blindly. We verify it. Here's how:

1. Extract the JWT from the HTTP header.

2. Re-generate the signature using the same header and payload, along with the secret key.

3. Compare the newly generated signature with the signature in the HTTP header. If they match, the token is valid, and the user is logged in. If they don't, it's a fail.

Okay, enough material, now let's get our hands dirty with the **jwt implementation in golang!**

First, create a function to generate jwt tokens.

```go
var SecretKey = config.LoadConfig().SecretKey

func GenerateToken(claims *jwt.MapClaims) (string, error) {
  //encode header and payload into token
  token := jwt.NewWithClaims(jwt.SigningMethodHS256, claims)

  //sign the token with a secret key
  //then combine the encoded header, payload and signature
  webtoken, err := token.SignedString([]byte(SecretKey))
  if err != nil {
    err = errors.New("failed to generate token: " + err.Error())
    return "", err
  }

  return webtoken, nil
}
```

Next, create a function to verify the signature.

```go
func VerifyToken(tokenString string) (*jwt.Token, error) {
  //verify payload and signature
  token, err := jwt.Parse(tokenString, func(token *jwt.Token) (any, error) {
    if _, isValid := token.Method.(*jwt.SigningMethodHMAC); !isValid {
      return nil, fmt.Errorf("unexpected signing method: %v", token.Header["alg"])
    }
    return []byte(SecretKey), nil
  })
  if err != nil {
    err = errors.New("failed to verify token: " + err.Error())
    return nil, err
  }

  return token, nil
}
```

get to know me better at: https://fauzancodes.id

Next, create a decode function to retrieve data from the successfully verified jwt token.

```go
func DecodeToken(tokenString string) (jwt.MapClaims, error) {
  //vefify token
  token, err := VerifyToken(tokenString)
  if err != nil {
    err = errors.New("failed to decode token: " + err.Error())
    return nil, err
  }

  //retrieve data from token
  claims, isOk := token.Claims.(jwt.MapClaims)
  if isOk && token.Valid {
    return claims, nil
  }

  return nil, fmt.Errorf("invalid token")
}
```

Next, use the function to generate a JWT token on the login endpoint.

```go
//data initiation for token
claims := jwt.MapClaims{}
claims["id"] = user[0].ID //retrieve from user data
claims["exp"] = time.Now().Add(time.Hour * 24).Unix()

// /generate token
token, err := webToken.GenerateToken(&claims)
if err != nil {
  c.AbortWithStatusJSON(
    http.StatusUnauthorized,
    dto.Response{
      Status:  401,
      Message: "Failed to generate jwt token",
      Error:   err.Error(),
    },
  )
}

c.JSON(
  statusCode,
  dto.Response{
    Status:  statusCode,
    Message: "Success to login",
    Data:    token,
  },
)
}
```

Then, create a middleware function to check the jwt token in the http header.

```go
func Auth() gin.HandlerFunc {
  return func(c *gin.Context) {
    //retrieve token from http header
    token := c.GetHeader("Authorization")

    //check if token is in http header
    if token == "" {
      c.AbortWithStatusJSON(http.StatusUnauthorized, dto.Response{
        Status:  http.StatusBadRequest,
        Message: "No jwt token provided",
      })
    }

    //decode token
    token = strings.Split(token, " ")[1]
    claims, err := jwt.DecodeToken(token)

    if err != nil {
      c.AbortWithStatusJSON(http.StatusUnauthorized, dto.Response{
        Status:  http.StatusUnauthorized,
        Message: "Failed to decode jwt token",
      })
    }

    c.Set("currentUser", claims)
    c.Next()
  }
}
```

Finally, apply the middleware to every endpoint that requires authentication.

```
//example of using middlewares on one of the endpoints
auth.GET("/user", middlewares.Auth(), controllers.GetCurrentUser)
```

Now, at this point, you might be thinking—“But wait! The payload is only base64 encoded. So, doesn’t that mean anyone could decode it and see the user ID?” Yes, you’re right, the payload is not encrypted, it’s just encoded. But here's the twist—JWT is for authentication, not security. It’s perfectly fine that the user ID is visible because the secret key ensures that the token is valid and hasn't been tampered with.

Sure, someone could try to tamper with the payload, but when they change the user ID, the signature will fail to verify. So, in essence, the exposed user ID doesn’t really matter. But here's a critical caveat: never store sensitive information like email addresses or passwords in the payload! That would be a security disaster.

Conclusion

So, now that you have a clearer picture of how JWT works, don’t just take my word for it. Dive deeper, explore the concepts, and make sure you're using JWT correctly in your own projects.

But remember, JWT is not a security solution—just an authentication one.

Got any questions or thoughts on JWT? Leave a comment or shoot me a message. Let’s discuss!

get to know me better at: https://fauzancodes.id