Understanding Session-Based Authentication from Scratch

Munechika Hayashi Follow 5 min read · Nov 25, 2023

96

User authentication in applications is a vital part of both security and user experience. This article explains the basics of session-based authentication (hereafter referred to as session authentication) and provides a detailed implementation method using the Go language (Golang). It's written simply using only the standard library, so it should be easy to understand even for those not familiar with Go.

What is Session Authentication?

Session authentication is a method of identifying users in subsequent requests after they have logged in once. Typically, session authentication involves the following steps:

1. The user sends a login request with an ID and password to the server through a browser.

2. The server verifies the provided authentication information and establishes a session if it's correct. Session information is stored both on the server and in a Cookie. For this example, we'll store it in memory, but actual applications commonly use databases or other storage.
3. The server sends a response back to the browser indicating that the session has been established.
4. The browser sends requests to the server, including session information, indicating that the user is authenticated.
5. The server verifies the session information included in the request and authorizes the user.
6. The server sends a response back to the browser, indicating that the user is authorized and allowed access to specific actions or resources.
7. When the user requests to log out, the browser sends a logout request to the server.
8. The server deletes the session from the server and Cookie, invalidating the session information.
9. The server sends a response back to the browser, indicating that the session has been successfully terminated.

```
Browser                    Server
  |                          |
  |--- 1. Login ------>      |
  |   (ID and Password)      |
  |                          | 2. Establish Session (Stored on server and Cool
  |                          |
  |<-- 3. Response -----     |
```

```
    |                          |
    |                          |
    |--- 4. Request ------>    |
    |   (Includes Session Info)| 5. Authorization Successful
    |                          |
    |<-- 6. Response -----     |
    |   (User Identified)      |
    |                          |
    |--- 7. Logout ------>     |
    |                          | 8. Delete Session
    |<-- 9. Response ---       |
    |   (Session Info Cleared)|
    |                          |
```

Sample Implementation of Session Authentication in Go

Below is an implementation of the basic flow of session authentication described above:

```go
package main

import (
 "fmt"
 "net/http"
 "sync"
)

var (
 // Map for storing session information
 sessions = make(map[string]bool)
 // Mutex to synchronize access to session information
 sessionMutex = &sync.Mutex{}
)

func main() {
 http.HandleFunc("/login", loginHandler)
 http.HandleFunc("/dashboard", dashboardHandler)
 http.HandleFunc("/logout", logoutHandler)
 http.ListenAndServe(":8080", nil)
```

```
}

func loginHandler(w http.ResponseWriter, r *http.Request) {
 // Normally, implement user authentication logic with ID and password here

 // Generate a session ID (here, simply using a username)
 sessionID := "user123" // In reality, generate a more secure random ID

 // Save the session on the server side
 sessionMutex.Lock()
 sessions[sessionID] = true
 for id := range sessions {
  fmt.Println("loginHandler: Current session ID is ", id)
 }
 sessionMutex.Unlock()

 // Send the session ID to the client as a Cookie
 http.SetCookie(w, &http.Cookie{
  Name:   "session_id",
  Value:  sessionID,
  Path:   "/",
  MaxAge: 60, // Set the expiration time to 60 seconds (1 minute)
 })

 w.Write([]byte("login successfully!"))
}

func dashboardHandler(w http.ResponseWriter, r *http.Request) {
 // Retrieve session ID from Cookie
 cookie, err := r.Cookie("session_id")
 if err != nil {
  http.Error(w, "Unauthorized", http.StatusUnauthorized)
  return
 }

 sessionMutex.Lock()
 authenticated, ok := sessions[cookie.Value]
 if ok {
  for id := range sessions {
   fmt.Println("dashboardHandler: Current session ID is ", id)
  }
 }
 sessionMutex.Unlock()

 // Check if the session is valid
 if !ok || !authenticated {
  http.Error(w, "Forbidden", http.StatusForbidden)
  return
```

```
 }

 w.Write([]byte("Welcome to your dashboard!"))
}

func logoutHandler(w http.ResponseWriter, r *http.Request) {
 // Retrieve session ID from Cookie and delete the session on the server sid
 cookie, err := r.Cookie("session_id")
 if err == nil {
  sessionMutex.Lock()
  delete(sessions, cookie.Value)
  if len(sessions) == 0 {
   fmt.Println("logoutHandler: Current session ID is nothing")
  }
  sessionMutex.Unlock()
 }

 http.SetCookie(w, &http.Cookie{
  Name:   "session_id",
  Value:  "",
  Path:   "/",
  MaxAge: -1, // Delete the Cookie
 })

 w.Write([]byte("Logout successful!"))
}
```

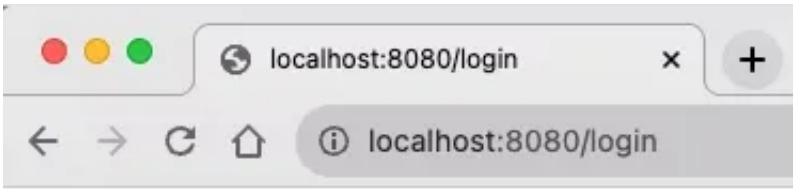
Running it in Practice

Starting the Server

```
go run ./main.go
```

Logging In

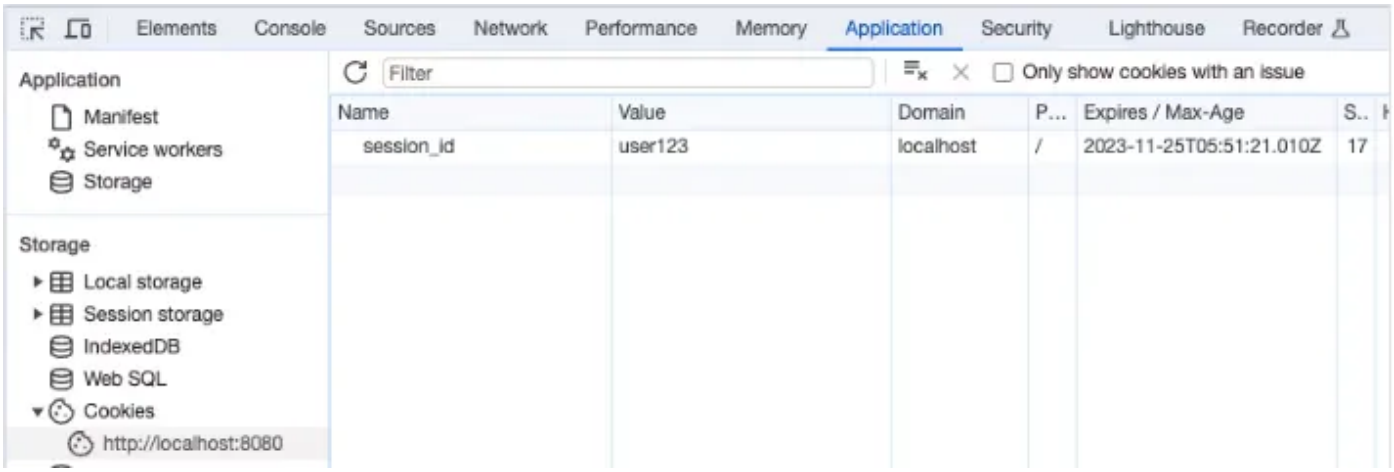
Access `http://localhost:8080/login` in your browser.



You'll see a successful access message, and the server will hold session information in memory.

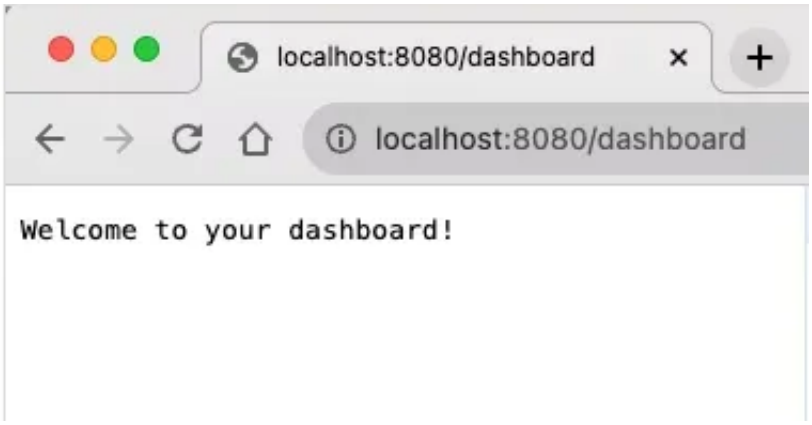
```
loginHandler: Current session ID is  user123
```

You can check the Cookie in the browser's developer tools, showing the session set and its expiration time.



Accessing the Dashboard

Next, access `http://localhost:8080/dashboard`. You'll see a successful access message



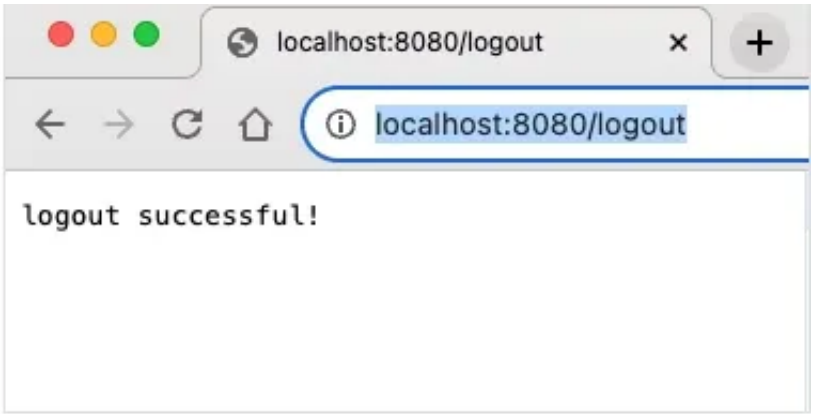
The server still holds session information in memory.

```
dashboardHandler: Current session ID is  user123
```

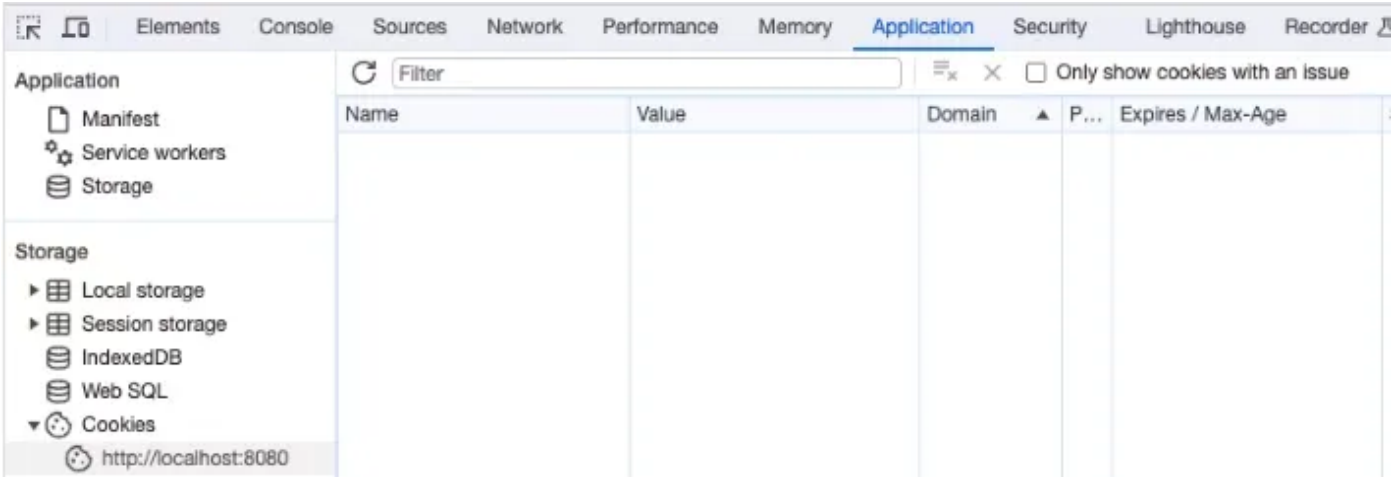
If you see 'Forbidden,' the session has expired, and you need to log in again.

Logging Out

Finally, access `http://localhost:8080/logout`.



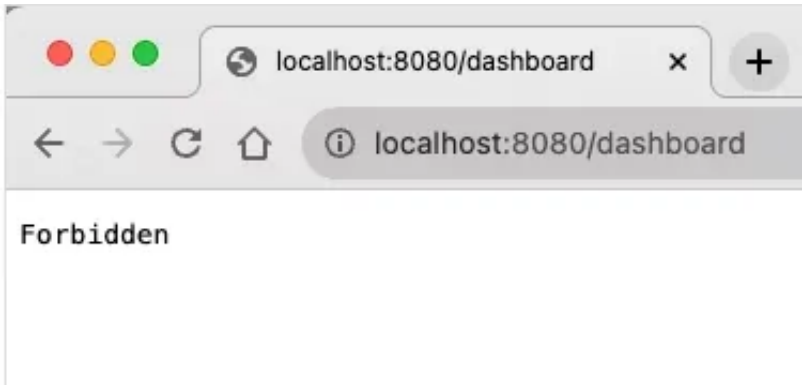
This will delete the session stored in the Cookie.



At this point, the server will show that it no longer holds any session information.

```
logoutHandler: Current session ID is nothing
```

Trying to access the dashboard now will result in an error due to the absence of a session.



Conclusion

Session authentication is a crucial component of web applications, and it

can be effectively implemented using the Go language. When developing commercially, ensure to adopt appropriate security measures to provide a safe and comfortable experience for users.

References

https://sherryhsu.medium.com/session-vs-token-based-authentication-11a6c5ac45e4

Golang Authentication Beginner Programming

Written by Munechika Hayashi

12 followers · 3 following

Follow

Medium Search Write

I Ishu

What are your thoughts?

More from Munechika Hayashi

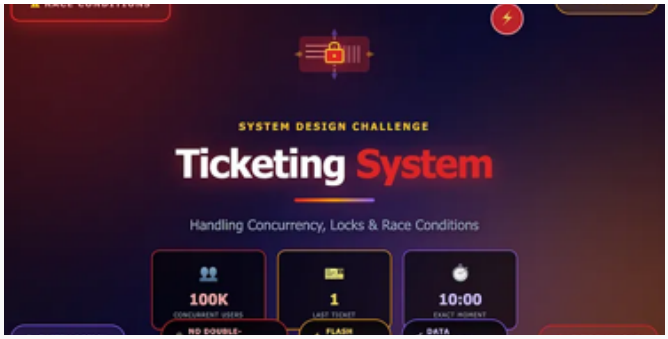
Munechika Hayashi

Using assert.InDelta and assert.InEpsilon for Handling Uncertainty in Tests with Go

Jul 13, 2024

See all from Munechika Hayashi

Recommended from Medium

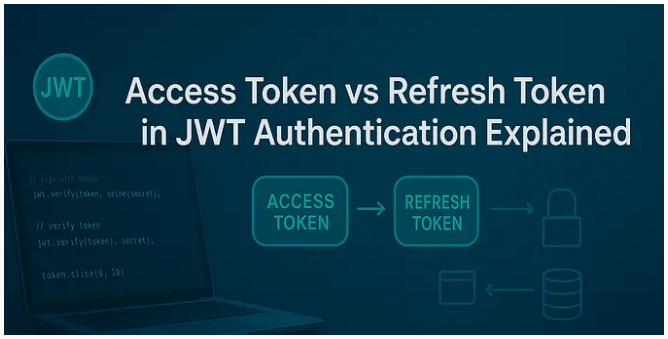


Arvind Kumar

Building a Ticketing System: Concurrency, Locks, and Race...

What happens when 100,000 fans try to book the same concert ticket at exactly...

Oct 30 398 5



In codingsprints by codingsprints

Access Token vs Refresh Token in JWT Authentication Explained

Learn the difference between Access Token and Refresh Token in JWT authentication,...

Aug 22 3

WeDev

🚀 Express vs Fastify (Which is better)

If you're still using Express.js in 2025, it's like using a Nokia 1100 in the iPhone era ...

Jun 29 82 2

Umesh Kumar Yadav

DTOs vs. Entities: A Necessary Separation for Cleaner, More...

In the world of software development, particularly when building robust and...

Jun 29 29

In ITNEXT by Animesh Gaitonde

Solving Double Booking at Scale: System Design Patterns from To...

Learn how Airbnb, Ticketmaster, and booking platforms handle millions of...

Oct 8 2.1K 26

Tosny

7 Websites I Visit Every Day in 2025

If there is one thing I am addicted to, besides coffee, it is the internet.

Sep 23 7.2K 257

See more recommendations