Flaky order APP

The application I've tried has several issues :

1. **The user is redirected to a login page that hasn't been created in HTML format.**

   Issue: Inside fmt.Fprint, there is no content-type specified as HTML. In order to render HTML in Go, we need to add the content type text/html to all HTML responses.

```
func loginHandler(w http.ResponseWriter, r *http.Request) {
    if r.Method == http.MethodPost {
        username := r.FormValue("username")
        password := r.FormValue("password")

        if storedPass, ok := users[username]; ok && storedPass == password {
            http.SetCookie(w, &http.Cookie{
                Name:  "session",
                Value: username,
            })
            http.Redirect(w, r, "/", http.StatusSeeOther)
            return
        }
        http.Error(w, "Invalid credentials", http.StatusUnauthorized)
        return
    }

    fmt.Fprint(w, `<form action="/login" method="post">
Username: <input type="text" name="username"><br>
Password: <input type="password" name="password"><br>
<input type="submit" value="Login">
</form>`)
}
```

   Solution 1: Setting Content-Type Header: Add the Content-Type: text/html header to all HTML responses

```
        return
    }
    w.Header().Set("Content-Type", "text/html")
    fmt.Fprintf(w,
        `<!DOCTYPE html>
        <html lang="en">
        <head>
            <meta charset="UTF-8">
            <meta name="viewport" content="width=device-width, initial-scale=1.0">
            <title>Login</title>
        </head>
        <body>
            <br>Login</h2>
            <form action="/login" method="post">
                <div>
                    <label>Username:</label>
                    <input type="text" name="username">
                </div>
                <div>
                    <label>Password:</label>
                    <input type="password" name="password">
                </div>
                <div>
                    <input type="submit" value="Login">
                </div>
            </form>
        </body>
        </html>
    `)
}
```

Solution 2: Creating a Page Redirecting to login.html

The loginPagePath function, which uses filepath.Abs(filepath.Join("static", "login.html")), is used to combine the relative path to the HTML login page with the root directory and return the absolute path. This ensures that the web server can always find the login.html file regardless of the current working directory when the application is running.

```
    loginPagePath, err := filepath.Abs(filepath.Join("static", "login.html"))
    if err != nil {
        http.Error(w, "Internal Server Error", http.StatusInternalServerError)
        return
    }

    http.ServeFile(w, r, loginPagePath)
}
```

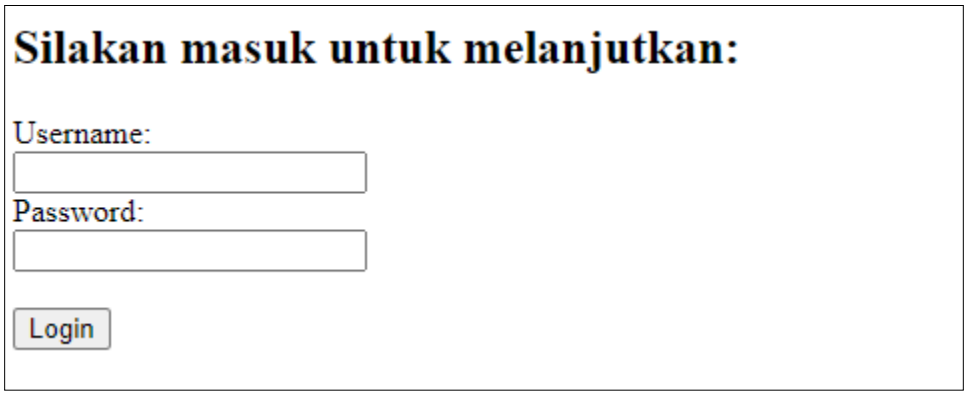
Next, we create a static folder containing the login.html file

main.go fse-assignment ● main.go C:\...\Rar$Dla3260.3854 1 login.html ×

fse-assignment > login.html > html > body > form > label

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Login</title>
</head>
<body>
    <h2>Silakan masuk untuk melanjutkan:</h2>
    <form action="/login" method="post">
        <label for="username">Username:</label><br>
        <input type="text" id="username" name="username"><br>
        <label for="password">Password:</label><br>
        <input type="password" id="password" name="password"><br><br>
        <input type="submit" value="Login">
    </form>
</body>
</html>

```

Output :



2. Issue with Product Cart:

Issued : a. In the `checkoutHandler`, improper handling occurs when a product is unavailable in the `products` list.
b. If a user attempts to add a non-existent product to the cart, the application still processes the request.
c. In the unrevised code, some HTML links lack the proper content-type settings.

```
fmt.Fprintf(w, "Hello, %s! Available products:\n", username)
for product, price := range products {
    fmt.Fprintf(w, "%s: $%.2f <a href=\"/add_to_cart/%s\">Add to cart</a>\n", product, price, product)
}

fmt.Fprint(w, "<br>Your cart:\n")
for product, quantity := range userCart {
    fmt.Fprintf(w, "%s: %d\n", product, quantity)
}

fmt.Fprint(w, `<br><a href="/checkout">Checkout</a>`)
```

Solution: Setting Content-Type to HTML and Improving UI for Product Cart

1. Add the Content-Type: text/html header to ensure proper rendering of HTML.
2. Improve the user interface (UI) for the product cart to provide a better user experience.

```
w.Header().Set("Content-Type", "text/html")
fmt.Fprintf(w, "<html><body>")
fmt.Fprintf(w, "Hello, %s! Available products:<br>", username)
fmt.Fprintf(w, "<ul>")
for product, price := range products {
    fmt.Fprintf(w, "<li>%s: $%.2f <a href=\"/add_to_cart/%s\">Add to cart</a></li>", product, price, product)
}
fmt.Fprintf(w, "</ul>")
fmt.Fprint(w, "Your cart:<br>")
fmt.Fprintf(w, "<ul>")
for product, quantity := range userCart {
    fmt.Fprintf(w, "<li>%s: %d</li>", product, quantity)
}
fmt.Fprintf(w, "</ul>")
fmt.Fprint(w, `<a href="/checkout">Checkout</a>`)
fmt.Fprintf(w, "</body></html>")
```

Output :

Hello, user1!
Your balance: $98.50

- apple: $1.00 Add to cart
- banana: $0.50 Add to cart

Your cart:

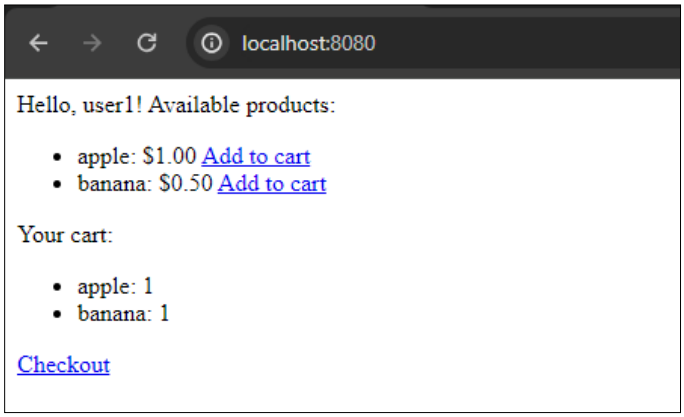
Checkout

3. Checkout Page:

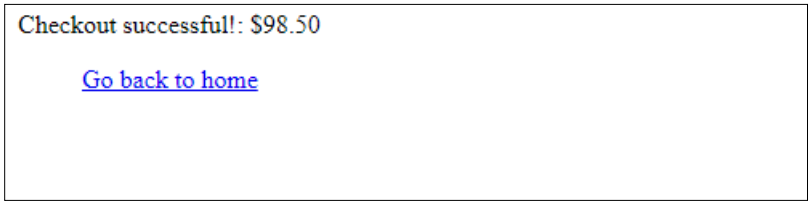
Issued : To fix the issue of the total price of products not being displayed and the information not appearing on the HTML page, we need to modify the checkoutHandler function to display the total and empty the cart after checkout. Additionally, we also need to update the HTML page to display this total price information on the checkout page.

```
    mu.Lock()
    defer mu.Unlock()

    username := cookie.Value
    userCart, ok := cart[username]
    if !ok {
        http.Redirect(w, r, "/", http.StatusSeeOther)
        return
    }

    total := 0.0
    for product, quantity := range userCart {
        price, ok := products[product]
        if !ok {
            continue
        }
        total += price * float64(quantity)
    }
    balances[username] -= total
    w.Header().Set("Content-Type", "text/html")
    fmt.Fprintf(w, "<html><body>")
    fmt.Fprintf(w, "</ul>")
    fmt.Fprintf(w, "Checkout successful!: $%.2f<br>", balances[username])
    fmt.Fprintf(w, "<ul>")
    fmt.Fprint(w, `<a href="/">Go back to home</a>`)
    fmt.Fprintf(w, "</body></html>")
}
```

Process Checkout :



Output :



Security Vulnerabilities and User Experience Improvements

1. **Security Vulnerabilities:**

- The website still uses HTTP, it would be better to switch to the HTTPS format for enhanced security.
- Handling of Passwords: Storing passwords in plaintext within the code in the users variable poses a security risk.
- Authentication: The current authentication system only uses the POST method without considering security measures such as protection against brute force attacks, man-in-the-middle attacks, or secure session management.

2. **User Experience:**

- Checking User Balances: Users' balances are checked to cover their total purchases. If the balance is insufficient, users receive an "Insufficient balance" error message.
- Handling of Invalid Products: There is no validation for invalid products added to the cart.
- Emptying Cart after Checkout: The cart is not emptied after checkout.
- Lack of Feedback during Checkout: There is a lack of feedback on the checkout page. After checkout, users receive no feedback regarding the success or failure of the transaction. Users may not know if their purchase was successful or if there was an error during the checkout process.