2021

Go in Action 2

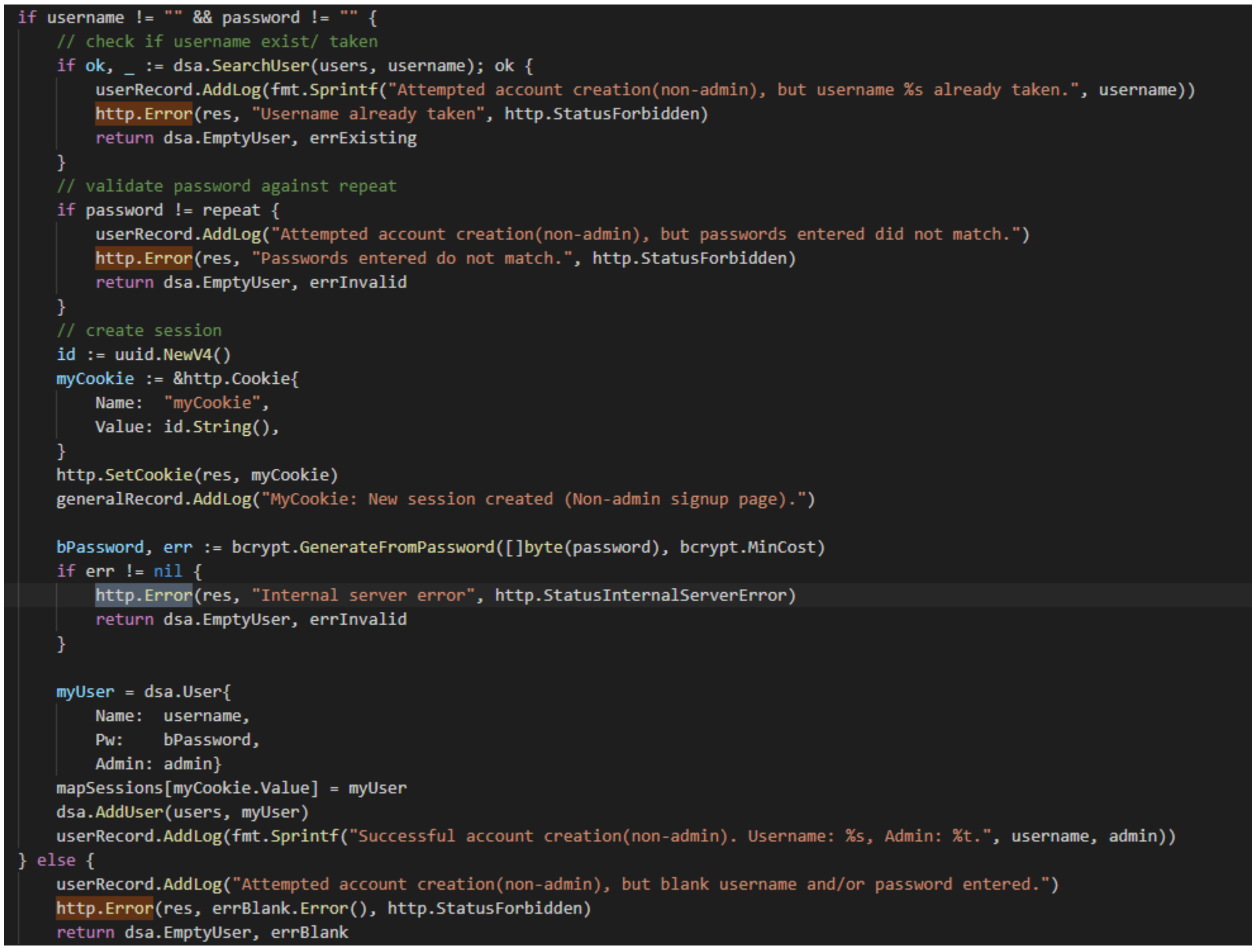
BUG-TRACKER APPLICATION

KEITH ANG

Secure Software Development Techniques

Overview

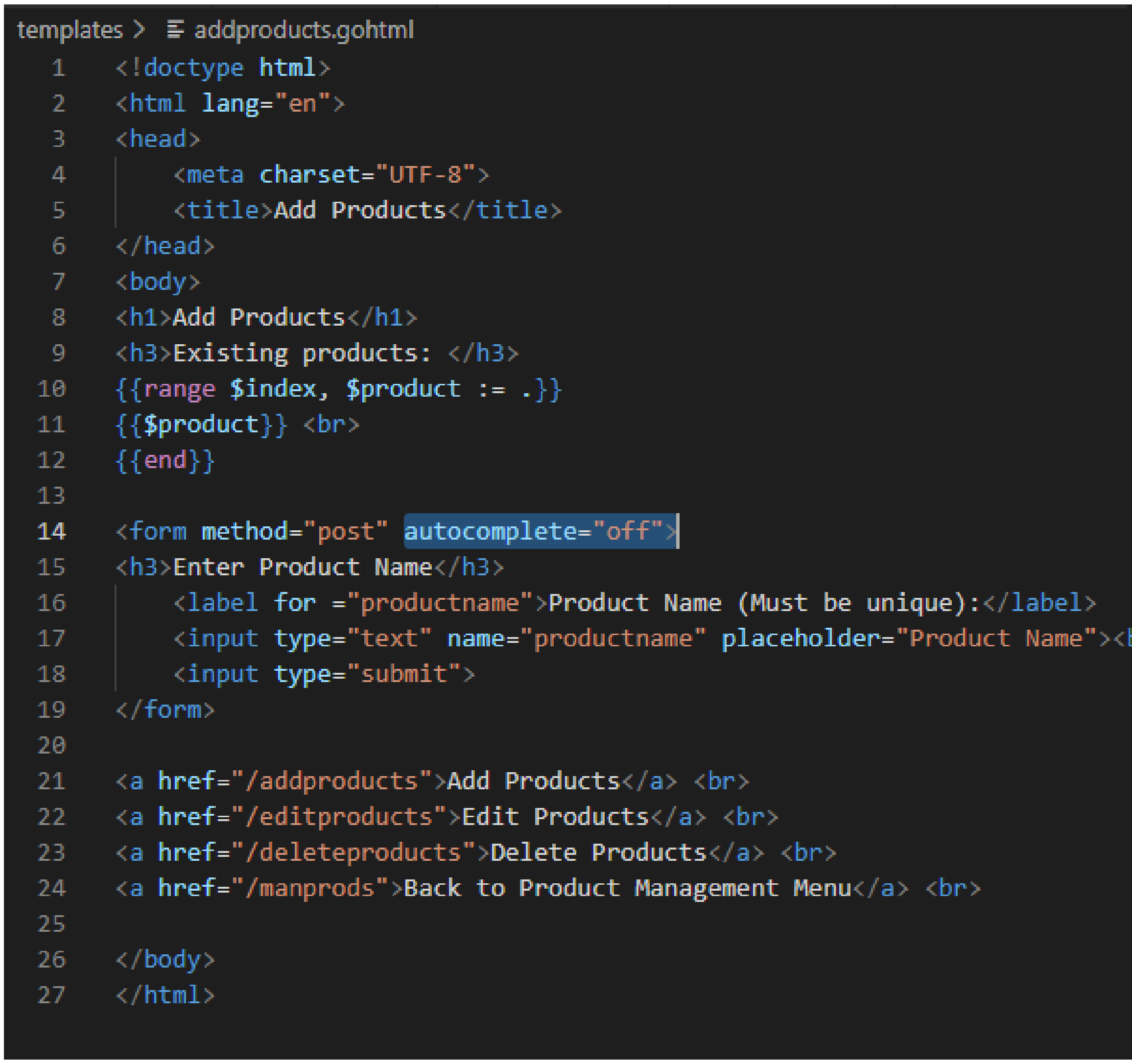
- Input Validation and Sanitization
  - Signing up for new account



```go
if username != "" && password != "" {
    // check if username exist/ taken
    if ok, _ := dsa.SearchUser(users, username); ok {
        userRecord.AddLog(fmt.Sprintf("Attempted account creation(non-admin), but username %s already taken.", username))
        http.Error(res, "Username already taken", http.StatusForbidden)
        return dsa.EmptyUser, errExisting
    }
    // validate password against repeat
    if password != repeat {
        userRecord.AddLog("Attempted account creation(non-admin), but passwords entered did not match.")
        http.Error(res, "Passwords entered do not match.", http.StatusForbidden)
        return dsa.EmptyUser, errInvalid
    }
    // create session
    id := uuid.NewV4()
    myCookie := &http.Cookie{
        Name:  "myCookie",
        Value: id.String(),
    }
    http.SetCookie(res, myCookie)
    generalRecord.AddLog("MyCookie: New session created (Non-admin signup page).")

    bPassword, err := bcrypt.GenerateFromPassword([]byte(password), bcrypt.MinCost)
    if err != nil {
        http.Error(res, "Internal server error", http.StatusInternalServerError)
        return dsa.EmptyUser, errInvalid
    }

    myUser = dsa.User{
        Name:  username,
        Pw:    bPassword,
        Admin: admin}
    mapSessions[myCookie.Value] = myUser
    dsa.AddUser(users, myUser)
    userRecord.AddLog(fmt.Sprintf("Successful account creation(non-admin). Username: %s, Admin: %t.", username, admin))
} else {
    userRecord.AddLog("Attempted account creation(non-admin), but blank username and/or password entered.")
    http.Error(res, errBlank.Error(), http.StatusForbidden)
    return dsa.EmptyUser, errBlank
```

  -
    - Cannot input an existing username
    - Cannot input blank username or password
    - Confirm password by asking user to repeat entry

- Forms across the application have autocomplete turned off

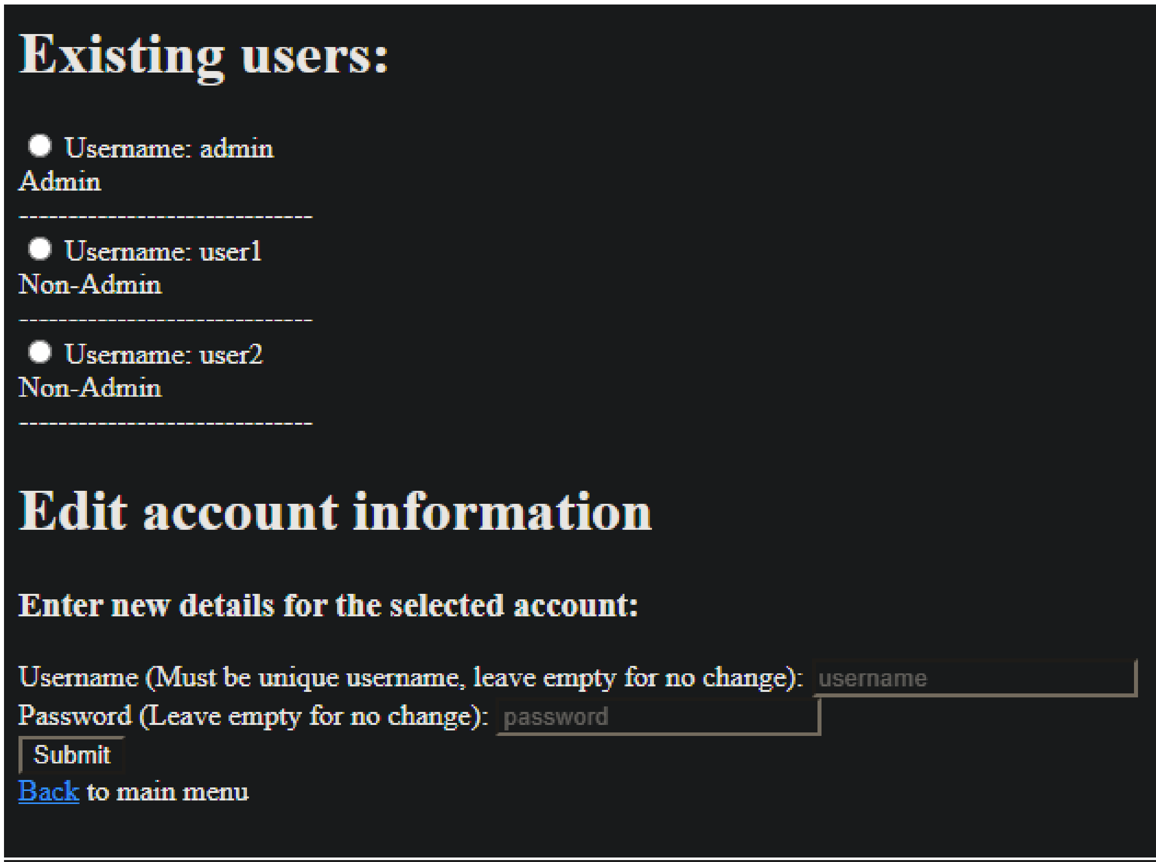


templates > addproducts.gohtml

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Add Products</title>
</head>
<body>
<h1>Add Products</h1>
<h3>Existing products: </h3>
{{range $index, $product := .}}
{{$product}} <br>
{{end}}

<form method="post" autocomplete="off">
<h3>Enter Product Name</h3>
    <label for ="productname">Product Name (Must be unique):</label>
    <input type="text" name="productname" placeholder="Product Name"><
    <input type="submit">
</form>

<a href="/addproducts">Add Products</a> <br>
<a href="/editproducts">Edit Products</a> <br>
<a href="/deleteproducts">Delete Products</a> <br>
<a href="/manprods">Back to Product Management Menu</a> <br>

</body>
</html>
```

▪

- Data Entry
  - Edit existing account (admin feature):



Existing users:

○ Username: admin
Admin

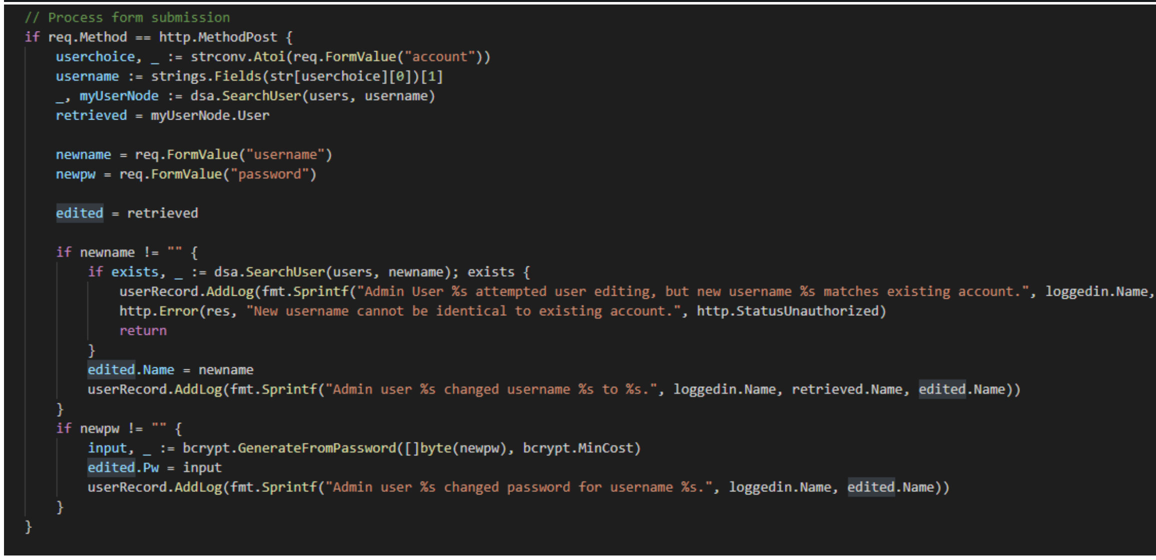
○ Username: user1
Non-Admin

○ Username: user2
Non-Admin

Edit account information

Enter new details for the selected account:

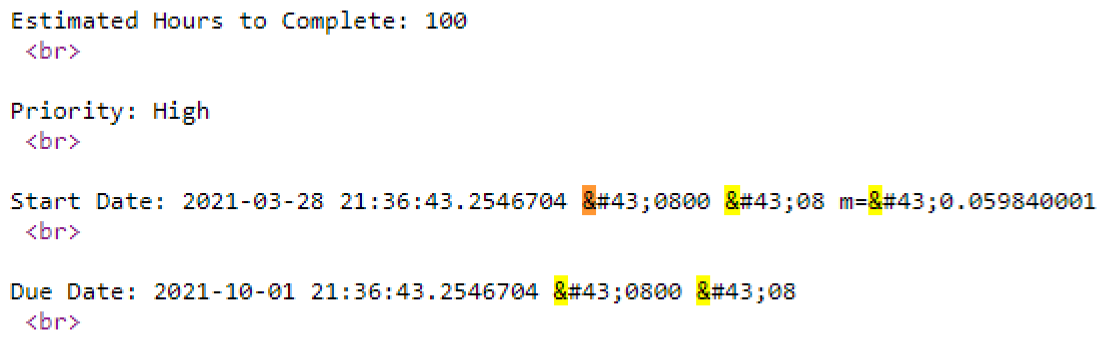
Username (Must be unique username, leave empty for no change): username
Password (Leave empty for no change): password
Submit
Back to main menu



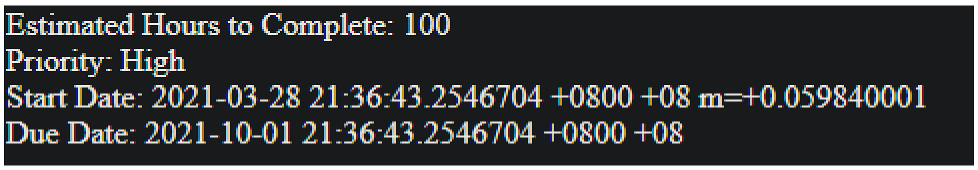
```
// Process form submission
if req.Method == http.MethodPost {
    userchoice, _ := strconv.Atoi(req.FormValue("account"))
    username := strings.Fields(str[userchoice][0])[1]
    _, myUserNode := dsa.SearchUser(users, username)
    retrieved = myUserNode.User

    newname = req.FormValue("username")
    newpw = req.FormValue("password")

    edited = retrieved

    if newname != "" {
        if exists, _ := dsa.SearchUser(users, newname); exists {
            userRecord.AddLog(fmt.Sprintf("Admin User %s attempted user editing, but new username %s matches existing account.", loggedin.Name,
            http.Error(res, "New username cannot be identical to existing account.", http.StatusUnauthorized)
            return
        }
        edited.Name = newname
        userRecord.AddLog(fmt.Sprintf("Admin user %s changed username %s to %s.", loggedin.Name, retrieved.Name, edited.Name))
    }
    if newpw != "" {
        input, _ := bcrypt.GenerateFromPassword([]byte(newpw), bcrypt.MinCost)
        edited.Pw = input
        userRecord.AddLog(fmt.Sprintf("Admin user %s changed password for username %s.", loggedin.Name, edited.Name))
    }
}
```

- Radio select buttons used in forms to facilitate data validation for fields where only one of a view options are valid
- Validation applied to ensure that username and/or password cannot be blank values
- Validation applied to ensure that username cannot match an existing user

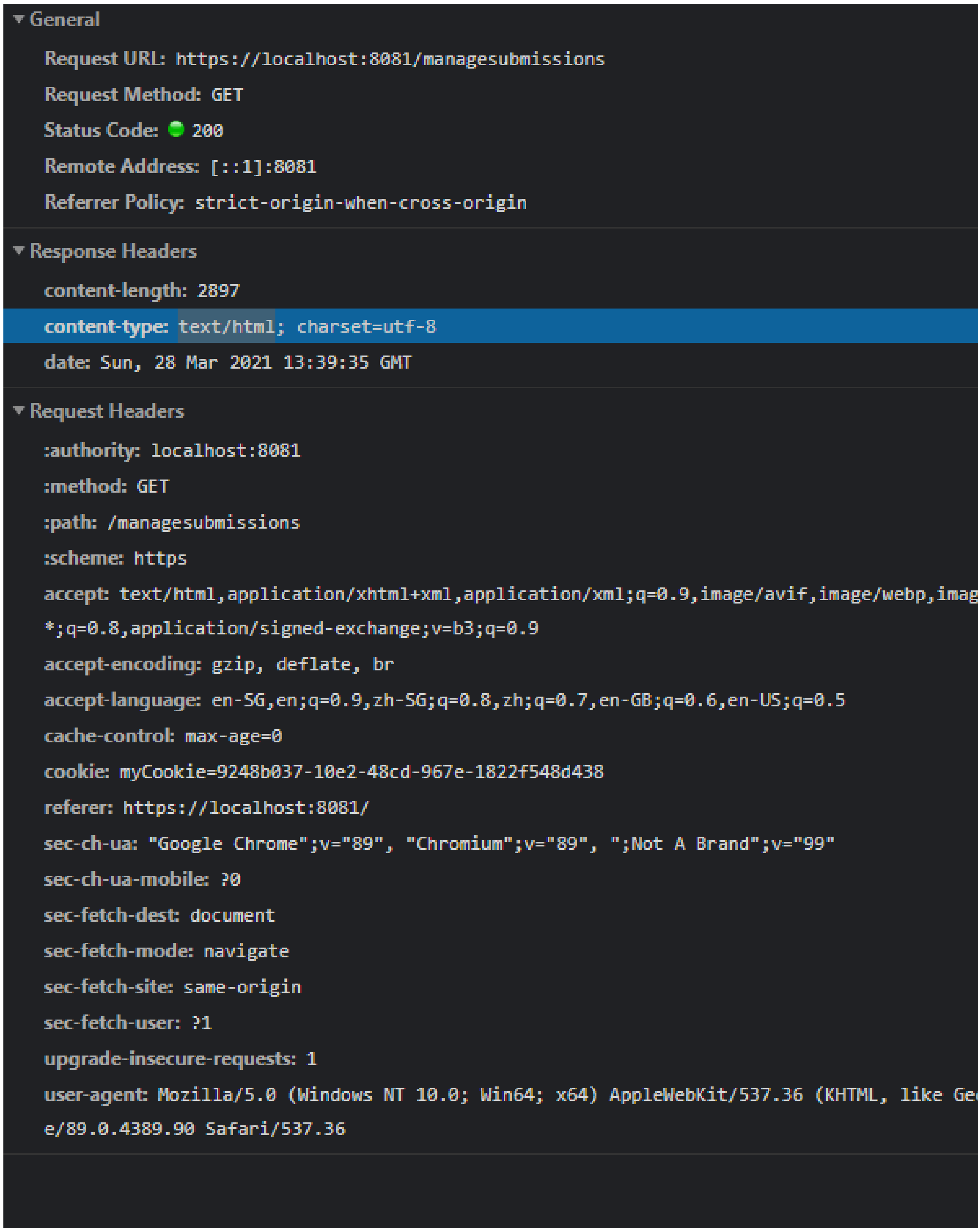
            - Account username and password can be edited, but admin status cannot be edited (to prevent case of a particular account being granted access to different set of features

- Output Encoding
    - Use of go html/template package to ensure that potential html tags inputted by user are correctly escaped (account creation and editing, product creation and editing, ticket submission)
        - 

```
Estimated Hours to Complete: 100
 <br>

Priority: High
 <br>

Start Date: 2021-03-28 21:36:43.2546704 &#43;0800 &#43;08 m=&#43;0.059840001
 <br>

Due Date: 2021-10-01 21:36:43.2546704 &#43;0800 &#43;08
 <br>
```

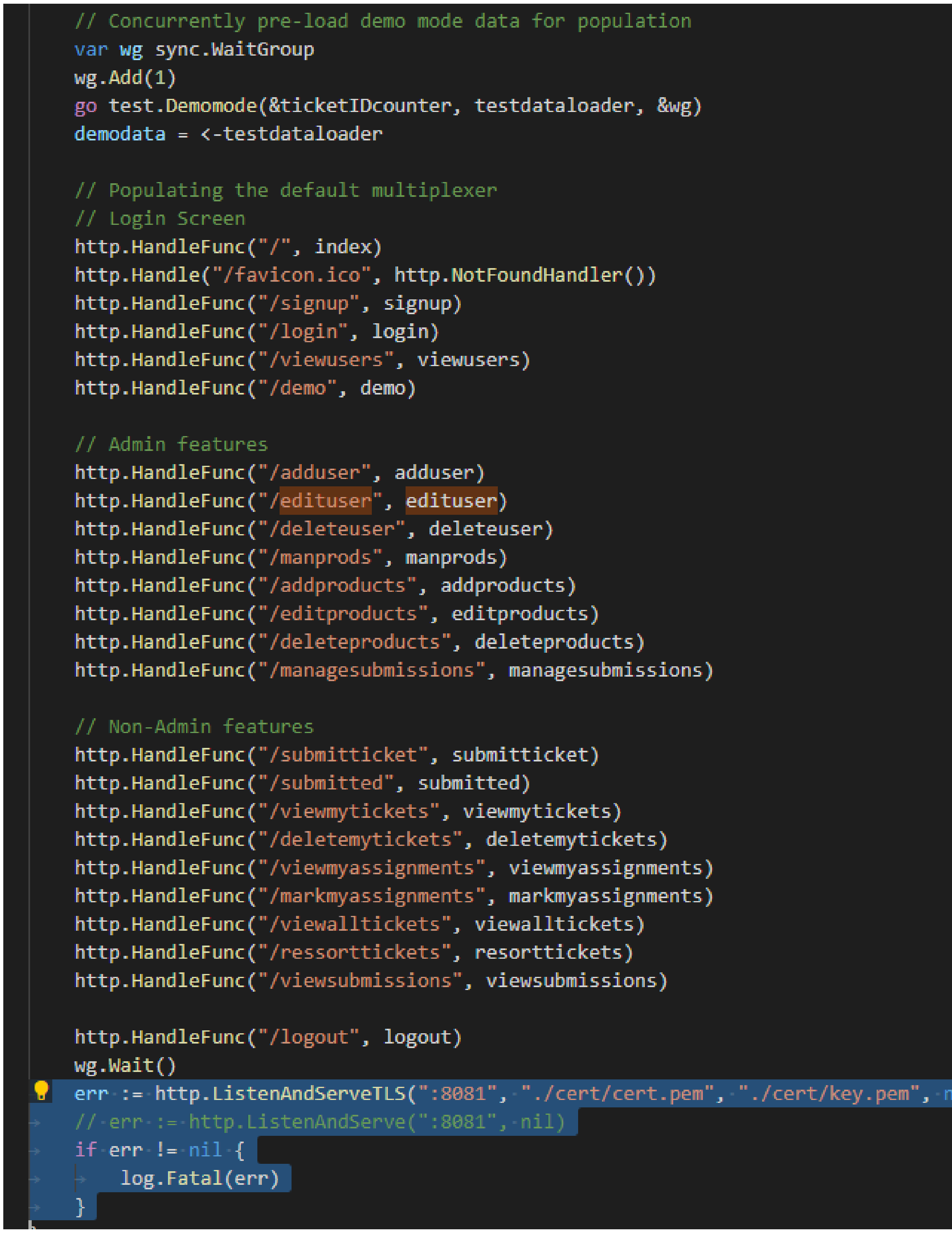
        - 

```
Estimated Hours to Complete: 100
Priority: High
Start Date: 2021-03-28 21:36:43.2546704 +0800 +08 m=+0.059840001
Due Date: 2021-10-01 21:36:43.2546704 +0800 +08
```



```
▼ General
    Request URL: https://localhost:8081/managesubmissions
    Request Method: GET
    Status Code: ● 200
    Remote Address: [::1]:8081
    Referrer Policy: strict-origin-when-cross-origin
▼ Response Headers
    content-length: 2897
    content-type: text/html; charset=utf-8
    date: Sun, 28 Mar 2021 13:39:35 GMT
▼ Request Headers
    :authority: localhost:8081
    :method: GET
    :path: /managesubmissions
    :scheme: https
    accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,imag
    *;q=0.8,application/signed-exchange;v=b3;q=0.9
    accept-encoding: gzip, deflate, br
    accept-language: en-SG,en;q=0.9,zh-SG;q=0.8,zh;q=0.7,en-GB;q=0.6,en-US;q=0.5
    cache-control: max-age=0
    cookie: myCookie=9248b037-10e2-48cd-967e-1822f548d438
    referer: https://localhost:8081/
    sec-ch-ua: "Google Chrome";v="89", "Chromium";v="89", ";Not A Brand";v="99"
    sec-ch-ua-mobile: ?0
    sec-fetch-dest: document
    sec-fetch-mode: navigate
    sec-fetch-site: same-origin
    sec-fetch-user: ?1
    upgrade-insecure-requests: 1
    user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Ge
    e/89.0.4389.90 Safari/537.36
```

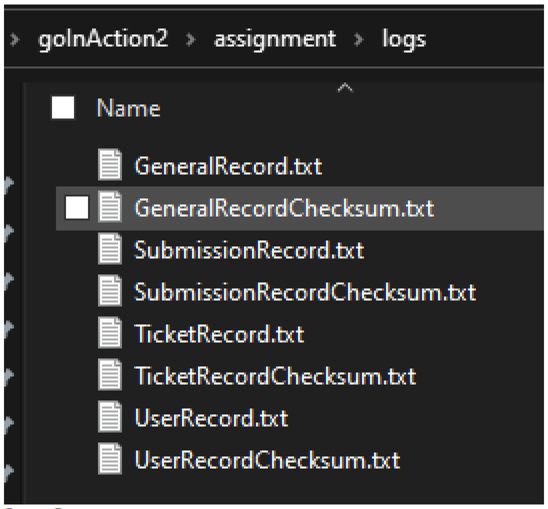
- Communications Security
    - HTTPS Implementation



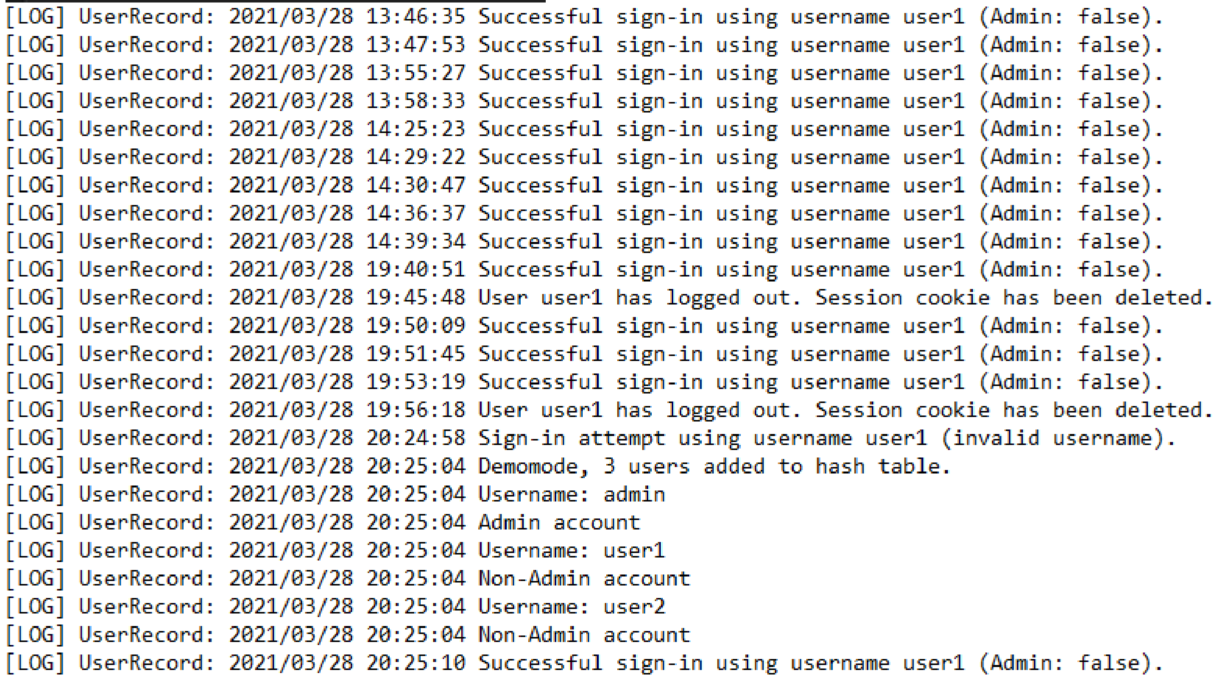
```go
    // Concurrently pre-load demo mode data for population
    var wg sync.WaitGroup
    wg.Add(1)
    go test.Demomode(&ticketIDcounter, testdataloader, &wg)
    demodata = <-testdataloader

    // Populating the default multiplexer
    // Login Screen
    http.HandleFunc("/", index)
    http.Handle("/favicon.ico", http.NotFoundHandler())
    http.HandleFunc("/signup", signup)
    http.HandleFunc("/login", login)
    http.HandleFunc("/viewusers", viewusers)
    http.HandleFunc("/demo", demo)

    // Admin features
    http.HandleFunc("/adduser", adduser)
    http.HandleFunc("/edituser", edituser)
    http.HandleFunc("/deleteuser", deleteuser)
    http.HandleFunc("/manprods", manprods)
    http.HandleFunc("/addproducts", addproducts)
    http.HandleFunc("/editproducts", editproducts)
    http.HandleFunc("/deleteproducts", deleteproducts)
    http.HandleFunc("/managesubmissions", managesubmissions)

    // Non-Admin features
    http.HandleFunc("/submitticket", submitticket)
    http.HandleFunc("/submitted", submitted)
    http.HandleFunc("/viewmytickets", viewmytickets)
    http.HandleFunc("/deletemytickets", deletemytickets)
    http.HandleFunc("/viewmyassignments", viewmyassignments)
    http.HandleFunc("/markmyassignments", markmyassignments)
    http.HandleFunc("/viewalltickets", viewalltickets)
    http.HandleFunc("/ressorttickets", resorttickets)
    http.HandleFunc("/viewsubmissions", viewsubmissions)

    http.HandleFunc("/logout", logout)
    wg.Wait()
    err := http.ListenAndServeTLS(":8081", "./cert/cert.pem", "./cert/key.pem",
    // err := http.ListenAndServe(":8081", nil)
    if err != nil {
        log.Fatal(err)
    }
```

▪

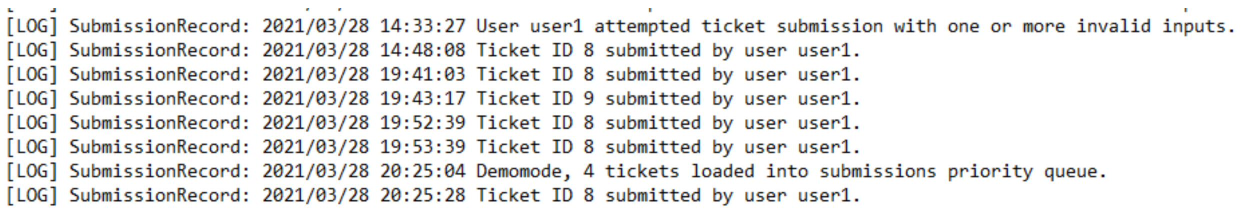
- Error Handling and Logging
    - Logger which records successful (and failed) login attempts



```
GeneralRecord.txt
GeneralRecordChecksum.txt
SubmissionRecord.txt
SubmissionRecordChecksum.txt
TicketRecord.txt
TicketRecordChecksum.txt
UserRecord.txt
UserRecordChecksum.txt
```

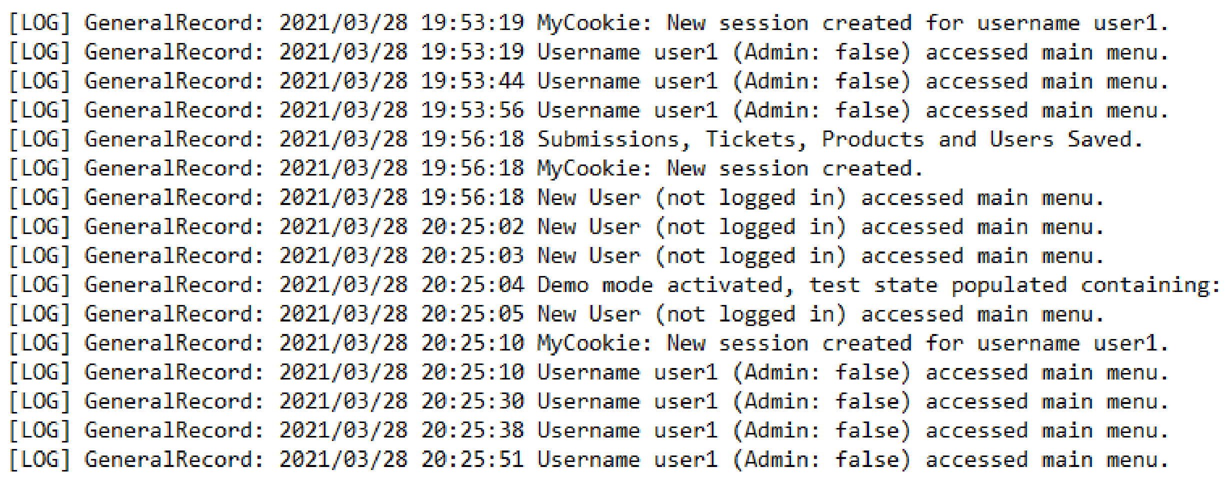
○ 

```
[LOG] UserRecord: 2021/03/28 13:46:35 Successful sign-in using username user1 (Admin: false).
[LOG] UserRecord: 2021/03/28 13:47:53 Successful sign-in using username user1 (Admin: false).
[LOG] UserRecord: 2021/03/28 13:55:27 Successful sign-in using username user1 (Admin: false).
[LOG] UserRecord: 2021/03/28 13:58:33 Successful sign-in using username user1 (Admin: false).
[LOG] UserRecord: 2021/03/28 14:25:23 Successful sign-in using username user1 (Admin: false).
[LOG] UserRecord: 2021/03/28 14:29:22 Successful sign-in using username user1 (Admin: false).
[LOG] UserRecord: 2021/03/28 14:30:47 Successful sign-in using username user1 (Admin: false).
[LOG] UserRecord: 2021/03/28 14:36:37 Successful sign-in using username user1 (Admin: false).
[LOG] UserRecord: 2021/03/28 14:39:34 Successful sign-in using username user1 (Admin: false).
[LOG] UserRecord: 2021/03/28 19:40:51 Successful sign-in using username user1 (Admin: false).
[LOG] UserRecord: 2021/03/28 19:45:48 User user1 has logged out. Session cookie has been deleted.
[LOG] UserRecord: 2021/03/28 19:50:09 Successful sign-in using username user1 (Admin: false).
[LOG] UserRecord: 2021/03/28 19:51:45 Successful sign-in using username user1 (Admin: false).
[LOG] UserRecord: 2021/03/28 19:53:19 Successful sign-in using username user1 (Admin: false).
[LOG] UserRecord: 2021/03/28 19:56:18 User user1 has logged out. Session cookie has been deleted.
[LOG] UserRecord: 2021/03/28 20:24:58 Sign-in attempt using username user1 (invalid username).
[LOG] UserRecord: 2021/03/28 20:25:04 Demomode, 3 users added to hash table.
[LOG] UserRecord: 2021/03/28 20:25:04 Username: admin
[LOG] UserRecord: 2021/03/28 20:25:04 Admin account
[LOG] UserRecord: 2021/03/28 20:25:04 Username: user1
[LOG] UserRecord: 2021/03/28 20:25:04 Non-Admin account
[LOG] UserRecord: 2021/03/28 20:25:04 Username: user2
[LOG] UserRecord: 2021/03/28 20:25:04 Non-Admin account
[LOG] UserRecord: 2021/03/28 20:25:10 Successful sign-in using username user1 (Admin: false).
```

○ 

```
[LOG] SubmissionRecord: 2021/03/28 14:33:27 User user1 attempted ticket submission with one or more invalid inputs.
[LOG] SubmissionRecord: 2021/03/28 14:48:08 Ticket ID 8 submitted by user user1.
[LOG] SubmissionRecord: 2021/03/28 19:41:03 Ticket ID 8 submitted by user user1.
[LOG] SubmissionRecord: 2021/03/28 19:43:17 Ticket ID 9 submitted by user user1.
[LOG] SubmissionRecord: 2021/03/28 19:52:39 Ticket ID 8 submitted by user user1.
[LOG] SubmissionRecord: 2021/03/28 19:53:39 Ticket ID 8 submitted by user user1.
[LOG] SubmissionRecord: 2021/03/28 20:25:04 Demomode, 4 tickets loaded into submissions priority queue.
[LOG] SubmissionRecord: 2021/03/28 20:25:28 Ticket ID 8 submitted by user user1.
```

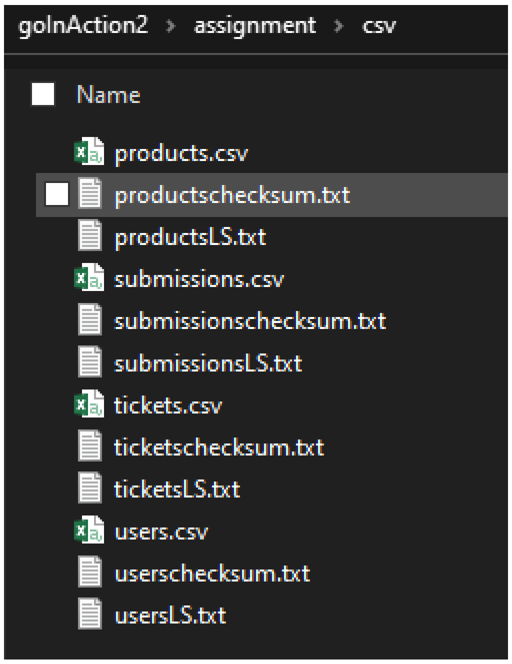
○



```
[LOG] GeneralRecord: 2021/03/28 19:53:19 MyCookie: New session created for username user1.
[LOG] GeneralRecord: 2021/03/28 19:53:19 Username user1 (Admin: false) accessed main menu.
[LOG] GeneralRecord: 2021/03/28 19:53:44 Username user1 (Admin: false) accessed main menu.
[LOG] GeneralRecord: 2021/03/28 19:53:56 Username user1 (Admin: false) accessed main menu.
[LOG] GeneralRecord: 2021/03/28 19:56:18 Submissions, Tickets, Products and Users Saved.
[LOG] GeneralRecord: 2021/03/28 19:56:18 MyCookie: New session created.
[LOG] GeneralRecord: 2021/03/28 19:56:18 New User (not logged in) accessed main menu.
[LOG] GeneralRecord: 2021/03/28 20:25:02 New User (not logged in) accessed main menu.
[LOG] GeneralRecord: 2021/03/28 20:25:03 New User (not logged in) accessed main menu.
[LOG] GeneralRecord: 2021/03/28 20:25:04 Demo mode activated, test state populated containing:
[LOG] GeneralRecord: 2021/03/28 20:25:05 New User (not logged in) accessed main menu.
[LOG] GeneralRecord: 2021/03/28 20:25:10 MyCookie: New session created for username user1.
[LOG] GeneralRecord: 2021/03/28 20:25:10 Username user1 (Admin: false) accessed main menu.
[LOG] GeneralRecord: 2021/03/28 20:25:30 Username user1 (Admin: false) accessed main menu.
[LOG] GeneralRecord: 2021/03/28 20:25:38 Username user1 (Admin: false) accessed main menu.
[LOG] GeneralRecord: 2021/03/28 20:25:51 Username user1 (Admin: false) accessed main menu.
```

  -
    - Logger is cryptographically hashed, with SHA256 checksum to verify against
    - Events logged include registering new user accounts (failed and successful)
    - Editing user accounts (failed and successful)
    - Deleting user accounts (failed and successful)
    - Login attempts and successful logins
    - Logouts
    - Ticket submissions and approvals/rejections
    - Cookie creations, and for which user
    - Implementation details found in hashlog source and documentation

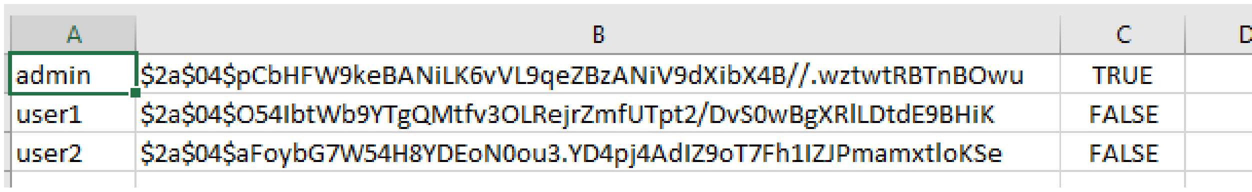
- Cryptographic implementations
  - Persistent storage implementation (CSV with SHA256 hash checking)
  - Only hashes of passwords saved
  - Hashed logs and hashed CSV storage are made concurrent safe – see hashcsv and hashlog packages for implementation details



goInAction2 › assignment › csv

Name
products.csv
productschecksum.txt
productsLS.txt
submissions.csv
submissionschecksum.txt
submissionsLS.txt
tickets.csv
ticketschecksum.txt
ticketsLS.txt
users.csv
userschecksum.txt
usersLS.txt

  -

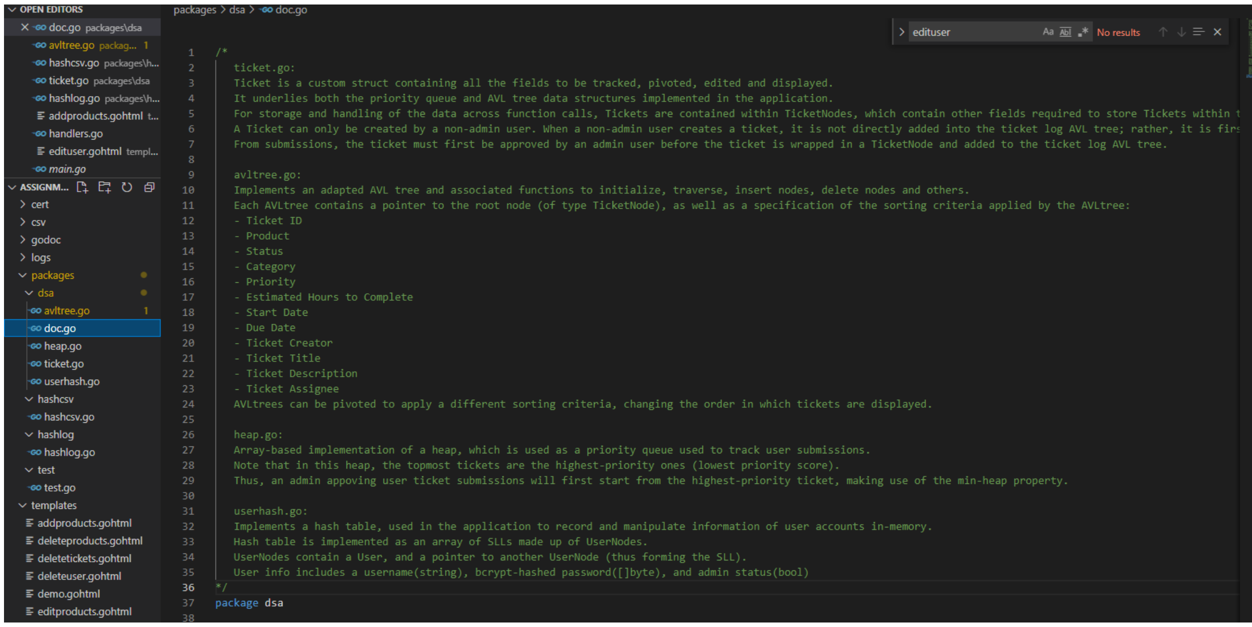


A	B	C	D
admin	$2a$04$pCbHFW9keBANiLK6vVL9qeZBzANiV9dXibX4B//.wztwtRBTnBOwu	TRUE	
user1	$2a$04$O54IbtWb9YTgQMtfv3OLRejrZmfUTpt2/DvS0wBgXRlLDtdE9BHiK	FALSE	
user2	$2a$04$aFoybG7W54H8YDEoN0ou3.YD4pj4AdIZ9oT7Fh1IZJPmamxtloKSe	FALSE	

- Other security-driven implementations
    - Cookie deleted upon logout

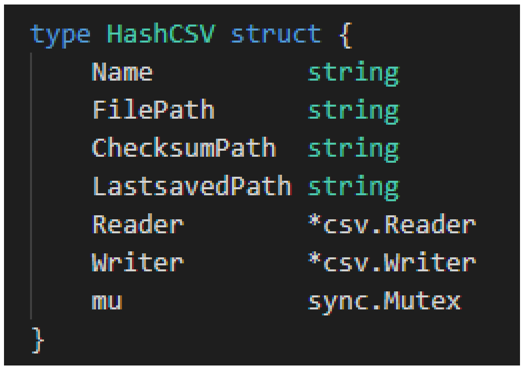
Idiomatic Go Techniques

- Go Documentation per convention (reference example for multi-file data structures and algorithms package)

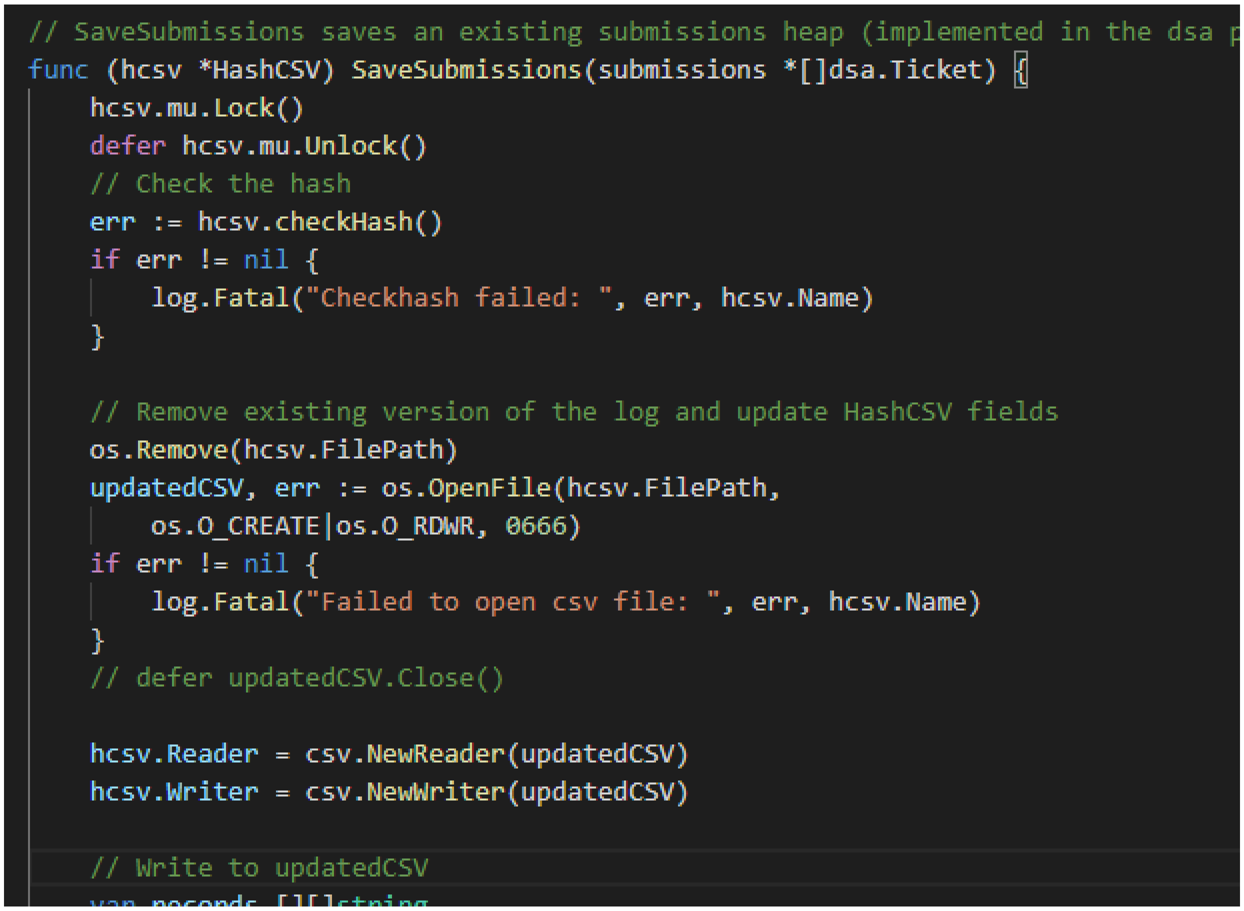


```go
/*
  ticket.go:
  Ticket is a custom struct containing all the fields to be tracked, pivoted, edited and displayed.
  It underlies both the priority queue and AVL tree data structures implemented in the application.
  For storage and handling of the data across function calls, Tickets are contained within TicketNodes, which contain other fields required to store Tickets within t
  A Ticket can only be created by a non-admin user. When a non-admin user creates a ticket, it is not directly added into the ticket log AVL tree; rather, it is firs
  From submissions, the ticket must first be approved by an admin user before the ticket is wrapped in a TicketNode and added to the ticket log AVL tree.

  avltree.go:
  Implements an adapted AVL tree and associated functions to initialize, traverse, insert nodes, delete nodes and others.
  Each AVLtree contains a pointer to the root node (of type TicketNode), as well as a specification of the sorting criteria applied by the AVLtree:
  - Ticket ID
  - Product
  - Status
  - Category
  - Priority
  - Estimated Hours to Complete
  - Start Date
  - Due Date
  - Ticket Creator
  - Ticket Title
  - Ticket Description
  - Ticket Assignee
  AVLtrees can be pivoted to apply a different sorting criteria, changing the order in which tickets are displayed.

  heap.go:
  Array-based implementation of a heap, which is used as a priority queue used to track user submissions.
  Note that in this heap, the topmost tickets are the highest-priority ones (lowest priority score).
  Thus, an admin appoving user ticket submissions will first start from the highest-priority ticket, making use of the min-heap property.

  userhash.go:
  Implements a hash table, used in the application to record and manipulate information of user accounts in-memory.
  Hash table is implemented as an array of SLLs made up of UserNodes.
  UserNodes contain a User, and a pointer to another UserNode (thus forming the SLL).
  User info includes a username(string), bcrypt-hashed password([]byte), and admin status(bool)
*/
package dsa
```

- Mutex hat included in struct

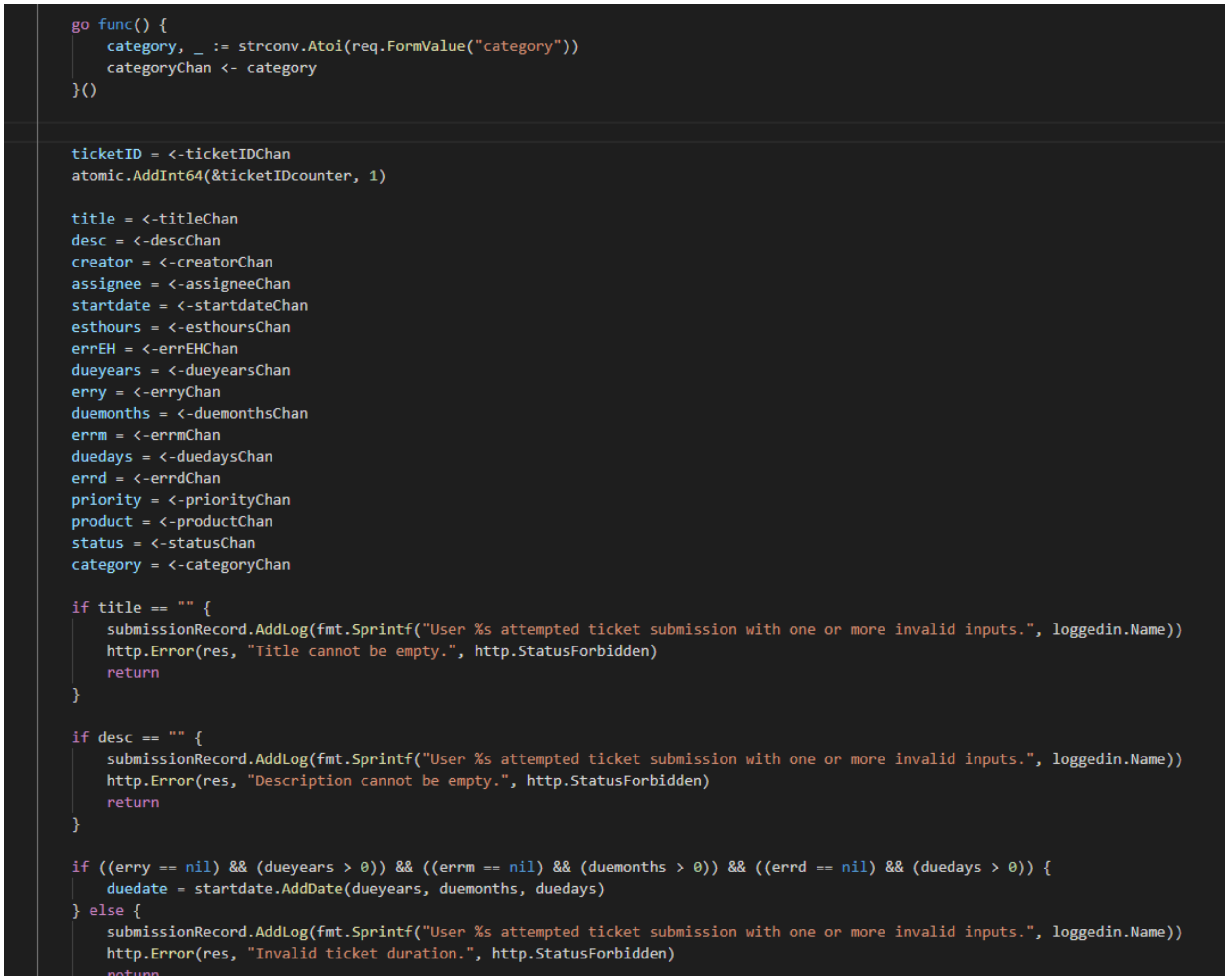


```go
type HashCSV struct {
    Name         string
    FilePath     string
    ChecksumPath string
    LastsavedPath string
    Reader       *csv.Reader
    Writer       *csv.Writer
    mu           sync.Mutex
}
```



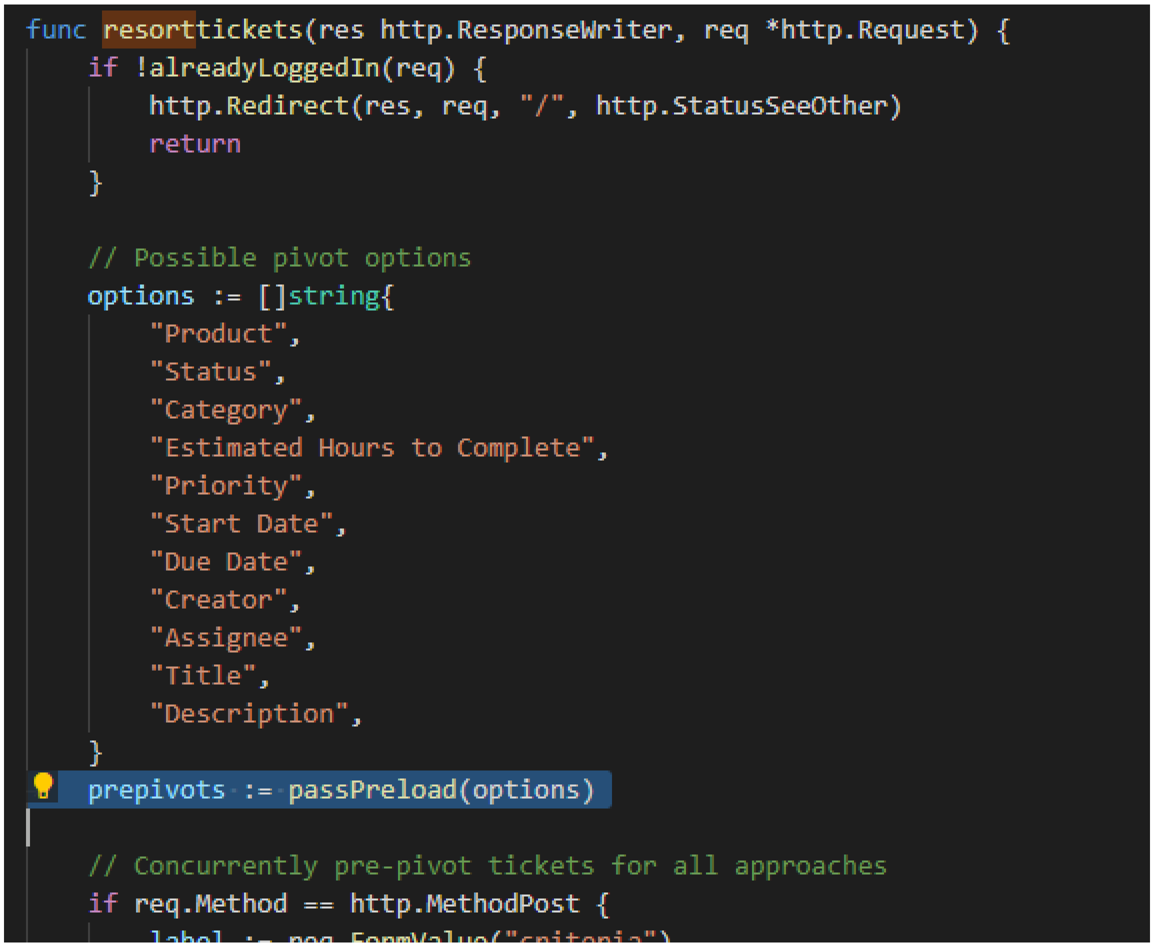
```go
// SaveSubmissions saves an existing submissions heap (implemented in the dsa p
func (hcsv *HashCSV) SaveSubmissions(submissions *[]dsa.Ticket) {
    hcsv.mu.Lock()
    defer hcsv.mu.Unlock()
    // Check the hash
    err := hcsv.checkHash()
    if err != nil {
        log.Fatal("Checkhash failed: ", err, hcsv.Name)
    }

    // Remove existing version of the log and update HashCSV fields
    os.Remove(hcsv.FilePath)
    updatedCSV, err := os.OpenFile(hcsv.FilePath,
        os.O_CREATE|os.O_RDWR, 0666)
    if err != nil {
        log.Fatal("Failed to open csv file: ", err, hcsv.Name)
    }
    // defer updatedCSV.Close()

    hcsv.Reader = csv.NewReader(updatedCSV)
    hcsv.Writer = csv.NewWriter(updatedCSV)

    // Write to updatedCSV
    var records [][]string
```

- Concurrent design patterns
  - Use of goroutines and channels to concurrently request multiple fields from HTML form



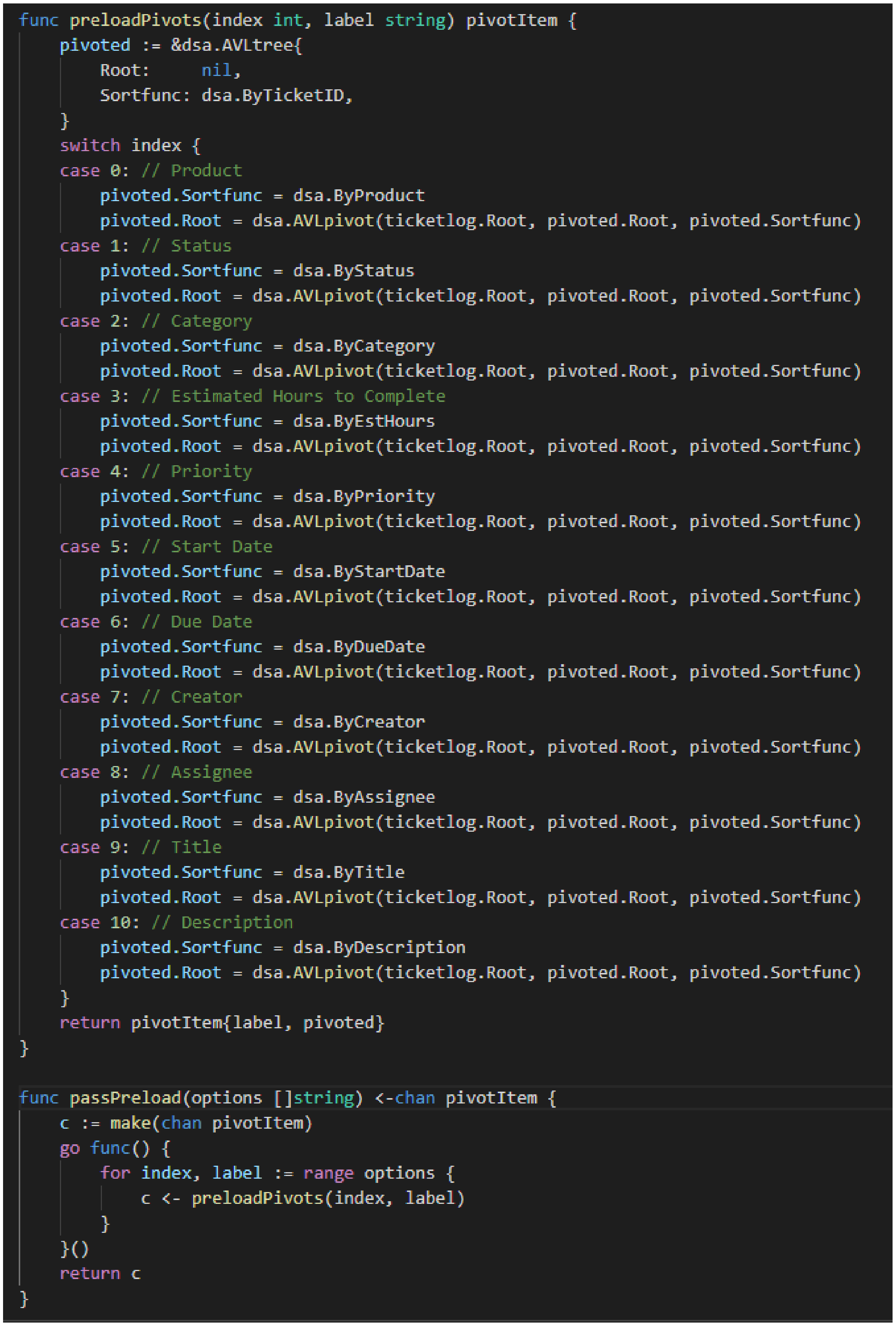
```go
go func() {
    category, _ := strconv.Atoi(req.FormValue("category"))
    categoryChan <- category
}()


ticketID = <-ticketIDChan
atomic.AddInt64(&ticketIDcounter, 1)

title = <-titleChan
desc = <-descChan
creator = <-creatorChan
assignee = <-assigneeChan
startdate = <-startdateChan
esthours = <-esthoursChan
errEH = <-errEHChan
dueyears = <-dueyearsChan
erry = <-erryChan
duemonths = <-duemonthsChan
errm = <-errmChan
duedays = <-duedaysChan
errd = <-errdChan
priority = <-priorityChan
product = <-productChan
status = <-statusChan
category = <-categoryChan

if title == "" {
    submissionRecord.AddLog(fmt.Sprintf("User %s attempted ticket submission with one or more invalid inputs.", loggedin.Name))
    http.Error(res, "Title cannot be empty.", http.StatusForbidden)
    return
}

if desc == "" {
    submissionRecord.AddLog(fmt.Sprintf("User %s attempted ticket submission with one or more invalid inputs.", loggedin.Name))
    http.Error(res, "Description cannot be empty.", http.StatusForbidden)
    return
}

if ((erry == nil) && (dueyears > 0)) && ((errm == nil) && (duemonths > 0)) && ((errd == nil) && (duedays > 0)) {
    duedate = startdate.AddDate(dueyears, duemonths, duedays)
} else {
    submissionRecord.AddLog(fmt.Sprintf("User %s attempted ticket submission with one or more invalid inputs.", loggedin.Name))
    http.Error(res, "Invalid ticket duration.", http.StatusForbidden)
```

- Implementation of generator pattern to concurrently pre-load resorted views of the tickets AVL tree

- 

```go
func resorttickets(res http.ResponseWriter, req *http.Request) {
    if !alreadyLoggedIn(req) {
        http.Redirect(res, req, "/", http.StatusSeeOther)
        return
    }

    // Possible pivot options
    options := []string{
        "Product",
        "Status",
        "Category",
        "Estimated Hours to Complete",
        "Priority",
        "Start Date",
        "Due Date",
        "Creator",
        "Assignee",
        "Title",
        "Description",
    }
    prepivots := passPreload(options)

    // Concurrently pre-pivot tickets for all approaches
    if req.Method == http.MethodPost {
        label := req.FormValue("criteria")
```

○



```go
func preloadPivots(index int, label string) pivotItem {
    pivoted := &dsa.AVLtree{
        Root:     nil,
        Sortfunc: dsa.ByTicketID,
    }
    switch index {
    case 0: // Product
        pivoted.Sortfunc = dsa.ByProduct
        pivoted.Root = dsa.AVLpivot(ticketlog.Root, pivoted.Root, pivoted.Sortfunc)
    case 1: // Status
        pivoted.Sortfunc = dsa.ByStatus
        pivoted.Root = dsa.AVLpivot(ticketlog.Root, pivoted.Root, pivoted.Sortfunc)
    case 2: // Category
        pivoted.Sortfunc = dsa.ByCategory
        pivoted.Root = dsa.AVLpivot(ticketlog.Root, pivoted.Root, pivoted.Sortfunc)
    case 3: // Estimated Hours to Complete
        pivoted.Sortfunc = dsa.ByEstHours
        pivoted.Root = dsa.AVLpivot(ticketlog.Root, pivoted.Root, pivoted.Sortfunc)
    case 4: // Priority
        pivoted.Sortfunc = dsa.ByPriority
        pivoted.Root = dsa.AVLpivot(ticketlog.Root, pivoted.Root, pivoted.Sortfunc)
    case 5: // Start Date
        pivoted.Sortfunc = dsa.ByStartDate
        pivoted.Root = dsa.AVLpivot(ticketlog.Root, pivoted.Root, pivoted.Sortfunc)
    case 6: // Due Date
        pivoted.Sortfunc = dsa.ByDueDate
        pivoted.Root = dsa.AVLpivot(ticketlog.Root, pivoted.Root, pivoted.Sortfunc)
    case 7: // Creator
        pivoted.Sortfunc = dsa.ByCreator
        pivoted.Root = dsa.AVLpivot(ticketlog.Root, pivoted.Root, pivoted.Sortfunc)
    case 8: // Assignee
        pivoted.Sortfunc = dsa.ByAssignee
        pivoted.Root = dsa.AVLpivot(ticketlog.Root, pivoted.Root, pivoted.Sortfunc)
    case 9: // Title
        pivoted.Sortfunc = dsa.ByTitle
        pivoted.Root = dsa.AVLpivot(ticketlog.Root, pivoted.Root, pivoted.Sortfunc)
    case 10: // Description
        pivoted.Sortfunc = dsa.ByDescription
        pivoted.Root = dsa.AVLpivot(ticketlog.Root, pivoted.Root, pivoted.Sortfunc)
    }
    return pivotItem{label, pivoted}
}

func passPreload(options []string) <-chan pivotItem {
    c := make(chan pivotItem)
    go func() {
        for index, label := range options {
            c <- preloadPivots(index, label)
        }
    }()
    return c
}
```

Documentation of application (Appendix)

Godoc pages for the packages implemented by me for this application, can be found in the ./godoc folder.

Note: In demo mode, user accounts are

- User1 (password: user1)
- User2 (password: user2)
- Admin (password: admin)