### **Go In Action II Assignment**

Author: Yeoh Aik Huang Alvin / [contact@alvinyeoh.com](mailto:contact@alvinyeoh.com)

1. **Comments & Instructions**

* The app is configured to be accessible at [**https://localhost:5221**](https://localhost:5221) by default. The hostname and port can be changed in the web package’s config file.
* The app is configured by default to connect to a live MySQL database, loads frontend dependencies (js/css) from CDNs and also fetches data from a web api so an internet connection is required. If desired, the default database connection configurations can be modified to connect to a different database in the ***.env*** file in the root directory.
* The app initializes the system with data fetched from the database by default.
* You can reset the database and re-seed test data by setting **resetAndSeedDB** to **true** in the clinic package’s ***config.go*** file and running the app. Seeding of test data is performed in the clinicpackage’s ***helper.go*** file via goroutines. You can also use this to quickly set up a custom database.
* If **resetDB** is set to **true** and **resetAndSeedDB** to **false**, a clean system with no data will be initialized.
* If both **resetDB** & **resetAndSeedDB** are set to false, the app will initialize with the existing data in the database. This is the default setting when the assignment is submitted.
* A staff / admin is a Patient whose Id is inside the Admins slice. Admins slice is loaded via **seedAdmins** function which in turns gets it from the ADMIN\_IDS field in the ***.env*** file.
* The test accounts that were seeded automatically are listed below. As mentioned before, they can be re-seeded by setting **resetAndSeedDB** to **true** and running the app. The default password of seeded test accounts are “**12345678**”. This is also declared in the ***.env*** file.

|  |  |
| --- | --- |
| **Username (NRIC)** | **Is Admin** |
| S1111111B | No |
| S2222222C | No |
| S3333333D | No |
| S4444444D | No |
| S5555555E | No |
| S6666666F | No |
| S7777777G | No |
| S0000000A | Yes |
| S1234567A | Yes |
| S7654321A | Yes |
| S8888888A | Yes |
| S9999999A | Yes |

1. **Secure software development techniques applied**

User inputs are all processed by the handler functions inside the ***web*** package.

**GET vs POST Requests**

All requests that change data on the server (create/update/delete) are sent via POST requests to prevent cross-site request forgery (CSRF).

E.g. If an endpoint like <https://localhost:5221/admin/appointment/edit?apptId=1008&action=cancel> accepts a GET request to delete an appointment of ID 1008, an authenticated user can be tricked into executing it either directly accessing it accidentally or unknowingly via an iframe embed or javascript request.

**Cookie Settings**

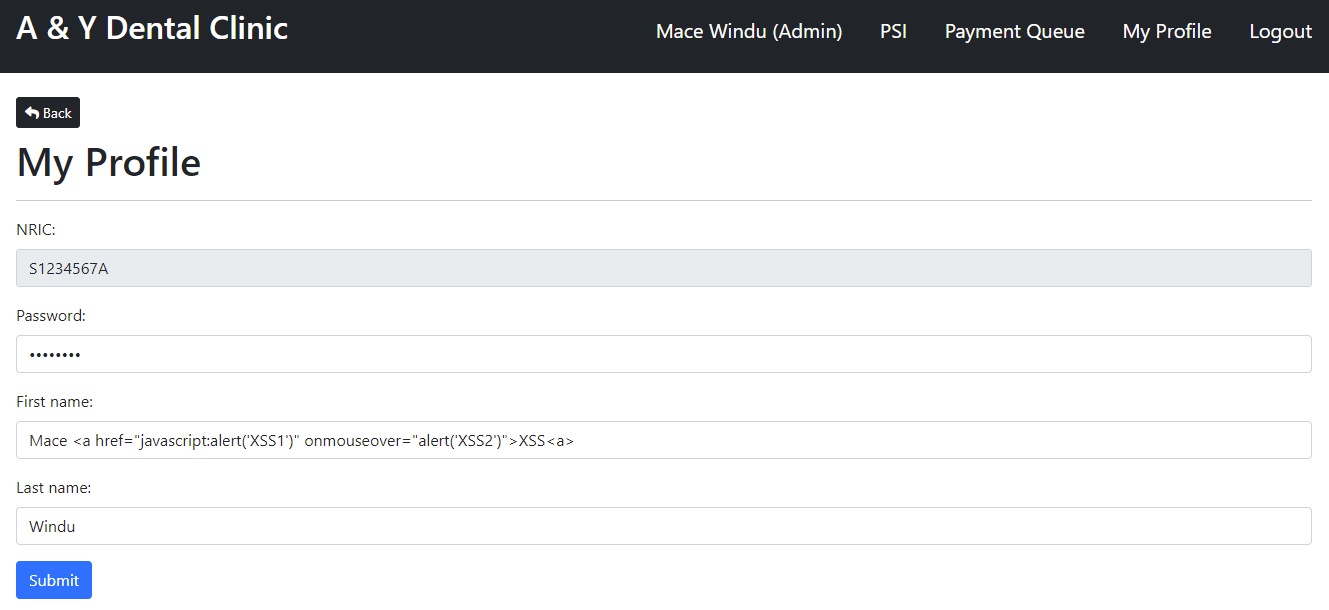
Cookies are created with the **SameSite** attribute set to 3 (Strict) and **Secure**. SameSite value being strict also helps prevent CSRF as it prevents cross-site requests. However, as cookies are stored client side, the enforcement depends on browser compatibility. The **HttpOnly** attribute restricts the cookie from being accessed by client-side APIs, such as JavaScript.

**Input Validation & Sanitization**

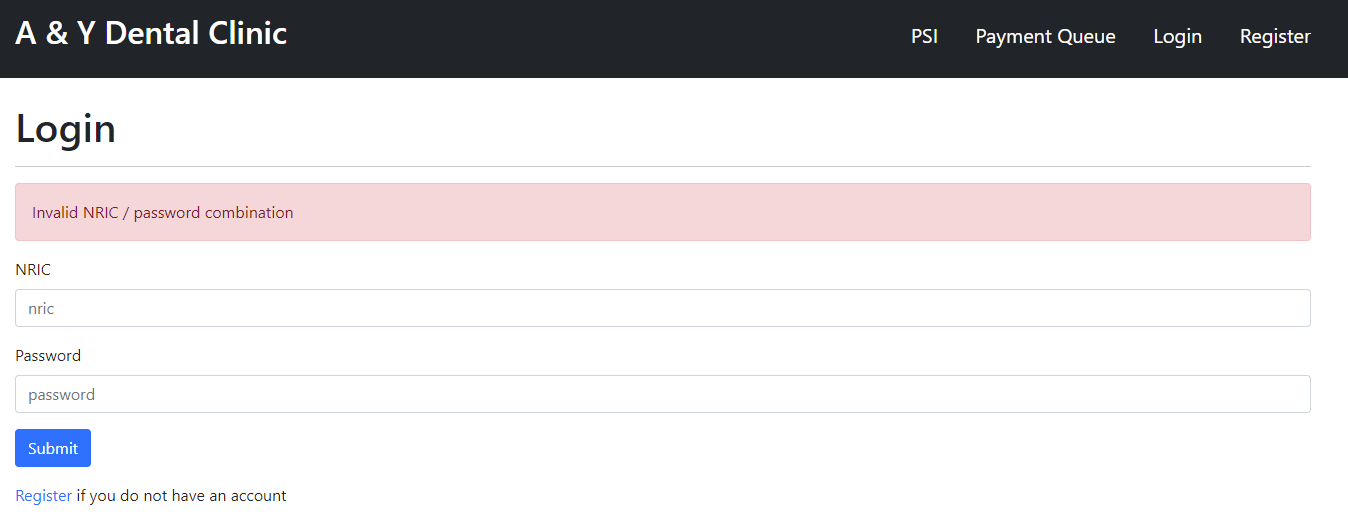
Only user registration and profile update consists of user inputs that are used as output. For the purpose of the assignment, external libraries are used sparingly.

**Register / Profile Update / Login**

* The **bluemonday** library is used to sanitize username (NRIC), firstname and lastname by stripping away tags to protect the application (app) from XSS. The tags shown below under first name will be stripped away and left with spaces which will be trimmed away via further sanitization resulting in the first name being just “Mace XSS”.

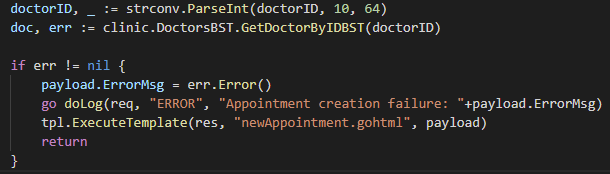


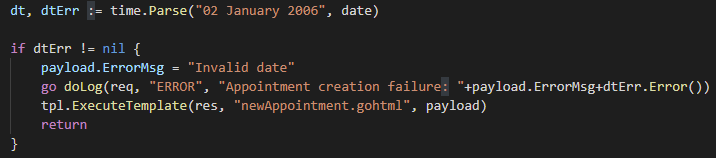
* After sanitization, I perform trimming of trailing and leading space. Username is transformed to all uppercase to standardize the attribute.
* The validity of user inputs are then checked for the following and returns the appropriate error messages if any fails.
  + Empty string (except for password e.g. space is a character)
  + Password length meeting the MinPasswordLength policy (default: 8)
  + NRIC validity
    - Currently, it only checks for length of 9
    - To enable the actual NRIC validity check which compares the NRIC against a checksum, set the **strictNRIC** attribute in ***clinic/config.go*** to true
  + Existing user (NRIC)
* When performing login, if either an NRIC can’t be found or Password is incorrect, a generic error message “**invalid NRIC / password combination**” to not alert the user which is the wrong input to help mitigate brute force attacks.

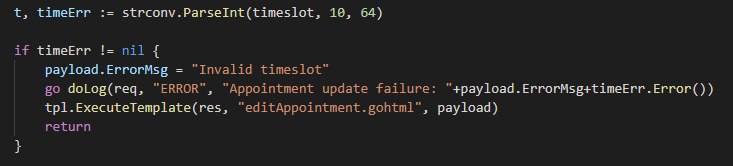


**String parsing**

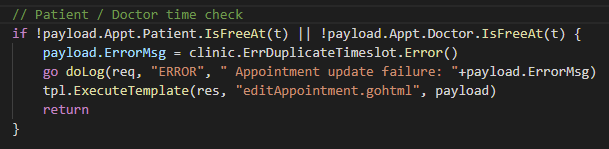
* Most of the app’s other handler functions accept inputs which are parsed from string to their appropriate types (e.g. int64 / time) and then checked for errors via Go’s strconv functions.
  + E.g. when making a new appointment request, **doctor id**, **date** and **time** are user inputs that are parsed and checked for errors.



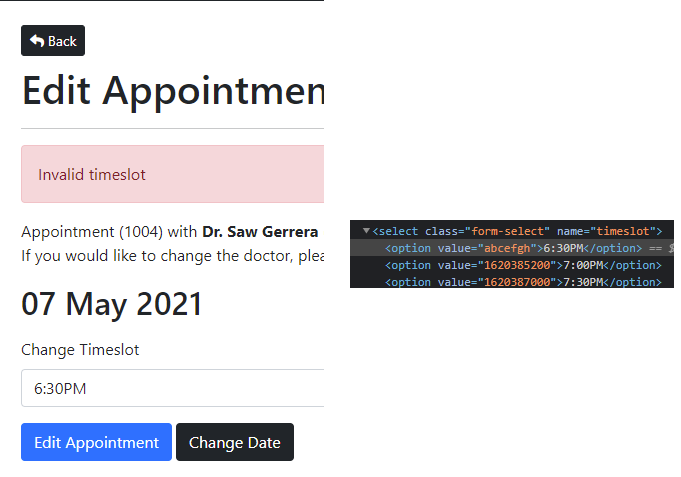




* + After type conversion checks have passed successfully, app specific checks are also performed such as checking if the doctor id really exists (shown above from **GetDoctorByIDBST**) and if the date given is an actual date a doctor and patient is actually able to make an appointment for (shown below) as it’s trivial to forge the payload of a POST request.



* + As seen below, when I modify the time value client side and submit, I’ll get the “**Invalid timeslot**” error.

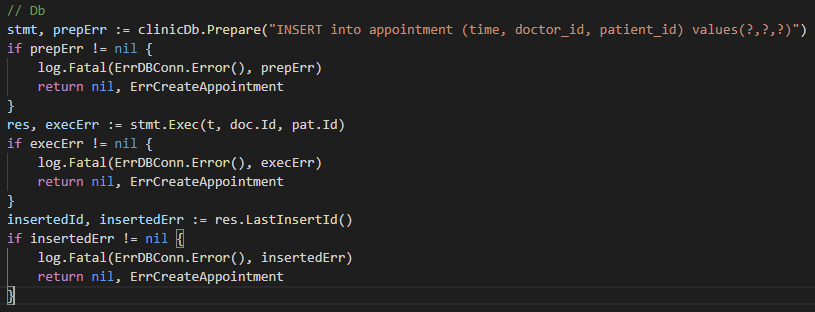


**Context Awareness**

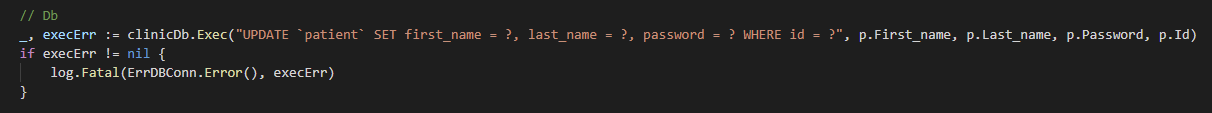
* Go’s template engine is used for all pages of the app so it’s another layer of protection against XSS via input users that are used as output in the HTML.
  + When I set my firstname as “Mace<script>alert('hi')</script>” without sanitization, it gets output as “Mace&lt;script&gt;alert('hi')&lt;/script&gt;” by the template engine instead and the script is not executed.

**SQL Injection**

* Sql statements are mostly found in the ***clinic*** package. Any sql statement that contains dynamic parameters uses prepared statements.
* E.g. Appointment creation



* E.g. User profile update
  + password contains the binary data of the hashed password and not the original password in plain text

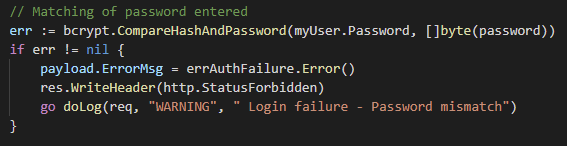


**Communication Security**

The app uses HTTPS to encrypt and protect communications. However, it uses a self signed key for the purpose of the assignment as opposed to one from a Certificate Authority so we’ll be presented with a “the certificate cannot be trusted error message”. The certs are located in the ***.cert*** hidden directory which is also a directory that has been excluded from being tracked and committed to git repository by editing ***.gitignore***.

**Error Handling & Logging**

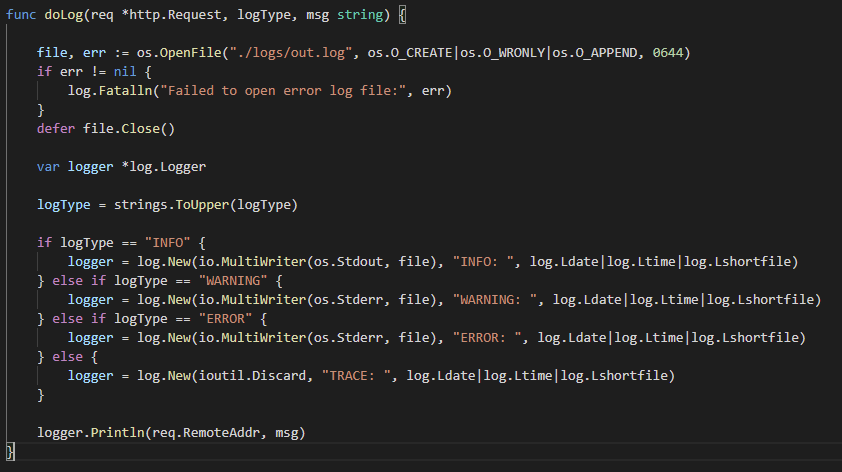
During any checks, if any error is encountered, the user is directed to either another page or the same page with the appropriate error message as well as having the action logged. The logs contain a more detailed description of the error as compared to the error message presented to the user to assist in debugging or identifying potential malicious actions. No sensitive information like passwords like displayed to a user or stored in logs.



For the example above, if the password entered is different from what the app has in record, the user will be presented with the error message “**invalid NRIC / password combination**” and it’ll be logged to the server log files in an entry shown below.

****

Logging in the ***web*** package is performed through the custom **doLog** function in **helper.go**.

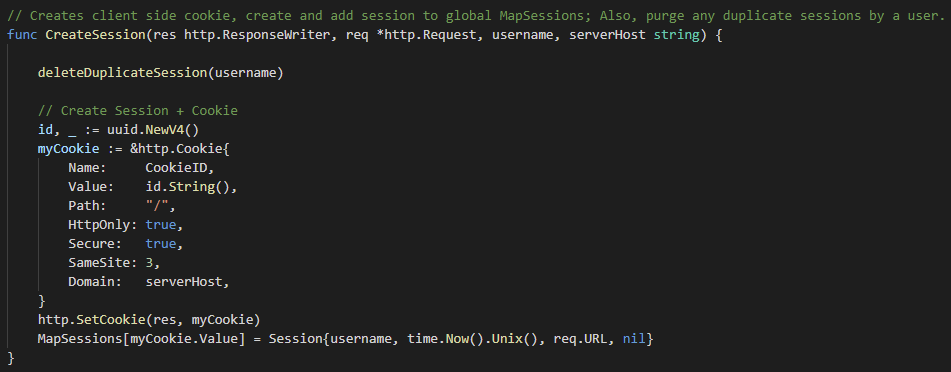


**Session Management**

Session is handled by the ***session*** package which creates a cookie in the **CreateSession** function. The attributes HttpOnly, Secure and SameSite have been covered prior so I’ll skip those.

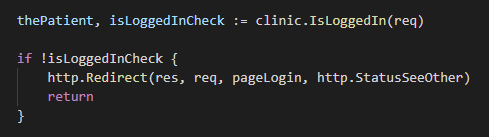
**UUID**

The value that’s stored is a **universally unique identifier** (UUID) string for the user that’s generated from the [**UUID**](https://github.com/satori/go.uuid) package. The UUID value for the user will then be stored server side in a map with the UUID as the key to a Session item containing the user’s username (NRIC), the current time and request URL.

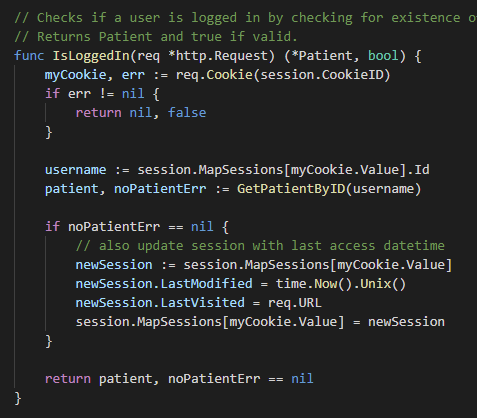
****

**Authentication**

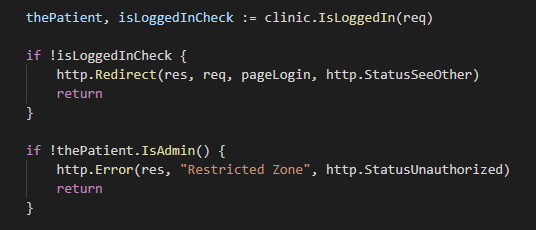
The app has 3 different access levels. Public with no requirements, members’ section where a user has to be logged in and staff section where the user that is logged in is identified as an admin (staff). All non-public pages will first check if the user is logged in via the IsLoggedIn function which returns both a Patient item and error. If the error isn’t nil, the user is redirected back to the public login page.



The IsLoggedIn function does this by checking if the client has a cookie, retrieves the value of it and tries to map it to our MapSessions global variable to retrieve the logged in user’s identity. It also updates the session item with the user’s last request time and page.

****

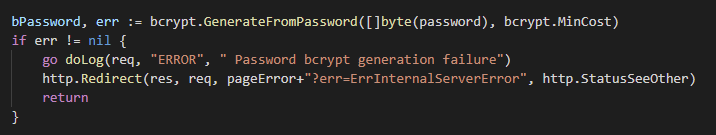
Similarly, when a user attempts to visit a staff restricted page, IsLoggedIn is first used to check that the user is a valid logged in then proceeds to check if he / she is an admin via the **IsAdmin** method of Patient and returning 401 error.

****

**Password Hashing**

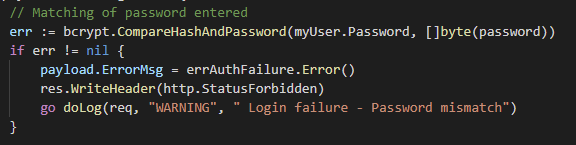
The life cycle of a password when registering or updating of profile in the app is as follows:

1. Sent encrypted over HTTPS from client to server.
2. Decrypted on the server side to original text.
3. Hashing performed via [**bcrypt**](https://pkg.go.dev/golang.org/x/crypto/bcrypt) library’s GenerateFromPassword function (show below).
4. Hash value of the original password is stored in the user's corresponding Patient item’s password field and also inserted / updated to the database as binary strings. The server does not know what is the original password in plain text from this point onwards nor stores that information anywhere.



**Login Check**

When a user performs login, the password provided is transferred from client to server via the same mechanism as before and a hash of the provided password is once again generated. This time the hashed value of the provided password is compared against the previously stored hashed password belonging to the user and if it matches, the user is logged in.

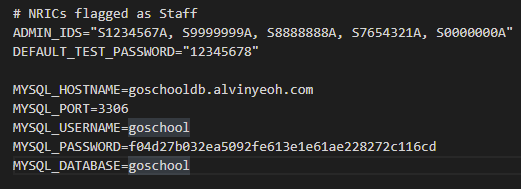


Any password mismatch failures are also logged.

**Sensitive App Data**

Secrets are stored in a ***.env*** file which is parsed and handled by the [**GoDotEnv**](https://github.com/joho/godotenv) library. It allows us to separate sensitive information from the source code. The file itself is hidden by default and also excluded from being committed to repositories by configuring ***.gitignore***.

Information inside my app’s .env file includes the csv list of ids recognized as staff, default password for test accounts and database connection information.



They can be accessed via the **os.Getenv** function as seen in the clinic package’s config.go file where I get the database connection information.



1. **Idiomatic Go techniques applied**

**Formatting**

Formatting of the app's source code was handled automatically by Visual Studio Code’s [**gopls**](https://code.visualstudio.com/docs/languages/go#_formatting) extension by Google instead of [**gofmt**](https://golang.org/pkg/cmd/gofmt/) or [**gofumpt**](https://github.com/mvdan/gofumpt).

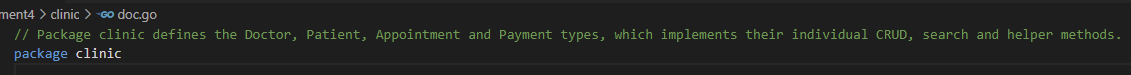
**Comments**

All packages and exported names have comments preceding them. Package names are all lower case single words and are of the same name as their containing directory. No underscore is used anywhere in the app for naming.

For packages with just 1 file like ***psi*** package, the package comment is before the package declaration in the sole file.

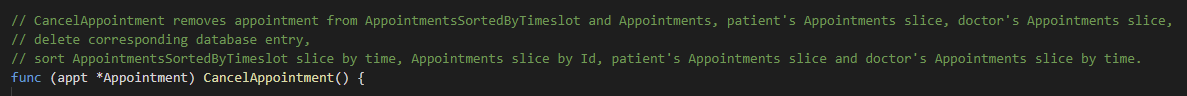


For packages with multiple files like ***web*** or ***clinic*** packages, their package commentary is located inside **doc.go**.



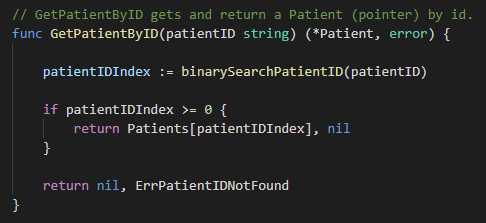
All exported functions begin with the name being declared and describes what they are for.





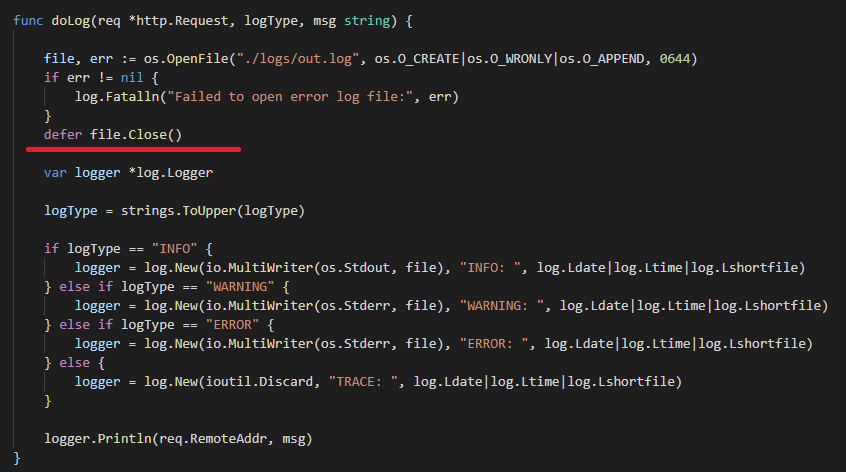
**Function Return**

Most functions in the app return both an intended result alongside an error which can be nil if there isn’t any error. As seen below, if binarySearchPatientID returns a value bigger than or equal to 0, we return the patient item and nil for the error. However, if it’s below 0, an **ErrPatientIDNotFound** error is returned alongside a nil pointer to a patient.

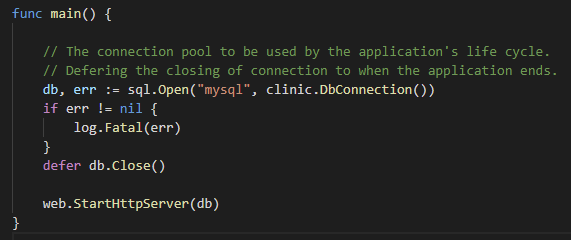
****

**Defer**

Defer is used in the **doLog** function to call **file.close** to ensure the file is closed before exiting the function.

****

It’s also used in package main right after creating a database connection so if the program is stopped, the database connection will be closed before exiting.

****

1. **Godoc Generated Documentation**

# **assignment4**

Package main kickstarts the application by calling StartHttpServer function from web package. It also creates a pooled mysql database connection that is passed to StartHttpServer and used in the rest of the program.

## Subdirectories

|  |  |
| --- | --- |
| **Name** | **Synopsis** |
| [..](http://localhost:8080/pkg/) | |
| [clinic](http://localhost:8080/pkg/assignment4/clinic/) | Package clinic defines the Doctor, Patient, Appointment and Payment types, which implements their individual CRUD, search and helper methods. |
| [psi](http://localhost:8080/pkg/assignment4/psi/) | Package psi defines the PSI type and provide implementation for fetching the 24H average Pollutant Standards Index (PSI) value. |
| [session](http://localhost:8080/pkg/assignment4/session/) | Package session defines the Session and Notification types; It provides implementation for creating and deleting of server side session, client side cookie and use of notification inside session to pass messages between http request. |
| [web](http://localhost:8080/pkg/assignment4/web/) | Package web defines route constants and provides the implementation for http handlers and server. |

# **Package clinic**

import "assignment4/clinic"

[Overview](http://localhost:8080/pkg/assignment4/clinic/#pkg-overview)

[Index](http://localhost:8080/pkg/assignment4/clinic/#pkg-index)

## Overview ▾

Package clinic defines the Doctor, Patient, Appointment and Payment types, which implements their individual CRUD, search and helper methods.

## Index ▾

[Constants](http://localhost:8080/pkg/assignment4/clinic/#pkg-constants)

[Variables](http://localhost:8080/pkg/assignment4/clinic/#pkg-variables)

[func BinarySearchApptID(apptID int64) int](http://localhost:8080/pkg/assignment4/clinic/#BinarySearchApptID)

[func BinarySearchApptTime(time int64) int](http://localhost:8080/pkg/assignment4/clinic/#BinarySearchApptTime)

[func DbConnection() string](http://localhost:8080/pkg/assignment4/clinic/#DbConnection)

[func GetAvailableTimeslot(dt int64, apptsToExclude []\*Appointment) []int64](http://localhost:8080/pkg/assignment4/clinic/#GetAvailableTimeslot)

[func IsApptTimeValid(t int64) (bool, error)](http://localhost:8080/pkg/assignment4/clinic/#IsApptTimeValid)

[func IsNRICValid(nric string) bool](http://localhost:8080/pkg/assignment4/clinic/#IsNRICValid)

[func IsThereTimeslot(dt int64, pat \*Patient, doc \*Doctor) (bool, error)](http://localhost:8080/pkg/assignment4/clinic/#IsThereTimeslot)

[func SeedData()](http://localhost:8080/pkg/assignment4/clinic/#SeedData)

[func SetDb(myDb \*sql.DB)](http://localhost:8080/pkg/assignment4/clinic/#SetDb)

[type Appointment](http://localhost:8080/pkg/assignment4/clinic/#Appointment)

[func MakeAppointment(t int64, pat \*Patient, doc \*Doctor, wgrp \*sync.WaitGroup) (\*Appointment, error)](http://localhost:8080/pkg/assignment4/clinic/#MakeAppointment)

[func (appt \*Appointment) CancelAppointment()](http://localhost:8080/pkg/assignment4/clinic/#Appointment.CancelAppointment)

[func (appt \*Appointment) EditAppointment(t int64, pat \*Patient, doc \*Doctor) error](http://localhost:8080/pkg/assignment4/clinic/#Appointment.EditAppointment)

[type BST](http://localhost:8080/pkg/assignment4/clinic/#BST)

[func (bst \*BST) GetDoctorByIDBST(docID int64) (\*Doctor, error)](http://localhost:8080/pkg/assignment4/clinic/#BST.GetDoctorByIDBST)

[type BinaryNode](http://localhost:8080/pkg/assignment4/clinic/#BinaryNode)

[type Doctor](http://localhost:8080/pkg/assignment4/clinic/#Doctor)

[func (d \*Doctor) GetAppointmentsByDate(dt int64) []\*Appointment](http://localhost:8080/pkg/assignment4/clinic/#Doctor.GetAppointmentsByDate)

[func (d \*Doctor) IsFreeAt(t int64) bool](http://localhost:8080/pkg/assignment4/clinic/#Doctor.IsFreeAt)

[type Patient](http://localhost:8080/pkg/assignment4/clinic/#Patient)

[func CreatePatient(username, first\_name, last\_name string, password []byte) (\*Patient, error)](http://localhost:8080/pkg/assignment4/clinic/#CreatePatient)

[func GetPatientByID(patientID string) (\*Patient, error)](http://localhost:8080/pkg/assignment4/clinic/#GetPatientByID)

[func IsLoggedIn(req \*http.Request) (\*Patient, bool)](http://localhost:8080/pkg/assignment4/clinic/#IsLoggedIn)

[func (p \*Patient) DeletePatient() error](http://localhost:8080/pkg/assignment4/clinic/#Patient.DeletePatient)

[func (p \*Patient) EditPatient(username, first\_name, last\_name string, password []byte)](http://localhost:8080/pkg/assignment4/clinic/#Patient.EditPatient)

[func (p \*Patient) GetAppointmentsByDate(dt int64) []\*Appointment](http://localhost:8080/pkg/assignment4/clinic/#Patient.GetAppointmentsByDate)

[func (p \*Patient) IsAdmin() bool](http://localhost:8080/pkg/assignment4/clinic/#Patient.IsAdmin)

[func (p \*Patient) IsFreeAt(t int64) bool](http://localhost:8080/pkg/assignment4/clinic/#Patient.IsFreeAt)

[type Payment](http://localhost:8080/pkg/assignment4/clinic/#Payment)

[func (pmy \*Payment) ClearPayment()](http://localhost:8080/pkg/assignment4/clinic/#Payment.ClearPayment)

[type PaymentNode](http://localhost:8080/pkg/assignment4/clinic/#PaymentNode)

[type PaymentQueue](http://localhost:8080/pkg/assignment4/clinic/#PaymentQueue)

[func CreatePayment(appt \*Appointment, amt float64, wg \*sync.WaitGroup) (\*PaymentQueue, error)](http://localhost:8080/pkg/assignment4/clinic/#CreatePayment)

[func (p \*PaymentQueue) Dequeue() (\*Payment, error)](http://localhost:8080/pkg/assignment4/clinic/#PaymentQueue.Dequeue)

[func (p \*PaymentQueue) DequeueToMissedPaymentQueue() (\*Payment, error)](http://localhost:8080/pkg/assignment4/clinic/#PaymentQueue.DequeueToMissedPaymentQueue)

[func (p \*PaymentQueue) DequeueToPaymentQueue() (\*Payment, error)](http://localhost:8080/pkg/assignment4/clinic/#PaymentQueue.DequeueToPaymentQueue)

[func (p \*PaymentQueue) Enqueue(pmy \*Payment) error](http://localhost:8080/pkg/assignment4/clinic/#PaymentQueue.Enqueue)

[func (p \*PaymentQueue) PrintAllQueueIDs(skipFirst bool) string](http://localhost:8080/pkg/assignment4/clinic/#PaymentQueue.PrintAllQueueIDs)

### **Package files**

[appointment.go](http://localhost:8080/src/assignment4/clinic/appointment.go) [config.go](http://localhost:8080/src/assignment4/clinic/config.go) [doc.go](http://localhost:8080/src/assignment4/clinic/doc.go) [doctor.go](http://localhost:8080/src/assignment4/clinic/doctor.go) [errors.go](http://localhost:8080/src/assignment4/clinic/errors.go) [helper.go](http://localhost:8080/src/assignment4/clinic/helper.go) [patient.go](http://localhost:8080/src/assignment4/clinic/patient.go) [payment.go](http://localhost:8080/src/assignment4/clinic/payment.go)

## Constants

Maximum number of days in the future allowed to make an appointment.

const MaxAdvanceApptDays = 90

Password policy.

const MinPasswordLength = 8

## Variables

Errors and accompanying messages to be output in logs or to users.

var (

// Appointments

ErrInvalidTimeslot = [errors](http://localhost:8080/pkg/errors/).[New](http://localhost:8080/pkg/errors/#New)("invalid timeslots entered")

ErrDoctorNoMoreTimeslot = [errors](http://localhost:8080/pkg/errors/).[New](http://localhost:8080/pkg/errors/#New)("doctor has no more timeslots available for today")

ErrPatientNoMoreTimeslot = [errors](http://localhost:8080/pkg/errors/).[New](http://localhost:8080/pkg/errors/#New)("patient has no more timeslots available for today")

ErrNoMoreTimeslot = [errors](http://localhost:8080/pkg/errors/).[New](http://localhost:8080/pkg/errors/#New)("there are no other timeslots available with the chosen doctor")

ErrDuplicateTimeslot = [errors](http://localhost:8080/pkg/errors/).[New](http://localhost:8080/pkg/errors/#New)("there's already an appointment scheduled for that timeslot")

ErrTimeslotExpired = [errors](http://localhost:8080/pkg/errors/).[New](http://localhost:8080/pkg/errors/#New)("timeslot has already expired")

ErrAppointmentIDNotFound = [errors](http://localhost:8080/pkg/errors/).[New](http://localhost:8080/pkg/errors/#New)("appointment id not found")

ErrCreateAppointment = [errors](http://localhost:8080/pkg/errors/).[New](http://localhost:8080/pkg/errors/#New)("unable to create appointment")

// Doc

ErrDoctorIDNotFound = [errors](http://localhost:8080/pkg/errors/).[New](http://localhost:8080/pkg/errors/#New)("doctor id not found")

ErrCreateDoctor = [errors](http://localhost:8080/pkg/errors/).[New](http://localhost:8080/pkg/errors/#New)("unable to create doctor")

// Patient

ErrPatientIDNotFound = [errors](http://localhost:8080/pkg/errors/).[New](http://localhost:8080/pkg/errors/#New)("patient id not found")

ErrCreatePatient = [errors](http://localhost:8080/pkg/errors/).[New](http://localhost:8080/pkg/errors/#New)("unable to create patient")

// Payment

ErrEmptyPaymentQueue = [errors](http://localhost:8080/pkg/errors/).[New](http://localhost:8080/pkg/errors/#New)("empty payment queue")

// DB

ErrDBConn = [errors](http://localhost:8080/pkg/errors/).[New](http://localhost:8080/pkg/errors/#New)("unable to get db connection")

)

Admins is a slice containing Ids of clinic staff.

var Admins = [][string](http://localhost:8080/pkg/builtin/#string){}

Appointments holds all the appointments sorted by Id.

var Appointments = []\*[Appointment](http://localhost:8080/pkg/assignment4/clinic/#Appointment){}

AppointmentsSortedByTimeslot holds all appointments sorted by time.

var AppointmentsSortedByTimeslot = []\*[Appointment](http://localhost:8080/pkg/assignment4/clinic/#Appointment){}

Doctors hold all the doctors sorted by incremental id.

var Doctors = []\*[Doctor](http://localhost:8080/pkg/assignment4/clinic/#Doctor){}

MissedPaymentQ holds the outstanding payments that have been moved over from PaymentQ.

var MissedPaymentQ = &[PaymentQueue](http://localhost:8080/pkg/assignment4/clinic/#PaymentQueue){}

Patients holds all the patients sorted by Id (alphanumeric).

var Patients = []\*[Patient](http://localhost:8080/pkg/assignment4/clinic/#Patient){}

PaymentQ holds the payments that are pending in a FIFO queue.

var PaymentQ = &[PaymentQueue](http://localhost:8080/pkg/assignment4/clinic/#PaymentQueue){}

var Wg [sync](http://localhost:8080/pkg/sync/).[WaitGroup](http://localhost:8080/pkg/sync/#WaitGroup)

## func [**BinarySearchApptID**](http://localhost:8080/src/assignment4/clinic/appointment.go?s=11802:11843#L413)

func BinarySearchApptID(apptID [int64](http://localhost:8080/pkg/builtin/#int64)) [int](http://localhost:8080/pkg/builtin/#int)

BinarySearchApptID performs binary search for appointment id in Appointments.

## func [**BinarySearchApptTime**](http://localhost:8080/src/assignment4/clinic/appointment.go?s=12526:12567#L436)

func BinarySearchApptTime(time [int64](http://localhost:8080/pkg/builtin/#int64)) [int](http://localhost:8080/pkg/builtin/#int)

BinarySearchApptTime performs binary search for appointment time in Appointments.

## func [**DbConnection**](http://localhost:8080/src/assignment4/clinic/config.go?s=1529:1555#L56)

func DbConnection() [string](http://localhost:8080/pkg/builtin/#string)

DbConnection returns the database connection string.

## func [**GetAvailableTimeslot**](http://localhost:8080/src/assignment4/clinic/appointment.go?s=6753:6827#L228)

func GetAvailableTimeslot(dt [int64](http://localhost:8080/pkg/builtin/#int64), apptsToExclude []\*[Appointment](http://localhost:8080/pkg/assignment4/clinic/#Appointment)) [][int64](http://localhost:8080/pkg/builtin/#int64)

GetAvailableTimeslot returns a slice of all the possible open timeslots for a given day by getting the delta between all timeslots for the day and a slice of appointments on the day.

## func [**IsApptTimeValid**](http://localhost:8080/src/assignment4/clinic/appointment.go?s=4841:4884#L169)

func IsApptTimeValid(t [int64](http://localhost:8080/pkg/builtin/#int64)) ([bool](http://localhost:8080/pkg/builtin/#bool), [error](http://localhost:8080/pkg/builtin/#error))

IsApptTimeValid checks if time of appointment is in the past - e.g. process started at 3:55 PM, user chose 4 PM timeslot but submitted form at 4:05 PM.

## func [**IsNRICValid**](http://localhost:8080/src/assignment4/clinic/patient.go?s=8909:8943#L329)

func IsNRICValid(nric [string](http://localhost:8080/pkg/builtin/#string)) [bool](http://localhost:8080/pkg/builtin/#bool)

IsNRICValid checks if a given NRIC is valid - Checks for length of 9 if strictNRIC is set to false (default) in clinic config; If true, will perform full NRIC validity check against checksum. Translated from <https://gist.github.com/kamerk22/ed5e0778b3723311d8dd074c792834ef>

## func [**IsThereTimeslot**](http://localhost:8080/src/assignment4/clinic/appointment.go?s=5500:5571#L188)

func IsThereTimeslot(dt [int64](http://localhost:8080/pkg/builtin/#int64), pat \*[Patient](http://localhost:8080/pkg/assignment4/clinic/#Patient), doc \*[Doctor](http://localhost:8080/pkg/assignment4/clinic/#Doctor)) ([bool](http://localhost:8080/pkg/builtin/#bool), [error](http://localhost:8080/pkg/builtin/#error))

IsThereTimeslot checks if there's timeslot available for the day by checking both the patient's and doctor's appointments for the day.

## func [**SeedData**](http://localhost:8080/src/assignment4/clinic/helper.go?s=299:314#L6)

func SeedData()

SeedData resets the database, setup the database, seed test data or load clinic globals from database depending on settings in clinic config.

## func [**SetDb**](http://localhost:8080/src/assignment4/clinic/config.go?s=1659:1683#L61)

func SetDb(myDb \*[sql](http://localhost:8080/pkg/database/sql/).[DB](http://localhost:8080/pkg/database/sql/#DB))

SetDb sets the singleton database connection to be used by package.

## type [**Appointment**](http://localhost:8080/src/assignment4/clinic/appointment.go?s=342:515#L9)

type Appointment struct {

Id  [int64](http://localhost:8080/pkg/builtin/#int64) // unique identifier

Time  [int64](http://localhost:8080/pkg/builtin/#int64) // unix time for easy sorting via int value comparison

Patient \*[Patient](http://localhost:8080/pkg/assignment4/clinic/#Patient)

Doctor \*[Doctor](http://localhost:8080/pkg/assignment4/clinic/#Doctor)

}

### **func** [**MakeAppointment**](http://localhost:8080/src/assignment4/clinic/appointment.go?s=1592:1692#L56)

func MakeAppointment(t [int64](http://localhost:8080/pkg/builtin/#int64), pat \*[Patient](http://localhost:8080/pkg/assignment4/clinic/#Patient), doc \*[Doctor](http://localhost:8080/pkg/assignment4/clinic/#Doctor), wgrp \*[sync](http://localhost:8080/pkg/sync/).[WaitGroup](http://localhost:8080/pkg/sync/#WaitGroup)) (\*[Appointment](http://localhost:8080/pkg/assignment4/clinic/#Appointment), [error](http://localhost:8080/pkg/builtin/#error))

Create Appointment, insert to database, add Appointment to global slice AppointmentsSortedByTimeslot and Appointments, sort global slice AppointmentsSortedByTimeslot, patient's Appointments slice and doctor's Appointments slice by appointment time.

### **func (\*Appointment)** [**CancelAppointment**](http://localhost:8080/src/assignment4/clinic/appointment.go?s=3818:3862#L132)

func (appt \*[Appointment](http://localhost:8080/pkg/assignment4/clinic/#Appointment)) CancelAppointment()

CancelAppointment removes appointment from AppointmentsSortedByTimeslot and Appointments, patient's Appointments slice, doctor's Appointments slice, delete corresponding database entry, sort AppointmentsSortedByTimeslot slice by time, Appointments slice by Id, patient's Appointments slice and doctor's Appointments slice by time.

### **func (\*Appointment)** [**EditAppointment**](http://localhost:8080/src/assignment4/clinic/appointment.go?s=2943:3025#L104)

func (appt \*[Appointment](http://localhost:8080/pkg/assignment4/clinic/#Appointment)) EditAppointment(t [int64](http://localhost:8080/pkg/builtin/#int64), pat \*[Patient](http://localhost:8080/pkg/assignment4/clinic/#Patient), doc \*[Doctor](http://localhost:8080/pkg/assignment4/clinic/#Doctor)) [error](http://localhost:8080/pkg/builtin/#error)

EditAppointment updates appointment item, updates corresponding database entry, sort AppointmentsSortedByTimeslot slice by time, Appointments slice by Id, patient's Appointments slice and doctor's Appointments slice by time.

## type [**BST**](http://localhost:8080/src/assignment4/clinic/doctor.go?s=3505:3544#L142)

type BST struct {

// contains filtered or unexported fields

}

DoctorsBST is a balanced binary search tree of doctors.

var DoctorsBST \*[BST](http://localhost:8080/pkg/assignment4/clinic/#BST)

### **func (\*BST)** [**GetDoctorByIDBST**](http://localhost:8080/src/assignment4/clinic/doctor.go?s=4752:4814#L197)

func (bst \*[BST](http://localhost:8080/pkg/assignment4/clinic/#BST)) GetDoctorByIDBST(docID [int64](http://localhost:8080/pkg/builtin/#int64)) (\*[Doctor](http://localhost:8080/pkg/assignment4/clinic/#Doctor), [error](http://localhost:8080/pkg/builtin/#error))

GetDoctorByIDBST gets a Doctor from the global DoctorsBST by Id. Returns pointer to Doctor if found.

## type [**BinaryNode**](http://localhost:8080/src/assignment4/clinic/doctor.go?s=3323:3501#L136)

type BinaryNode struct {

// contains filtered or unexported fields

}

## type [**Doctor**](http://localhost:8080/src/assignment4/clinic/doctor.go?s=269:387#L7)

type Doctor struct {

Id  [int64](http://localhost:8080/pkg/builtin/#int64)

First\_name  [string](http://localhost:8080/pkg/builtin/#string)

Last\_name  [string](http://localhost:8080/pkg/builtin/#string)

Appointments []\*[Appointment](http://localhost:8080/pkg/assignment4/clinic/#Appointment)

}

### **func (\*Doctor)** [**GetAppointmentsByDate**](http://localhost:8080/src/assignment4/clinic/doctor.go?s=2125:2188#L90)

func (d \*[Doctor](http://localhost:8080/pkg/assignment4/clinic/#Doctor)) GetAppointmentsByDate(dt [int64](http://localhost:8080/pkg/builtin/#int64)) []\*[Appointment](http://localhost:8080/pkg/assignment4/clinic/#Appointment)

GetAppointmentsByDate returns a slice of Appointments (pointers) on the given date (unix time). Todo: Can improve by making it binary search instead of sequential since Appointments is sorted by time.

### **func (\*Doctor)** [**IsFreeAt**](http://localhost:8080/src/assignment4/clinic/doctor.go?s=1771:1810#L79)

func (d \*[Doctor](http://localhost:8080/pkg/assignment4/clinic/#Doctor)) IsFreeAt(t [int64](http://localhost:8080/pkg/builtin/#int64)) [bool](http://localhost:8080/pkg/builtin/#bool)

## type [**Patient**](http://localhost:8080/src/assignment4/clinic/patient.go?s=347:489#L13)

type Patient struct {

Id  [string](http://localhost:8080/pkg/builtin/#string)

First\_name  [string](http://localhost:8080/pkg/builtin/#string)

Last\_name  [string](http://localhost:8080/pkg/builtin/#string)

Password [][byte](http://localhost:8080/pkg/builtin/#byte)

Appointments []\*[Appointment](http://localhost:8080/pkg/assignment4/clinic/#Appointment)

}

### **func** [**CreatePatient**](http://localhost:8080/src/assignment4/clinic/patient.go?s=1388:1481#L63)

func CreatePatient(username, first\_name, last\_name [string](http://localhost:8080/pkg/builtin/#string), password [][byte](http://localhost:8080/pkg/builtin/#byte)) (\*[Patient](http://localhost:8080/pkg/assignment4/clinic/#Patient), [error](http://localhost:8080/pkg/builtin/#error))

CreatePatient is for creating new Patient, inserting to database, add Patient to Patients slice and sort Patients slice via mergesort.

### **func** [**GetPatientByID**](http://localhost:8080/src/assignment4/clinic/patient.go?s=4395:4450#L170)

func GetPatientByID(patientID [string](http://localhost:8080/pkg/builtin/#string)) (\*[Patient](http://localhost:8080/pkg/assignment4/clinic/#Patient), [error](http://localhost:8080/pkg/builtin/#error))

GetPatientByID gets and return a Patient (pointer) by id.

### **func** [**IsLoggedIn**](http://localhost:8080/src/assignment4/clinic/patient.go?s=10543:10594#L390)

func IsLoggedIn(req \*[http](http://localhost:8080/pkg/net/http/).[Request](http://localhost:8080/pkg/net/http/#Request)) (\*[Patient](http://localhost:8080/pkg/assignment4/clinic/#Patient), [bool](http://localhost:8080/pkg/builtin/#bool))

IsLoggedIn checks if a user is logged in by checking for existence of client side Cookie and comparing Cookie's value to server side session data in MapSessions to check for validity; Returns Patient and true if valid.

### **func (\*Patient)** [**DeletePatient**](http://localhost:8080/src/assignment4/clinic/patient.go?s=3166:3205#L121)

func (p \*[Patient](http://localhost:8080/pkg/assignment4/clinic/#Patient)) DeletePatient() [error](http://localhost:8080/pkg/builtin/#error)

DeletePatient is for deleting Patient, Patient's appointments, removing patient from Patients slice and deleting corresponding database entry.

### **func (\*Patient)** [**EditPatient**](http://localhost:8080/src/assignment4/clinic/patient.go?s=2306:2392#L90)

func (p \*[Patient](http://localhost:8080/pkg/assignment4/clinic/#Patient)) EditPatient(username, first\_name, last\_name [string](http://localhost:8080/pkg/builtin/#string), password [][byte](http://localhost:8080/pkg/builtin/#byte))

EditPatient is for updating Patient, update corresponding database entry and sort Patients slice via mergesort.

### **func (\*Patient)** [**GetAppointmentsByDate**](http://localhost:8080/src/assignment4/clinic/patient.go?s=4729:4793#L182)

func (p \*[Patient](http://localhost:8080/pkg/assignment4/clinic/#Patient)) GetAppointmentsByDate(dt [int64](http://localhost:8080/pkg/builtin/#int64)) []\*[Appointment](http://localhost:8080/pkg/assignment4/clinic/#Appointment)

GetAppointmentsByDate gets a Patient's appointments (slice of pointers) on a given date (unix time).

### **func (\*Patient)** [**IsAdmin**](http://localhost:8080/src/assignment4/clinic/patient.go?s=8314:8346#L309)

func (p \*[Patient](http://localhost:8080/pkg/assignment4/clinic/#Patient)) IsAdmin() [bool](http://localhost:8080/pkg/builtin/#bool)

IsAdmin returns true if Patient is an admin. Checks recursively against Admins slice.

### **func (\*Patient)** [**IsFreeAt**](http://localhost:8080/src/assignment4/clinic/patient.go?s=2871:2911#L110)

func (p \*[Patient](http://localhost:8080/pkg/assignment4/clinic/#Patient)) IsFreeAt(t [int64](http://localhost:8080/pkg/builtin/#int64)) [bool](http://localhost:8080/pkg/builtin/#bool)

## type [**Payment**](http://localhost:8080/src/assignment4/clinic/payment.go?s=313:406#L7)

type Payment struct {

Id  [int64](http://localhost:8080/pkg/builtin/#int64)

Appointment \*[Appointment](http://localhost:8080/pkg/assignment4/clinic/#Appointment)

Amount  [float64](http://localhost:8080/pkg/builtin/#float64)

}

### **func (\*Payment)** [**ClearPayment**](http://localhost:8080/src/assignment4/clinic/payment.go?s=2173:2207#L91)

func (pmy \*[Payment](http://localhost:8080/pkg/assignment4/clinic/#Payment)) ClearPayment()

ClearPayment deletes the payment entry from database.

## type [**PaymentNode**](http://localhost:8080/src/assignment4/clinic/payment.go?s=410:480#L13)

type PaymentNode struct {

Payment \*[Payment](http://localhost:8080/pkg/assignment4/clinic/#Payment)

Next \*[PaymentNode](http://localhost:8080/pkg/assignment4/clinic/#PaymentNode)

}

## type [**PaymentQueue**](http://localhost:8080/src/assignment4/clinic/payment.go?s=484:567#L18)

type PaymentQueue struct {

Front \*[PaymentNode](http://localhost:8080/pkg/assignment4/clinic/#PaymentNode)

Back \*[PaymentNode](http://localhost:8080/pkg/assignment4/clinic/#PaymentNode)

Size  [int](http://localhost:8080/pkg/builtin/#int)

}

### **func** [**CreatePayment**](http://localhost:8080/src/assignment4/clinic/payment.go?s=1231:1324#L54)

func CreatePayment(appt \*[Appointment](http://localhost:8080/pkg/assignment4/clinic/#Appointment), amt [float64](http://localhost:8080/pkg/builtin/#float64), wg \*[sync](http://localhost:8080/pkg/sync/).[WaitGroup](http://localhost:8080/pkg/sync/#WaitGroup)) (\*[PaymentQueue](http://localhost:8080/pkg/assignment4/clinic/#PaymentQueue), [error](http://localhost:8080/pkg/builtin/#error))

CreatePayment is for creating payment, adding to database, adding to payment queue and removing the appointment.

### **func (\*PaymentQueue)** [**Dequeue**](http://localhost:8080/src/assignment4/clinic/payment.go?s=2725:2775#L120)

func (p \*[PaymentQueue](http://localhost:8080/pkg/assignment4/clinic/#PaymentQueue)) Dequeue() (\*[Payment](http://localhost:8080/pkg/assignment4/clinic/#Payment), [error](http://localhost:8080/pkg/builtin/#error))

Dequeue is for removing a payment from a queue.

### **func (\*PaymentQueue)** [**DequeueToMissedPaymentQueue**](http://localhost:8080/src/assignment4/clinic/payment.go?s=3908:3978#L177)

func (p \*[PaymentQueue](http://localhost:8080/pkg/assignment4/clinic/#PaymentQueue)) DequeueToMissedPaymentQueue() (\*[Payment](http://localhost:8080/pkg/assignment4/clinic/#Payment), [error](http://localhost:8080/pkg/builtin/#error))

DequeueToMissedPaymentQueue removes a payment from a queue and move it to MissedPaymentQ.

### **func (\*PaymentQueue)** [**DequeueToPaymentQueue**](http://localhost:8080/src/assignment4/clinic/payment.go?s=4231:4295#L193)

func (p \*[PaymentQueue](http://localhost:8080/pkg/assignment4/clinic/#PaymentQueue)) DequeueToPaymentQueue() (\*[Payment](http://localhost:8080/pkg/assignment4/clinic/#Payment), [error](http://localhost:8080/pkg/builtin/#error))

DequeueToPaymentQueue removes a payment from a queue and move it to PaymentQ.

### **func (\*PaymentQueue)** [**Enqueue**](http://localhost:8080/src/assignment4/clinic/payment.go?s=2415:2465#L100)

func (p \*[PaymentQueue](http://localhost:8080/pkg/assignment4/clinic/#PaymentQueue)) Enqueue(pmy \*[Payment](http://localhost:8080/pkg/assignment4/clinic/#Payment)) [error](http://localhost:8080/pkg/builtin/#error)

Enqueue is for adding a payment to a queue.

### **func (\*PaymentQueue)** [**PrintAllQueueIDs**](http://localhost:8080/src/assignment4/clinic/payment.go?s=3115:3177#L143)

func (p \*[PaymentQueue](http://localhost:8080/pkg/assignment4/clinic/#PaymentQueue)) PrintAllQueueIDs(skipFirst [bool](http://localhost:8080/pkg/builtin/#bool)) [string](http://localhost:8080/pkg/builtin/#string)

PrintAllQueueIDs returns a CSV of appointment ids from payments inside a payment queue.

# **Package psi**

import "assignment4/psi"

[Overview](http://localhost:8080/pkg/assignment4/psi/#pkg-overview)

[Index](http://localhost:8080/pkg/assignment4/psi/#pkg-index)

## Overview ▾

Package psi defines the PSI type and provide implementation for fetching the 24H average Pollutant Standards Index (PSI) value.

## Index ▾

[type PSI](http://localhost:8080/pkg/assignment4/psi/#PSI)

[func GetPSI() (\*PSI, error)](http://localhost:8080/pkg/assignment4/psi/#GetPSI)

### **Package files**

[psi.go](http://localhost:8080/src/assignment4/psi/psi.go)

## type [**PSI**](http://localhost:8080/src/assignment4/psi/psi.go?s=238:300#L3)

type PSI struct {

Value  [string](http://localhost:8080/pkg/builtin/#string)

Description [string](http://localhost:8080/pkg/builtin/#string)

}

### **func** [**GetPSI**](http://localhost:8080/src/assignment4/psi/psi.go?s=419:446#L9)

func GetPSI() (\*[PSI](http://localhost:8080/pkg/assignment4/psi/#PSI), [error](http://localhost:8080/pkg/builtin/#error))

GetPSI returns a PSI item containing the 24H national average pollutant standards index value and description.

# **Package session**

import "assignment4/session"

[Overview](http://localhost:8080/pkg/assignment4/session/#pkg-overview)

[Index](http://localhost:8080/pkg/assignment4/session/#pkg-index)

## Overview ▾

Package session defines the Session and Notification types; It provides implementation for creating and deleting of server side session, client side cookie and use of notification inside session to pass messages between http request.

## Index ▾

[Variables](http://localhost:8080/pkg/assignment4/session/#pkg-variables)

[func ClearNotification(req \*http.Request) error](http://localhost:8080/pkg/assignment4/session/#ClearNotification)

[func CreateSession(res http.ResponseWriter, req \*http.Request, username, serverHost string)](http://localhost:8080/pkg/assignment4/session/#CreateSession)

[func DeleteSession(res http.ResponseWriter, req \*http.Request)](http://localhost:8080/pkg/assignment4/session/#DeleteSession)

[func SetNotification(req \*http.Request, notificationMsg, notificationType string) error](http://localhost:8080/pkg/assignment4/session/#SetNotification)

[type Notification](http://localhost:8080/pkg/assignment4/session/#Notification)

[func GetNotification(req \*http.Request) (\*Notification, error)](http://localhost:8080/pkg/assignment4/session/#GetNotification)

[type Session](http://localhost:8080/pkg/assignment4/session/#Session)

### **Package files**

[doc.go](http://localhost:8080/src/assignment4/session/doc.go) [notification.go](http://localhost:8080/src/assignment4/session/notification.go) [session.go](http://localhost:8080/src/assignment4/session/session.go)

## Variables

CookieID is the name of the client side cookie.

var CookieID [string](http://localhost:8080/pkg/builtin/#string)

MapSessions stores all the user session(s) of the app.

var MapSessions = [make](http://localhost:8080/pkg/builtin/#make)(map[[string](http://localhost:8080/pkg/builtin/#string)][Session](http://localhost:8080/pkg/assignment4/session/#Session))

## func [**ClearNotification**](http://localhost:8080/src/assignment4/session/notification.go?s=1049:1096#L32)

func ClearNotification(req \*[http](http://localhost:8080/pkg/net/http/).[Request](http://localhost:8080/pkg/net/http/#Request)) [error](http://localhost:8080/pkg/builtin/#error)

ClearNotification deletes a notification message from user's session.

## func [**CreateSession**](http://localhost:8080/src/assignment4/session/session.go?s=1132:1223#L41)

func CreateSession(res [http](http://localhost:8080/pkg/net/http/).[ResponseWriter](http://localhost:8080/pkg/net/http/#ResponseWriter), req \*[http](http://localhost:8080/pkg/net/http/).[Request](http://localhost:8080/pkg/net/http/#Request), username, serverHost [string](http://localhost:8080/pkg/builtin/#string))

CreateSession creates client side cookie, create and add session to global MapSessions; Also, purge any duplicate sessions by a user.

## func [**DeleteSession**](http://localhost:8080/src/assignment4/session/session.go?s=1715:1777#L61)

func DeleteSession(res [http](http://localhost:8080/pkg/net/http/).[ResponseWriter](http://localhost:8080/pkg/net/http/#ResponseWriter), req \*[http](http://localhost:8080/pkg/net/http/).[Request](http://localhost:8080/pkg/net/http/#Request))

DeleteSession expires a user's client side cookie and remove session from global MapSessions.

## func [**SetNotification**](http://localhost:8080/src/assignment4/session/notification.go?s=221:308#L4)

func SetNotification(req \*[http](http://localhost:8080/pkg/net/http/).[Request](http://localhost:8080/pkg/net/http/#Request), notificationMsg, notificationType [string](http://localhost:8080/pkg/builtin/#string)) [error](http://localhost:8080/pkg/builtin/#error)

SetNotification sets a notification message to user's session.

## type [**Notification**](http://localhost:8080/src/assignment4/session/notification.go?s=58:150#L1)

type Notification struct {

Message [string](http://localhost:8080/pkg/builtin/#string)

Type  [string](http://localhost:8080/pkg/builtin/#string) // Types: "Success", "Error"

}

### **func** [**GetNotification**](http://localhost:8080/src/assignment4/session/notification.go?s=730:792#L22)

func GetNotification(req \*[http](http://localhost:8080/pkg/net/http/).[Request](http://localhost:8080/pkg/net/http/#Request)) (\*[Notification](http://localhost:8080/pkg/assignment4/session/#Notification), [error](http://localhost:8080/pkg/builtin/#error))

GetNotification gets a notification message from user's session.

## type [**Session**](http://localhost:8080/src/assignment4/session/session.go?s=127:247#L4)

type Session struct {

Id  [string](http://localhost:8080/pkg/builtin/#string)

LastModified [int64](http://localhost:8080/pkg/builtin/#int64)

LastVisited \*[url](http://localhost:8080/pkg/net/url/).[URL](http://localhost:8080/pkg/net/url/#URL)

Notification \*[Notification](http://localhost:8080/pkg/assignment4/session/#Notification)

}

# **Package web**

import "assignment4/web"

[Overview](http://localhost:8080/pkg/assignment4/web/#pkg-overview)

[Index](http://localhost:8080/pkg/assignment4/web/#pkg-index)

## Overview ▾

Package web defines route constants and provides the implementation for http handlers and server.

## Index ▾

[func StartHttpServer(myDb \*sql.DB)](http://localhost:8080/pkg/assignment4/web/#StartHttpServer)

### **Package files**

[adminPages.go](http://localhost:8080/src/assignment4/web/adminPages.go) [appointment.go](http://localhost:8080/src/assignment4/web/appointment.go) [config.go](http://localhost:8080/src/assignment4/web/config.go) [doc.go](http://localhost:8080/src/assignment4/web/doc.go) [doctor.go](http://localhost:8080/src/assignment4/web/doctor.go) [errorPages.go](http://localhost:8080/src/assignment4/web/errorPages.go) [errors.go](http://localhost:8080/src/assignment4/web/errors.go) [helper.go](http://localhost:8080/src/assignment4/web/helper.go) [httpServer.go](http://localhost:8080/src/assignment4/web/httpServer.go) [index.go](http://localhost:8080/src/assignment4/web/index.go) [patient.go](http://localhost:8080/src/assignment4/web/patient.go) [payment.go](http://localhost:8080/src/assignment4/web/payment.go) [psi.go](http://localhost:8080/src/assignment4/web/psi.go) [routes.go](http://localhost:8080/src/assignment4/web/routes.go)

## func [**StartHttpServer**](http://localhost:8080/src/assignment4/web/httpServer.go?s=861:895#L21)

func StartHttpServer(myDb \*[sql](http://localhost:8080/pkg/database/sql/).[DB](http://localhost:8080/pkg/database/sql/#DB))

StartHttpServer setup routes & handlers and start web server over https. It also calls SeedData from clinic package to perform database setup and/or seeding of test data depending on clinic package's config settings.