# Go Track
# Week 5 Practical

# Networking in Go

## Use of “net” package

| Description | In this lesson, you will learn about the linked list data structure in Go. |
|---|---|
| Learning Objectives | Understand the use of “net” package<br>Understand the use of simple TCP communications |
| Duration | 30 ~ 45 minutes |

## Activity #1: Explore your System Details

Make use of the package “net” and derive the different information from your desktop/ laptop.
Experiment around with the different properties to see what information you can derive from the Go package.

## Activity #2: Simple TCP Server in Action

Create a simple TCP Server to

1. Listen at port 5331.
2. Handle multiple clients.
3. Receive text messages from the client and sends an echo message back to the client.
4. Create a simple TCP Client to
5. Dial up to the TCP server at port 5331.
6. Request for user to key in message.
7. Send message to TCP server at port 5331
8. Read for response from the TCP server after sending the message and prints out the message.

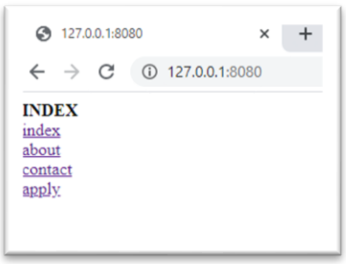
## Activity #3: Simple TCP Server with HTTP addon

Create a TCP server that can
1. Listen to port 8080.
2. Check the first line of request and print out
3. The method request used
4. The URI of the request
5. The type of the request

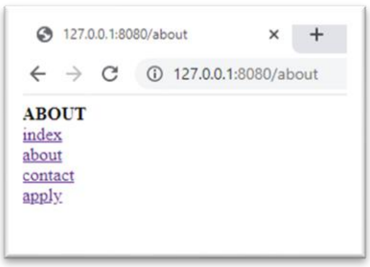
Create a simple HTML webpage that would respond to the click request of the user

A sample webpage mapping to consider
1. Main index at "/"
2. About page at "/about"
3. Contact page at "/contact"
4. Apply page at "/apply"
5. A "posted" apply page at "/apply"

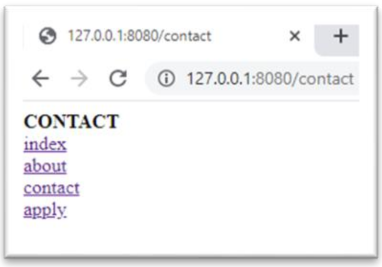
The server should respond with the proper header to the browser
- "HTTP/1.1 200 OK\r\n"
- "Content-Length: <length of message>\r\n"
- "Content-Type: text/html\r\n"


127.0.0.1:8080

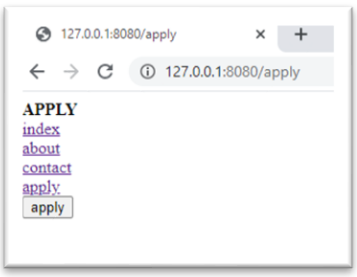
**INDEX**
index
about
contact
apply


127.0.0.1:8080/about

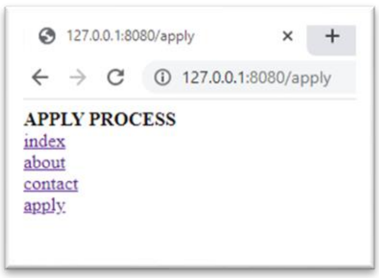
**ABOUT**
index
about
contact
apply


127.0.0.1:8080/contact

**CONTACT**
index
about
contact
apply


127.0.0.1:8080/apply

**APPLY**
index
about
contact
apply
apply


127.0.0.1:8080/apply

**APPLY PROCESS**
index
about
contact
apply

You may wish to use the following simple HTML

```
   body := `<!DOCTYPE html><html lang="en"><head><meta charet="UTF8">
<title></title></head><body>
   <strong>INDEX</strong><br>
   <a href="/">index</a><br>
   <a href="/about">about</a><br>
   <a href="/contact">contact</a><br>
   <a href="/apply">apply</a><br>
   </body></html>`
```

# Networking in Go

## Use of “net\http” package

| Description | *In this lesson, you will learn about the linked list data structure in Go.* |
|---|---|
| Learning Objectives | Understand the use of “net\http” package<br>Understand the use of simple HTTP communications |
| Duration | 30 ~ 45  minutes |

## Activity #1: Simple Server using ServeHTTP

Use the package “net/http” to create a server based on ServeHTTP function.

Create a server that

1. Listen to port 8080
2. Serves a simple message response
3. Header “GoTrack” with data “This is from Go Track”
4. Prints a message – “Your code here will run”

Modify the existing server further to serve multiple calls to ServeHTTP.

Modify the existing server further to serve multiple calls using URL routing.

## Activity #2: Simple Server using ServeMux

Use the package “net/http” to create a server based on the ServeMux function.

Create a server that

1. Listen to port 8080
2. Loads information into the ServeMux object.
    a. “/”
    b. “/advanced”
    c. “/basic”

Launches a new page indicating which is selected by changing the url

- 127.0.0.1:8080
- 127.0.0.1:8080/advanced
- 127.0.0.1:8080/basic

## Activity #3: Simple HTTP Server

Use the package “net/http” to create a server based on the Handle function.

Create a server that
1. Listen to port 8080
2. Maps functions to the pattern string

You may want to consider the following
1. 127.0.0.1:8080/feature1 -- > “Feature 1 is activated” text
2. 127.0.0.1:8080//feature2 -- > “Feature 2 is activated” text

Modify the existing server to use the HandleFunc function.

Modify the server to handle request for “Javicon.ico” if you are using Chrome to response with “Http not found”

## Activity #4: Simple HTTP FileServer

Create a simple HTML index

Apply CSS and assets provided to create a simple index.html load screen.

Modify the File server to serve different content based on different request.

You may wish to consider using the following
1. A link response to an existing pic in server.
2. A link response to an online URL
   a. https://upload.wikimedia.org/wikipedia/commons/2/22/Kyoto_montage.jpg

## Activity #5: Simple HTTP Client with Timeout

Create a HTTP Client that
1. Queries the server at 127.0.0.1:8080
2. Has the property of timeout of 15 seconds

Run the HTTP Client without a server. Observe the behaviour of the client.

Run any server at 127.0.0.1:8080 and run the HTTP Client. Observe the behaviour of the client.

## Activity #6: Simple HTTP Client with Concurrency

Create a HTTP Server that serves the current browse count to the client browser.

Recall the previous concurrency count using Mutex.

Launch 2 browsers as client and request for an updated count from the server.

NGEE ANN POLYTECHNIC

# Networking in Go

## Templates

| Description | *In this lesson, you will learn about the linked list data structure in Go.* |
|---|---|
| Learning Objectives | **Understand the use of “templates” package**<br>**Understand the use of simple template usage** |
| Duration | **30 ~ 45 minutes** |

## Activity #1: Template basic read output

Create a template to read basic output from the server.
Create a simple name variable to be assigned on print.

```
<!DOCTYPE html>
<html lang = "en'>
<head>
<meta charset = "UTF-8">
<title>Hello! </title>
</head>
<body>
<h1>` + name + `</h1>
</body>
```

## Activity #2: Create HTML from templates

Modify the template go file to create a html file on runtime.

## Activity #3: Further read output for templates.

Create 3 simple files of any filename and put messages into them.

For example, a file of message1.what would have a content of “This is from message1.what”.

Create a go file that would parse the information and print in accordance.

Modify the existing files to use global read from a directory of files with extension .gohtml or of your own preference.

## Activity #4: Passing Data

Create a simple template to receive strings from a go file.

Send a message “Go Track. This is the message;The message will go in as a variable” on runtime.

## Activity #5: Passing Data Slices

Create a simple template to receive slices from a go file.

Send a message which consist of
1. Go Data 1
2. Go Data 2
3. Go Data 3
4. Go Data 4

Launch the browser at runtime and display the data slice.

Modify the display to include the index using variables $index and $element.

## Activity #5: Passing Data Maps

Create a simple template to receive map from a go file.

Send a message which consist of the following key and value
- “1”: “Go Data 1”
- “2”: “Go Data 2”
- “3”: “Go Data 3”
- “4”: “Go Data 4”

Launch the browser at runtime and display the data map.

Modify the display to include the index using variables $Key and $Val.

## Activity #6: Passing Data Struct

Create a simple template to receive Struct from a go file.

Send a message which consist of the following Struct *student*
- FirstName – string
- LastName – string

You may wish to use the following info
- FirstName : “Obiwan”
- LastName : “Kenobi”

Launch the browser at runtime and display the data struct.

Modify the existing struct to include another Struct *course* that would contain the following
- CourseName – string
- MaxCount – int

You may wish to use the following info
Student1

- FirstName : “Anakin”
- LastName :”Skywalker”

Student2

- FirstName: “R2”
- LastName : “D2”

Course1

- CourseName: “Starfighter Navigation”
- MaxCount : 50

Course2

- CourseName: “Deathstar Assault”
- MaxCount : 100

Launch the browser at runtime and display the data struct.

## Activity #7: Functions in Templates

Create a template that allows functions to be in.

Create 3 functions that would do the following

1. Return the name of the student
2. Return the class of the student
3. Do a summation of the scores and return the sum

You may wish to use the following

student1

- Name:  "Obiwan"
- Class: "Tatooine"

student2

- Name:  "Lord Vader"
- Class: "Death Star"

score1

- Module1: 10
- Module2: 20

score2

- Module1: 30
- Module2: 40

## Activity #7: Pipeline and Global Functions in Templates

Create a template that allows functions to be in.

Create 3 functions that would do the following
1. Return a double of a value
2. Return a square of a value
3. Return a square root of a value

Create a pipeline in the template that does
- Display the value
- Display the value, display double of the value
- Display the value, display double of the value, display square of the value
- Display the value, display double of the value, display square of the value, display square root of the value.

Create a template that would do comparison using the global function of Boolean

Send 2 values into the template to do
- Compare if value 1 is greater than value 2
- Compare if value 1 is smaller than value 2
- Compare if value 1 is same as value 2
- Compare if value 1 is smaller than 10

## Activity #8: Nested Template

Using the existing template from global functions,

Create 3 templates
1. Header template
    a. Contains the header for HTML
2. Compare template
    a. Contains the comparison
3. Main template
    a. Contains the main template layout

# Networking in Go

## States and Sessions

| **Description** | ***In this lesson, you will learn about the linked list data structure in Go.*** |
|---|---|
| **Learning Objectives** | **Understand the use of URL, Form and Cookies**<br>**Understand the use of Sessions**<br>**Understand the use of Redirects, 3xx series**<br>**Understand the use of Cookies** |
| **Duration** | **30 ~ 45 minutes** |

## Activity #1: URL State

Make use of the URL identifier and value to send information to the templates.

Create a simple server to response to a query and print out the query.
You may wish to use the keyword "query"

Launch a browser and use the URL to query a value.

## Activity #2: Forms State

Make use of the Form to POST information to the templates.

Create a simple server to response with a form

You may wish to use the following simple form

```
io.WriteString(w, `
<form method="POST">
 <input type="text" name="text">
 <input type="submit">
</form>`)
```

Launch a browser and submit a value. Observe the outcome

Change the method to "GET". Launch a browser and submit a value. Observe the outcome.

Modify to include multiple entries to be submitted to the form.

- First name
- Last name
- Grade

## Activity #3: Uploading, reading and storing simple text file to a Server

Make use of Forms

1. Upload a simple text file of your choice into the Server.
2. Have the Server read and send the content of the text file to the web browser.
3. Create and keep a copy of the text file to the Server.

Create a simple server to load a simple form for submitting files.

In addition to the simple form, an *enctype=”multipart/form-data”* property is added.

The server needs to

1. Identify the method that is triggered from.
2. Read the parameters of the file using req.FormFile
3. Read the file from the form using ioutil.ReadAll
4. Prepare a new file to be written using os.Create
5. Write to an output file using writer.Write

You may wish to use the following simple form

```
io.WriteString(res, `
<form method="POST" enctype="multipart/form-data">
<input type="file" name="filename">
<input type="submit">
</form>`)
```

## Activity #4: Redirects for 3xx series

Create as simple server with 2-3 functions to observe the redirect sequences.

Have 1 function as the primary direct access function.

- You may wish to use “127.0.0.1:8080/request”

Have 2 functions to demonstrate each of the 3xx series reactions.

- You may wish to use “127.0.0.1:8080/redirect1” or “127.0.0.1:8080/redirect2”

Each function should print out its own function name and current http method it receives.

Explore and observe the behaviour for 3xx series –

1. 303 – See Other
   – Server sent this response to direct the client to another URI with GET request.
2. 301/ 308 – Moved Permanently / Permanent Redirect
   – URL of the request has been changed permanently. New URL is given.

## Activity #5: Cookies

Create a simple HTTP Server.

1. Request for a simple page using the web browser.
2. Write a cookie object with the following details or your own preferred values.
3. Name: Go Track Cookie
4. Value: GT001
5. Set the cookie to the user browser.
6. Open the developer view by using Ctrl – Shift – I in Chrome.
7. Navigate under dev tools / application / cookies to observe the cookie and its corresponding values.
8. Read the cookie value and display in the web browser.
9. Delete the cookie.

You may wish to create the following function mappings

- Set – to set the cookie when requested
- Read – to read the cookie content when requested
- Delete – to delete the cookie content when requested

You may also explore the use of the following HTML to allow ease of transition between pages.

- <a href=”/string”> Label </a>

## Activity #6: Sessions

Install package for UUID using the following command

*go get github.com/satori/go.uuid*

Create a simple HTTP Server.

- Create a session with a cookie.
- Retrieve and print the session code using the cookie.

## Activity #7: Sessions control

Install package for UUID using the following command

*go get github.com/satori/go.uuid*

Create a simple HTTP Server.

- Create a session with a cookie.
- If the cookie does not exist, create a new cookie
- Display the main index page.
- The page will allow user to enter
- Username
- First name
- Last name
- The page will have a “submit” button
- There is a Restricted Area.
- If the user does not exist, the restricted area is not displayed.
- A user needs to sign up before he has access to the restricted area.
- Current sign up will be shown upon clicking submit.
- The restricted area will display user information with the cookie value.

## Activity #8: Login and Logout

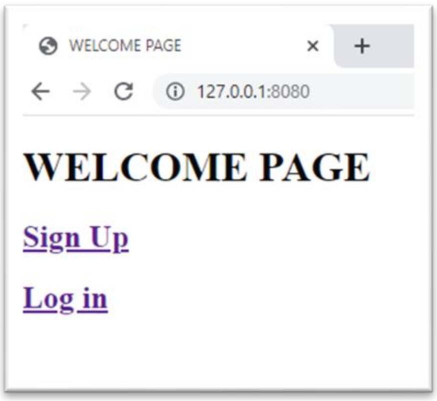
Install package for UUID using the following command

*go get golang.org/x/crypto/bcrypt*

Create a simple HTTP Server.

- Create a session with a cookie.
- Create a signup page for user.
- "127.0.0.1:8080/signup"
- Create a login page for user.
- "127.0.0.1:8080/login"
- Create a landing page for user.
- "127.0.0.1:8080"
- Allow the correct user to login in after signing up.
- Allow the user to logout.

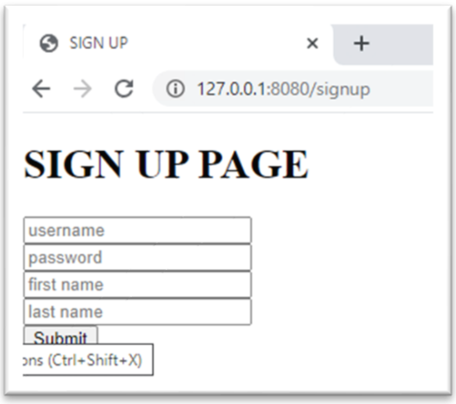
You may wish to use the following pages as a reference



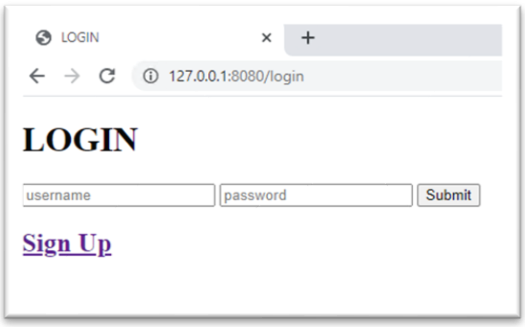
WELCOME PAGE

Sign Up

Log in

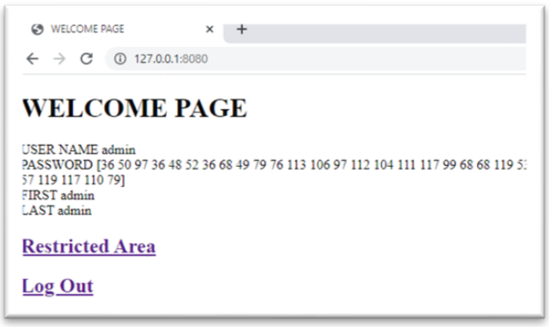


SIGN UP PAGE



LOGIN

Sign Up

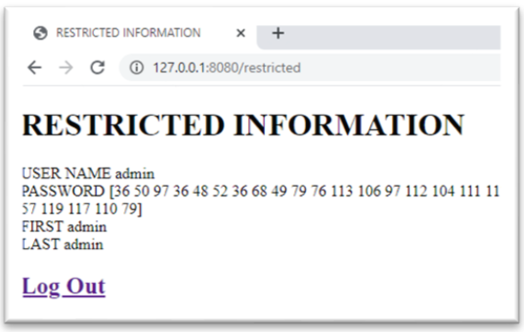


WELCOME PAGE

USER NAME admin
PASSWORD [36 50 97 36 48 52 36 68 49 79 76 113 106 97 112 104 111 117 99 68 68 119 5
57 119 117 110 79]
FIRST admin
LAST admin

Restricted Area

Log Out



RESTRICTED INFORMATION

USER NAME admin
PASSWORD [36 50 97 36 48 52 36 68 49 79 76 113 106 97 112 104 111 11
57 119 117 110 79]
FIRST admin
LAST admin

Log Out

# Networking in Go

## JSON & XML

| | |
|---|---|
| **Description** | ***In this lesson, you will learn about the linked list data structure in Go.*** |
| **Learning Objectives** | **Understand the use of Encode and Decode**<br>**Understand the use of Marshal and Unmarshal** |
| **Duration** | **30 ~ 45 minutes** |

## Activity #1: JSON Data Display

Create a Server that would

- Request for the latest weather data from the following URL
  a. https://api.data.gov.sg/v1/environment/24-hour-weather-forecast
- Serve a page for the highest and lowest temperature that is in the forecast
- Serve a page for the weather situation in the forecast

## Activity #2: XML Data Configuration

Create a XML that would configure the following server properties

- IP Address: 127.0.0.1
- Port Number: 5003

The XML will take on the following form of tags

- <address>127.0.0.1</address>
- <port>5003</port>

Make further changes to the configuration file and observe the relevant changes.