Using Sample App

Application Architecture, Page Layout, and Data Storage

Learning Outcomes

After completing this lab you should be able to

- Experiment on **architecting** Go applications
- Design **page layouts** using Go Templating
- Store application data **in-memory**
- Store application data on using Go's `csv` and `gob` packages

Sample Web Application for the Course

Simple Restaurant Application (Functional Requirement)

As a **customer** I want to see the **menu** so that I can order the food I want

As a **customer** I want to order a particular food from the menu

As a **customer** I want to contact the restaurant manager

Sample Web Application for the Course

Acceptance Criteria

Functional

As a **customer** can I save my order and come back to it later?

As a **customer** can I change my order before it is delivered?

As a **customer** can I see a running total of the cost of what I have chosen so far?

Sample Web Application for the Course

Simple Restaurant Application (Functional Requirement)

As a **manager** I want to maintain food menu

As a **manager** I want to read customer suggestions and complaints

Sample Web Application for the Course

Acceptance Criteria

Functional

As a **manager** can I maintain food menu?

As a **manager** can I read customer complaints?

Sample Web Application for the Course

Acceptance Criteria

Non Functional (Security)

Are unauthorised persons and other customers prevented from viewing customer orders?

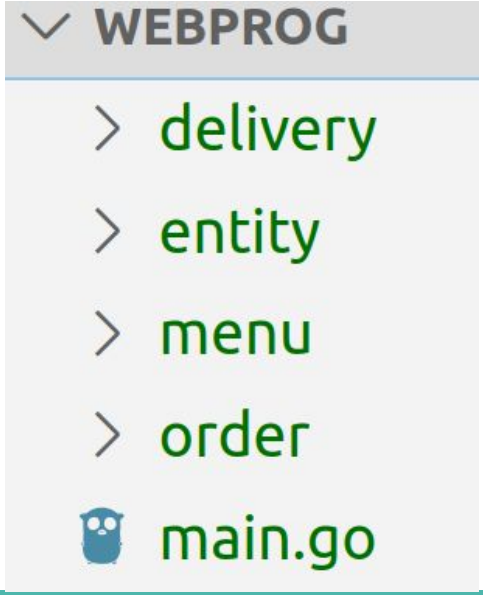
Are unauthorised persons and other customers prevented from viewing to customer complaints?

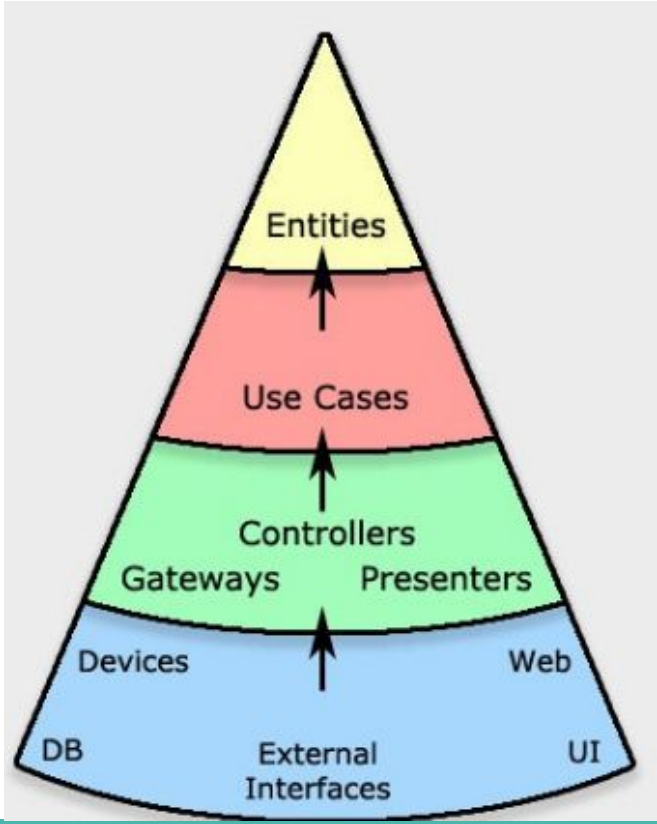
Are unauthorised persons and other customers prevented from maintaining food menu?

Application Directory/Package Structure

Following the clean architecture principle

Create the following directory structure



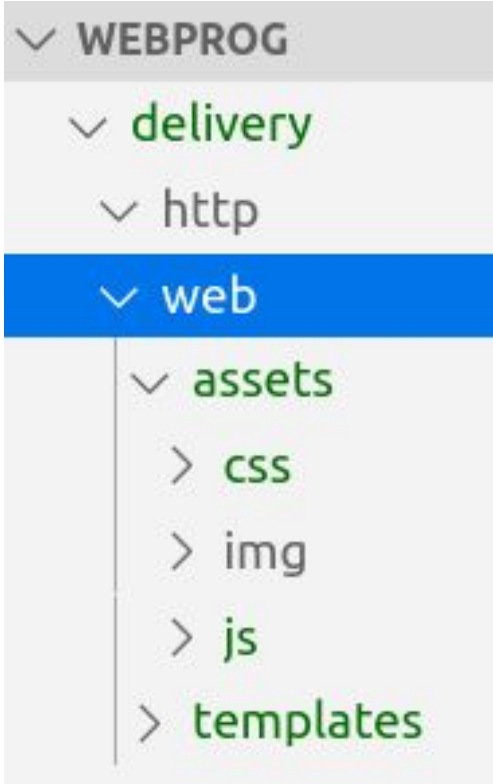


Application Directory/Package Structure

Create a directory named **`web`** inside the **`delivery`** directory

Inside the **`web`** directory create two directories - **`assets`** and **`templates`**

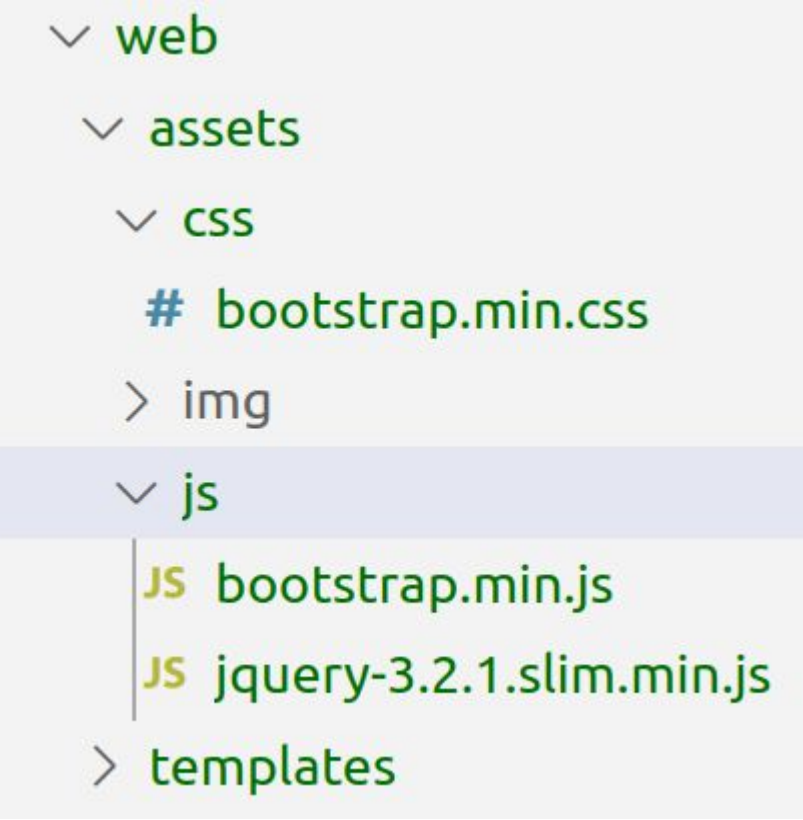
Inside **`assets`** directory create three directories - **`css`**, **`img`**, and **`js`**



Template Layout

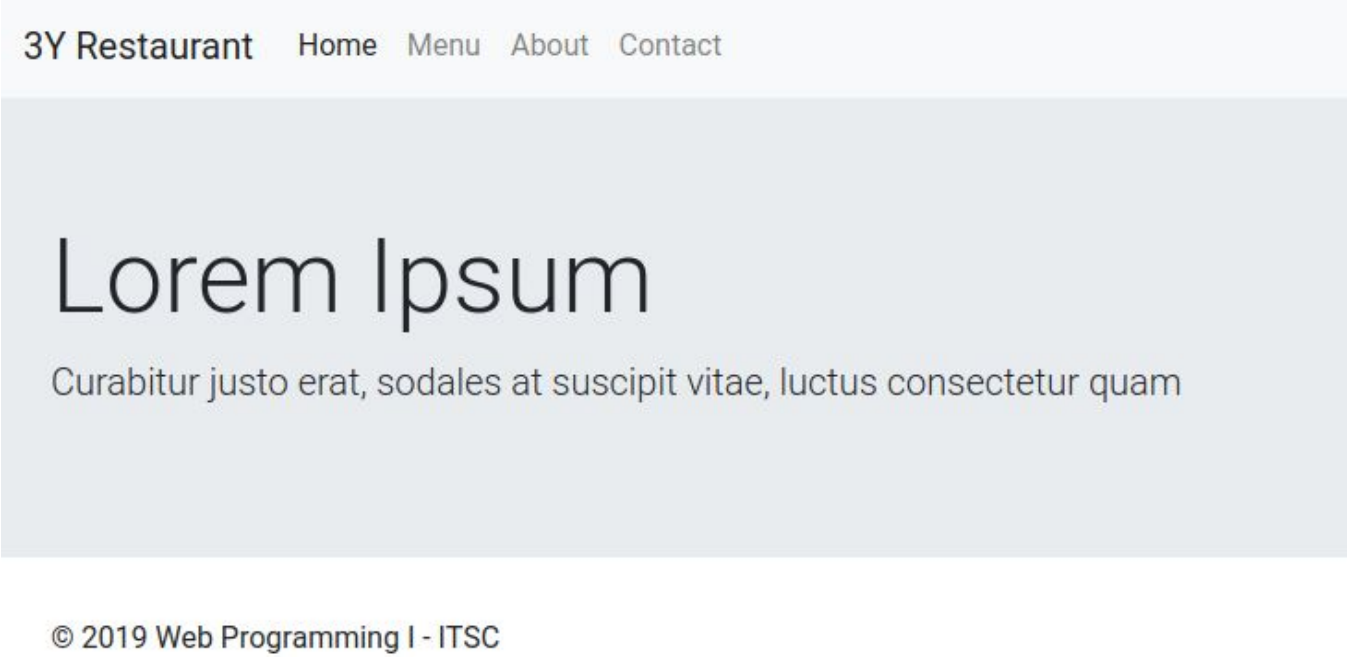
We will use **Bootstrap** framework for the front end

Place the Bootstrap `css` and `js` files inside css and js directory as shown in the right hand side



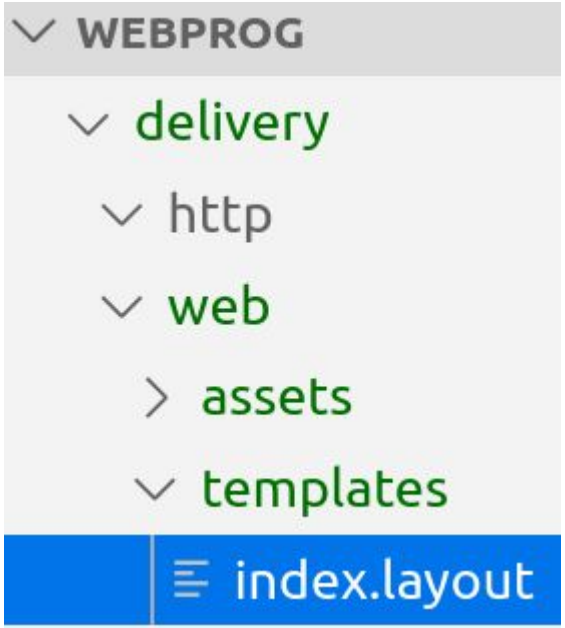
Template Layout

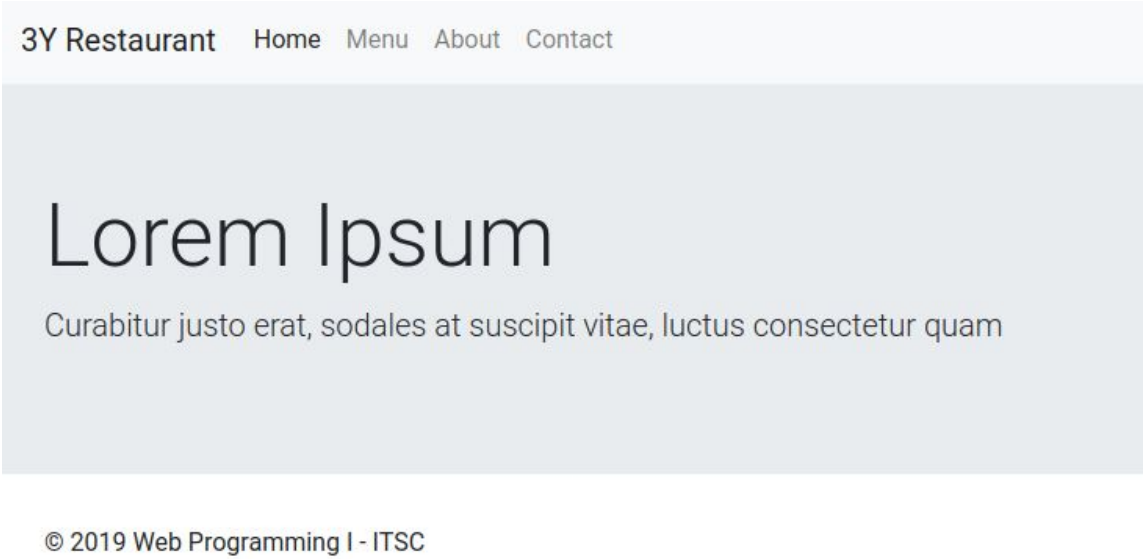
Let us see how to create the following page layout



Template Layout

Create `index.layout` file inside `templates` directory





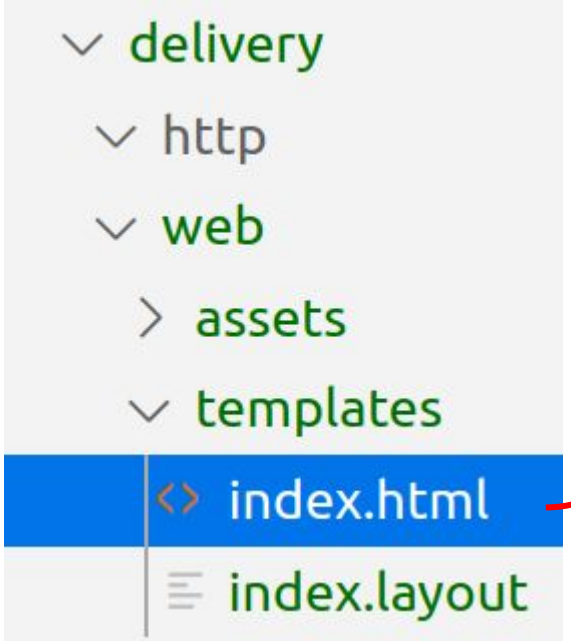
Template Layout

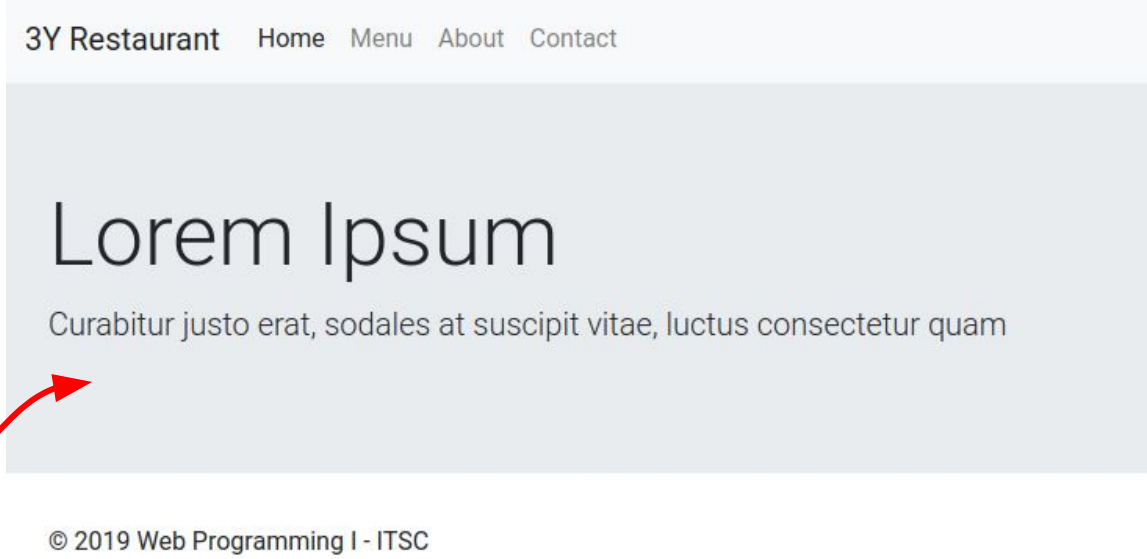
Put the following content inside `index.layout` file

```
{{ define "index.layout" }}
 {{ template "navbar" }}
 {{ template "index.content" . }}
 {{ template "footer" }}
{{ end }}
```

Template Layout

Create `index.html` file inside `templates` directory





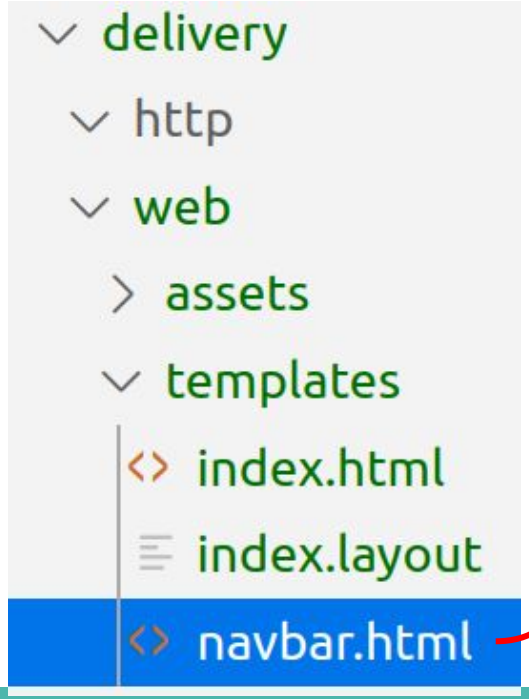
Template Layout

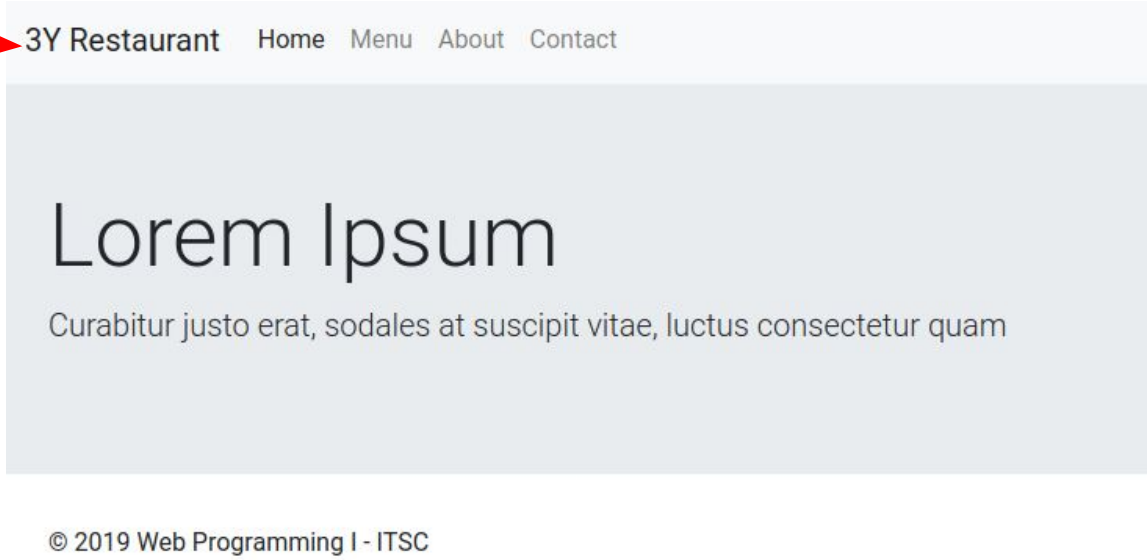
Put the following content inside `index.html` file

```
{{ define "index.content" }}
<div class="jumbotron jumbotron-fluid">
    <div class="container">
        <h1 class="display-4">Lorem Ipsum</h1>
        <p class="lead">Curabitur justo eratr</p>
    </div>
</div>
{{ end }}
```

Template Layout

Create `navbar.html` file inside `templates` directory





Template Layout

Put the following content inside **`navbar.html`** file

```
{{ define "navbar" }}

<!DOCTYPE html>
<html>

<head>
    <title>{{ .Title }}</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <!-- css -->
    <link rel="stylesheet" href="/assets/css/bootstrap.min.css">
</head>
```

Template Layout

Put the following content inside **`navbar.html`** file

```
<body>
    <nav class="navbar navbar-expand-lg navbar-light bg-light">
        <a class="navbar-brand" href="/">3Y Restaurant</a>
        <button class="navbar-toggler" type="button" data-toggle="
            aria-controls="navbarSupportedContent" aria-expanded="
            <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse" id="navbarSupported
            <ul class="navbar-nav mr-auto">…
            </ul>
        </div>
    </nav>
{{ end }}
```

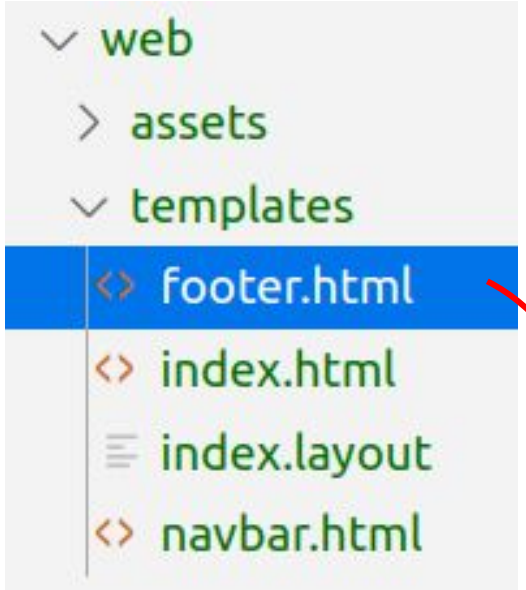
The collapsed content is found on next slide

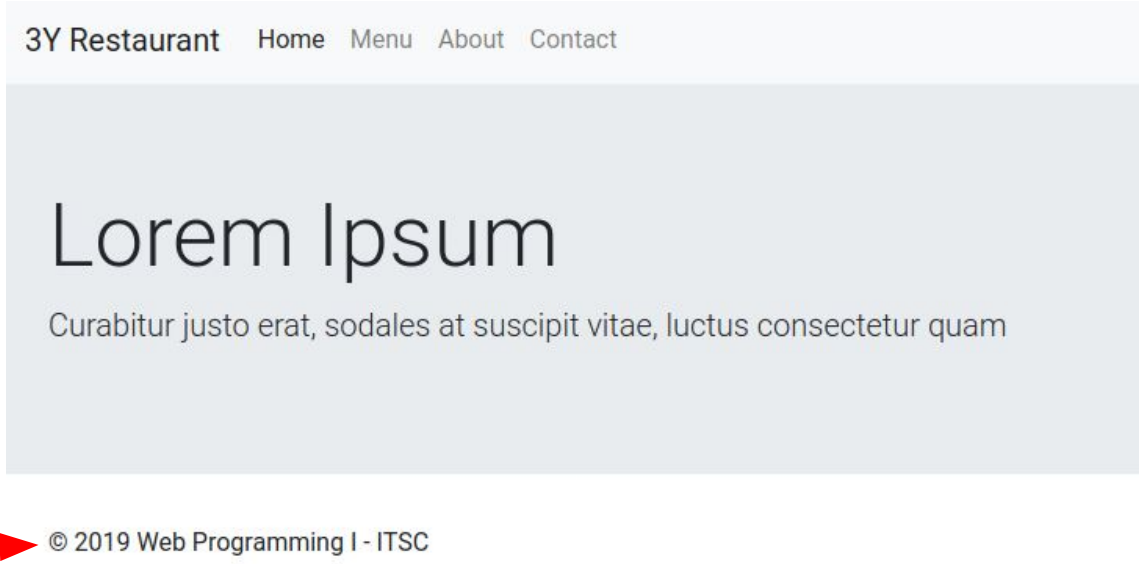
Template Layout

```
<ul class="navbar-nav mr-auto">
    <li class="nav-item active">
        <a class="nav-link" href="/">Home <span class="sr-only'
    </li>
    <li class="nav-item">
        <a class="nav-link" href="menu">Menu</a>
    </li>
    <li class="nav-item">
        <a class="nav-link" href="about">About</a>
    </li>
    <li class="nav-item">
        <a class="nav-link" href="contact">Contact</a>
    </li>
</ul>
```

Template Layout

Create `footer.html` file inside `templates` directory





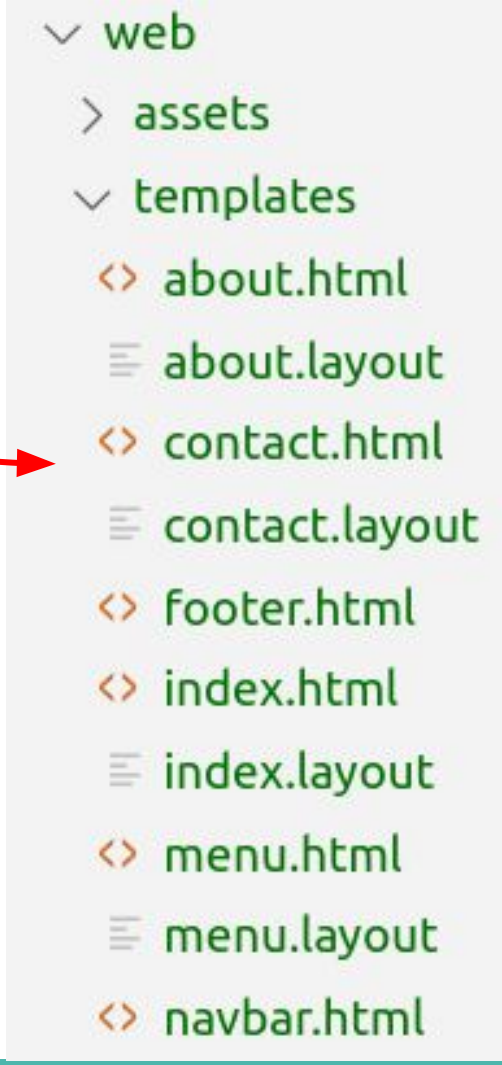
Template Layout

Put the following content inside **`footer.html`** file

```
{{ define "footer" }}
<footer>
    <div class="container">
        <div class="row">
            <div class="col-md-12">
                <p class="copyright">&copy; 2019 Web Programm
            </div>
        </div>
    </div>
</footer>
<!-- js -->
<script src="/assets/js/jquery-3.2.1.slim.min.js"></script>
<script src="/assets/js/bootstrap.min.js"></script>
</body>
</html>
{{ end }}
```

Template Layout

Create the layout and content of **other pages** such as **`about`**, **`contact`**, and **`menu`** in similar manner



Testing the layout

In the `main.go` file parse the templates, provide a handler for each route as shown below

```go
var tmpl = template.Must(template.ParseGlob("delivery/web/templates/*"))

func index(w http.ResponseWriter, r *http.Request) {
    tmpl.ExecuteTemplate(w, "index.layout", nil)
}

func about(w http.ResponseWriter, r *http.Request) {
    tmpl.ExecuteTemplate(w, "about.layout", nil)
}
```

Testing the layout

In the `main.go` file parse the templates, provide a handler for each route as shown below

```go
18  func contact(w http.ResponseWriter, r *http.Request) {
19      tmpl.ExecuteTemplate(w, "contact.layout", nil)
20  }
21
22  func menu(w http.ResponseWriter, r *http.Request) {
23      tmpl.ExecuteTemplate(w, "menu.layout", nil)
24  }
```

Testing the layout

The main function looks like the following (run the main.go file and check the routes)

```go
26  func main() {
27      fs := http.FileServer(http.Dir("delivery/web/assets"))
28      http.Handle("/assets/", http.StripPrefix("/assets/", fs))
29      http.HandleFunc("/", index)
30      http.HandleFunc("/about", about)
31      http.HandleFunc("/contact", contact)
32      http.HandleFunc("/menu", menu)
33      http.ListenAndServe(":8181", nil)
34  }
```

In-memory storage

Store and Retrieve data stored in memory

The code can be found in the following link

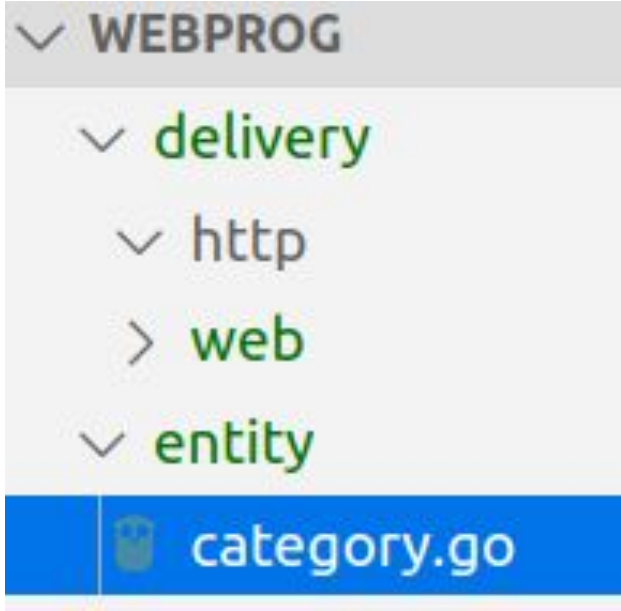
https://github.com/betsegawlemma/web-prog-lab-05-mem

In-memory storage: entity/category.go

Data can be stored in memory for caching purpose

Let us see how store Food Menu **Category** data in memory

Inside the `entity` directory create a `category.go` file



In-memory storage: Category

Write the following code inside `category.go`

Notice the name of the package at the top

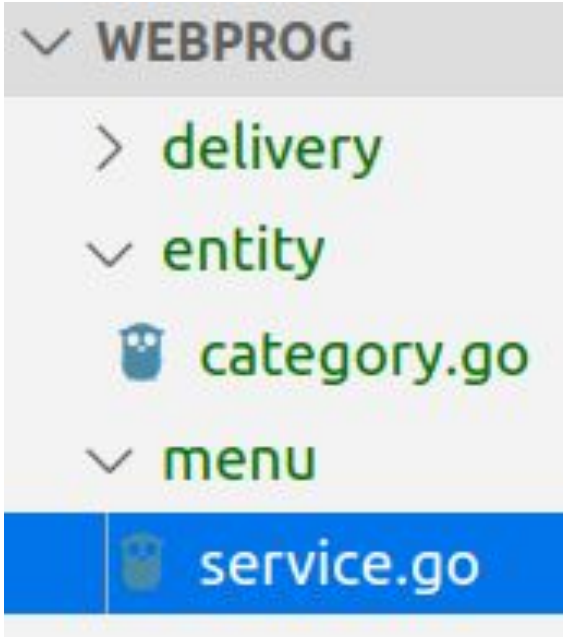
entity > category.go > ...

```go
package entity

// Category represents Food Menu Category
type Category struct {
    ID          int
    Name        string
    Description string
    Image       string
}
```

In-memory storage: menu/service.go

Inside the menu directory create a file named `service.go` which defines food menu related services

Inside `service.go` we will define methods related to storing and retrieving food menu categories



In-memory storage: CategoryService interface

Write the following interface definition Inside `service.go`

Notice the import statement

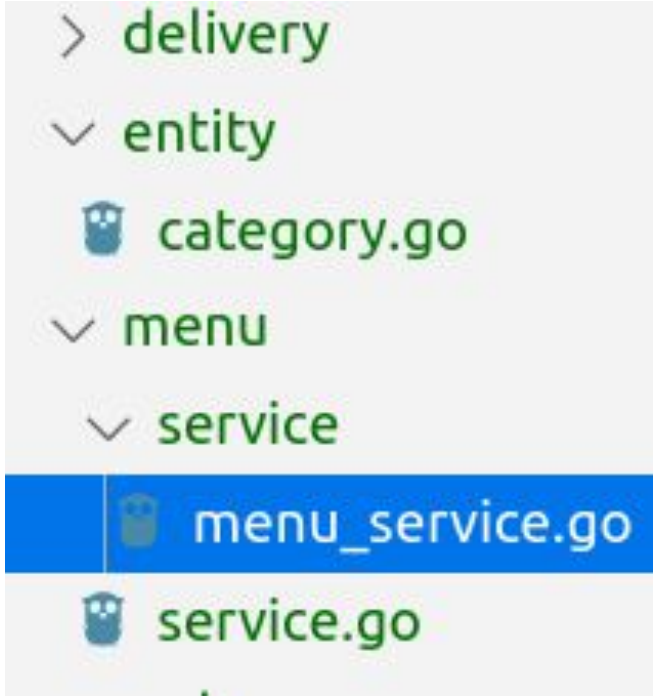
```go
package menu

import "github.com/betsegawlemma/webprog/entity"

// CategoryService specifies food menu category related services
type CategoryService interface {
    Category(id int) (*entity.Category, error)
    StoreCategory(category *entity.Category) error
}
```

In-memory storage: CategoryService implemention

Let's implement the `CategoryService` interface

Inside the `menu` directory create a directory named `service`

Inside the `service` directory create a file named `menu_service.go`



In-memory storage: CategoryService implemention

Steps for implementing CategoryService

- Create a `CategoryCache` map container type
- Create a constructor which creates a `new CategoryCache` map container
- Implement the `Category` and `StoreCategory` methods of the `CategoryService` interface shown below

```
type CategoryService interface {
    Category(id int) (*entity.Category, error)
    StoreCategory(category *entity.Category) error
}
```

In-memory storage: CategoryService implemention

Create a `CategoryCache` map container type inside the `menu_service.go` file

```go
package service

import (
    "errors"

    "github.com/betsegawlemma/webprog/entity"
)

// CategoryCache provides an in-memory cache
type CategoryCache map[int]*entity.Category

```

In-memory storage: CategoryService implemention

Next add a constructor which creates a **`new`** **`CategoryCache`** map container in side the **`menu_service.go`** file

```
12  // NewCategoryCache returns a new category cache
13  func NewCategoryCache() CategoryCache {
14      return make(map[int]*entity.Category)
15  }
```

In-memory storage: CategoryService implemention

Then implement the `Category` interface as a method on `CategoryCache` map container

```go
// Category returns a category for a given id from the cache
func (c CategoryCache) Category(id int) (*entity.Category, error) {
    if cat, ok := c[id]; ok {
        return cat, nil
    }
    return nil, errors.New("Category was not found")
}
```

In-memory storage: CategoryService implemention

Finaly implement the `StoreCategory` interface as a method on `CategoryCache` map container

```go
// StoreCategory stores category data to the cache
func (c CategoryCache) StoreCategory(category *entity.Category) error {
    if _, ok := c[category.ID]; !ok {
        c[category.ID] = category
        return nil
    }
    return errors.New("Category already exists")
}
```

In-memory storage: main.go

We can now use the cache container to store and retrieve food menu categories

We will use the following URLs

```
http://localhost:8181/?id=1

http://localhost:8181/?id=2

http://localhost:8181/?id=3

http://localhost:8181/?id=4
```

In-memory storage: main.go

Steps:

Get a handle of the cache container map from the `service` package

Store some initial data in the container using the `init()` function

Retrieve the stored data from cache when Get request arrives at the URLs shown in the previous slide

In-memory storage: main.go

Get a handle of the cache container map from the `service` package

```
1   package main
2
3   import (
4       "html/template"
5       "net/http"
6       "strconv"
7
8       "github.com/betsegawlemma/webprog/entity"
9       "github.com/betsegawlemma/webprog/menu/service"
10  )
11
12  var tmpl = template.Must(template.ParseGlob("delivery
13  var categoryCache service.CategoryCache
```

Container

In-memory storage: main.go

Store some initial data in the container using the `init()` function

```
29  func init() {
30      categoryCache = service.NewCategoryCache()
31      breakfast := entity.Category{ID: 1, Name: "Breakfast",
32          Description: "Lorem ipsum", Image: "bkt.png"}
33      lunch := entity.Category{ID: 2, Name: "Lunch",
34          Description: "Lorem ipsum", Image: "lnc.png"}
35      categoryCache.StoreCategory(&breakfast)
36      categoryCache.StoreCategory(&lunch)
37  }
```

Invoking the constructor

Storing data in the container

In-memory storage: main.go

Handling requests

```go
15  func index(w http.ResponseWriter, r *http.Request) {
16
17      idRaw := r.URL.Query().Get("id")
18      id, err := strconv.Atoi(idRaw)
19      if err != nil {
20          id = 1
21      }
22      cat, err := categoryCache.Category(id)
23      if err != nil {
24          panic(err)
25      }
26      tmpl.ExecuteTemplate(w, "index.layout", cat)
27  }
28
```

In-memory storage: main.go

The `index.html` file looks like the following

```
{{ define "index.content"}}
<div class="jumbotron jumbotron-fluid">
    <div class="container">
        <h1 class="display-4">Lorem Ipsum</h1>
        <p class="lead">Curabitur justo erat, sodale
    </div>
</div>
<div class="container">
    <hr>
    <p><b>Name:</b> {{ .Name }}</p>
    <p><b>Description:</b> {{ .Description }}</p>
    <img src="/assets/img/{{ .Image }}"/>
</div>
{{ end }}
```

In-memory storage: main.go

Main function

```go
func main() {
    fs := http.FileServer(http.Dir("delivery/web/assets"))
    http.Handle("/assets/", http.StripPrefix("/assets/", fs))
    http.HandleFunc("/", index)
    http.ListenAndServe(":8181", nil)
}
```

Run the server and send the following requests on your browser

http://localhost:8181/?id=1

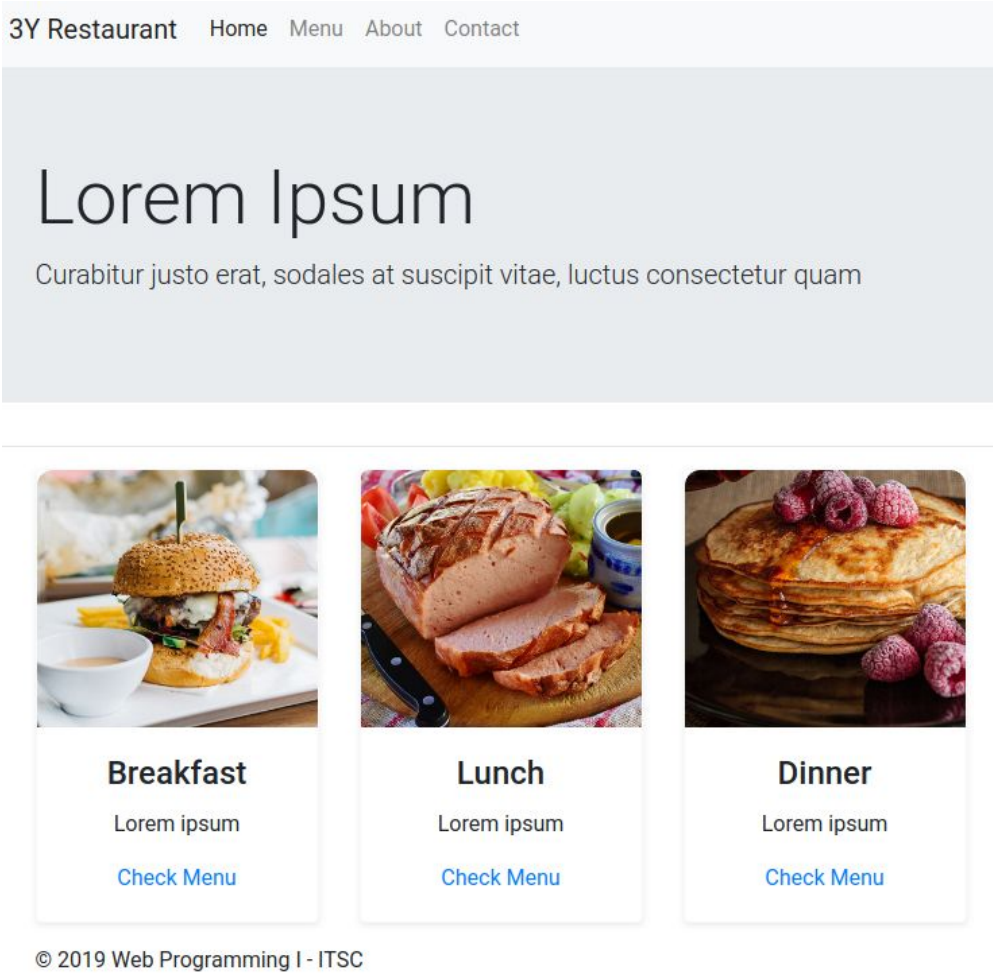
http://localhost:8181/?id=2

CSV File Storage

Store and retrieve data using CSV file

The code can be found in the following link

https://github.com/betsegawlemma/web-prog-lab-05-csv



CSV File Storage: CategoryService interface

The service interface now looks like the following

For simplicity we want to read/write food categories at once

```
3   import "github.com/betsegawlemma/webprog/entity"
4
5   // CategoryService specifies food menu category related services
6   type CategoryService interface {
7       Categories() ([]entity.Category, error)
8       StoreCategores(categoies []entity.Category) error
9   }
```

CSV File Storage: CategoryService interface

Steps:

Create `CategoryService` **struct** that represent the implementation of the `CategoryService` **interface**

Create a **constructor** that initialize the `CategoryService` struct

Implement the **methods** defined in the `CategoryService` interface on the the `CategoryService` struct

CSV File Storage: CategoryService struct

FileName is the name of the file that will store the csv data

```go
package service

import (
    "encoding/csv"
    "errors"
    "os"
    "strconv"

    "github.com/betsegawlemma/webprogcsv/entity"
)

// CategoryService represents csv implementation of
type CategoryService struct {
    FileName string
}
```

CSV File Storage: Constructor

```go
// NewCategoryService returns new Category Service
func NewCategoryService(fileName string) *CategoryService {
    return &CategoryService{FileName: fileName}
}
```

CSV File Storage: Categories method

```go
// Categories returns all categories read from csv file
func (cs CategoryService) Categories() ([]entity.Category, error) {
    file, err := os.Open(cs.FileName)
    if err != nil {
        return nil, errors.New("File could not be open")
    }
    defer file.Close()
    reader := csv.NewReader(file)
    reader.FieldsPerRecord = -1
    record, err := reader.ReadAll()
    if err != nil {
        return nil, errors.New("File could not be open")
    }
```

CSV File Storage: Categories method

```go
    for _, item := range record {
        id, _ := strconv.ParseInt(item[0], 0, 0)
        c := entity.Category{ID: int(id), Name: item[1],
            Description: item[2], Image: item[3]}
        ctgs = append(ctgs, c)
    }
    return ctgs, nil
}
```

CSV File Storage: StoreCategories method

```go
// StoreCategories stores a batch of categories data to the a csv file
func (cs CategoryService) StoreCategories(ctgs []entity.Category) error {
    csvFile, err := os.Create(cs.FileName)
    if err != nil {
        return errors.New("File could not be created")
    }
    defer csvFile.Close()
    writer := csv.NewWriter(csvFile)
    for _, c := range ctgs {
        line := []string{strconv.Itoa(c.ID), c.Name, c.Description, c.Image}
        writer.Write(line)
    }
    writer.Flush()
    return nil
}
```

CSV File Storage: main.go

Steps:

Get a handle of the `CategoryService` struct from the `service` package

Store some initial data in the `csv` file using the `init()` function

Retrieve the stored data from the `csv` file when Get request arrives at the following URLs

```
http://localhost:8181/
```

CSV File Storage: main.go

Get a handle of the `CategoryService` struct from the `service` package

```
3   import (
4       "html/template"
5       "net/http"
6
7       "github.com/betsegawlemma/webprogcsv/entity"
8       "github.com/betsegawlemma/webprogcsv/menu/service"
9   )
10
11  var tmpl = template.Must(template.ParseGlob("delivery/web/templates/*"))
12  var categoryService *service.CategoryService
```

CSV File Storage: main.go

Store some initial data in the `csv` file using the `init()` function

```
23  func init() {
24      categoryService = service.NewCategoryService("category.csv")
25      categories := []entity.Category{
26          entity.Category{ID: 1, Name: "Breakfast",
27              Description: "Lorem ipsum", Image: "bkt.png"},
28          entity.Category{ID: 2, Name: "Lunch",
29              Description: "Lorem ipsum", Image: "lnc.png"},
30      }
31      categoryService.StoreCategories(categories)
32  }
```

CSV File Storage: main.go

Handle the request arriving at **http://localhost:8181/**

```
14  func index(w http.ResponseWriter, r *http.Request) {
15
16      categories, err := categoryService.Categories()
17      if err != nil {
18          panic(err)
19      }
20      tmpl.ExecuteTemplate(w, "index.layout", categories)
21  }
22
```

CSV File Storage

The **`index.html`** file looks like the following

Put the table below the jumbotron content

```
<div class="container">
    <table>
        <thead>
            <td></td>
            <td>Name</td>
            <td>Description</td>
        </thead>
        {{ range . }}
        <tr>
            <td>
                <img src="/assets/img/{{ .Image }}"/>
            </td>
            <td>{{ .Name }}</td>
            <td>{{ .Description }}</td>
        </tr>
        {{ end }}
    </table>
</div>
{{ end }}
```

CSV File Storage: main.go

The **`main`** function looks like the following

```go
34  func main() {
35      fs := http.FileServer(http.Dir("delivery/web/assets"))
36      http.Handle("/assets/", http.StripPrefix("/assets/", fs))
37      http.HandleFunc("/", index)
38      http.ListenAndServe(":8181", nil)
39  }
```

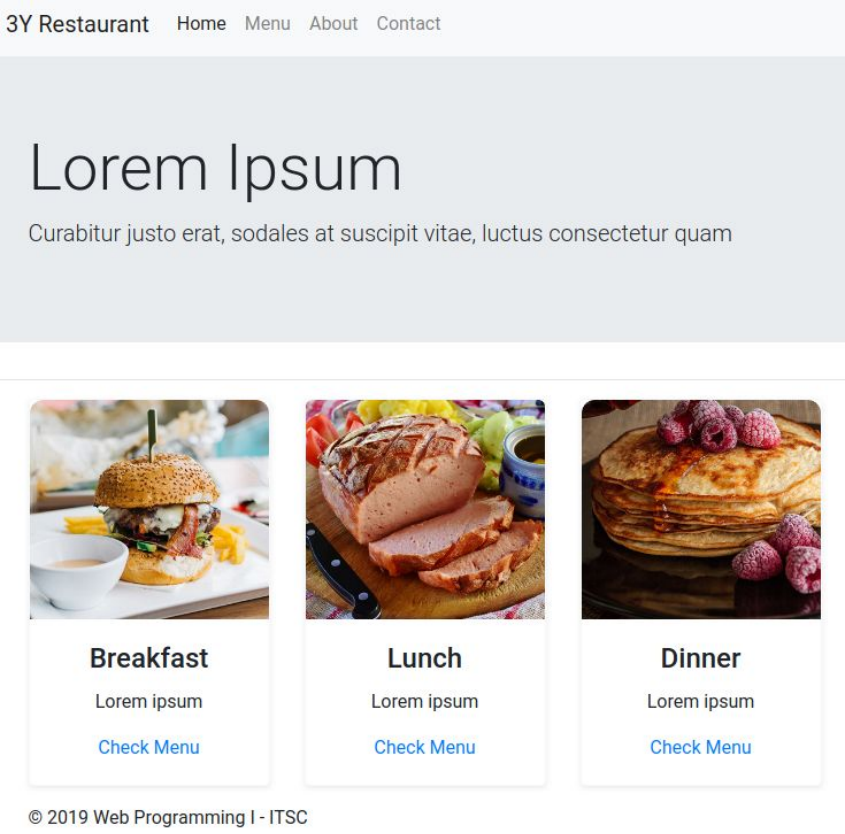
File storage using gob package

Assignment

You can use the same approach used for the csv file to store data using gob package

Solution (Try by yourself before checking the solution)

https://github.com/betsegawlemma/web-prog-lab-05-gob



Reference

https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

https://medium.com/@benbjohnson/standard-package-layout-7cdbc8391fc1

https://hackernoon.com/golang-clean-architecture-efd6d7c43047

https://hackernoon.com/trying-clean-architecture-on-golang-2-44d615bf8fdf

https://github.com/bxcodec/go-clean-arch