Go Applications

Architecting Application, Page Layout, Data Store

Learning Objective

After completing this session you should be able to explain

- Different architectures of enterprise applications
- Characteristics of good application architecture
- Different ways of storing data

Types of Architectures

Different architectures to choose from

Hexagonal Architecture

Onion Architecture

Data, Context and Interaction (DCI) architecture

Boundary Control Entity (BCE) architecture

...

Architecture

What is Architecture?

Why it is important?

What are the characteristics of a good architecture?

Types of Architectures

Theses all architectures have the same objective, help achieve **separation of concerns**

They all help achieve this separation by **dividing the software into layers**

Clean Architecture Constraints

These architectures helps to produce systems that are

Independent of Frameworks

The architecture does not depend on the existence of some library of feature laden software

Testable

The business rules can be tested without the UI, Database, Web Server, or any other external element

Clean Architecture Constraints

These architectures helps to produce systems that are

Independent of UI

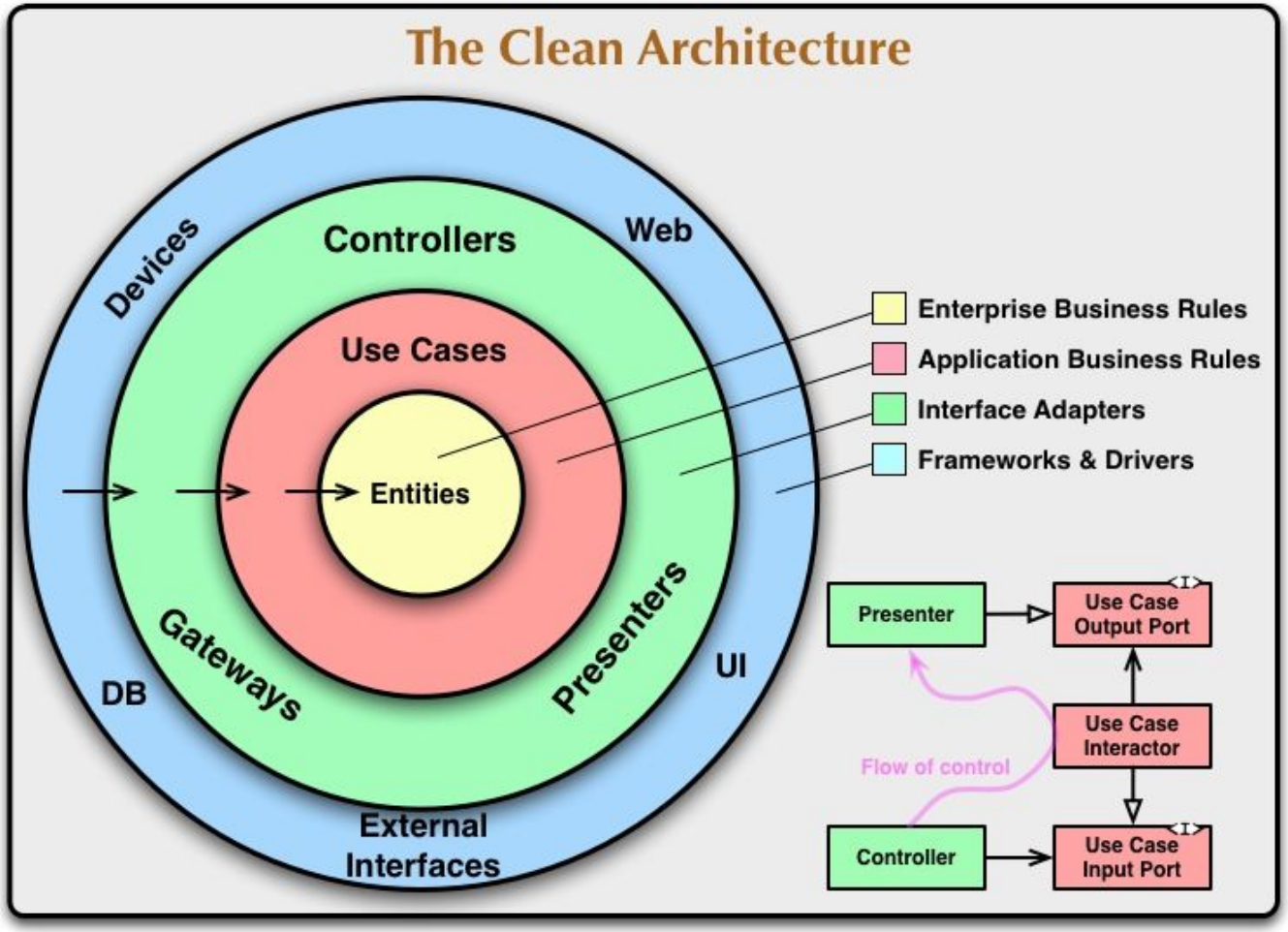
The UI can change easily, without changing the rest of the system

Independent of Database

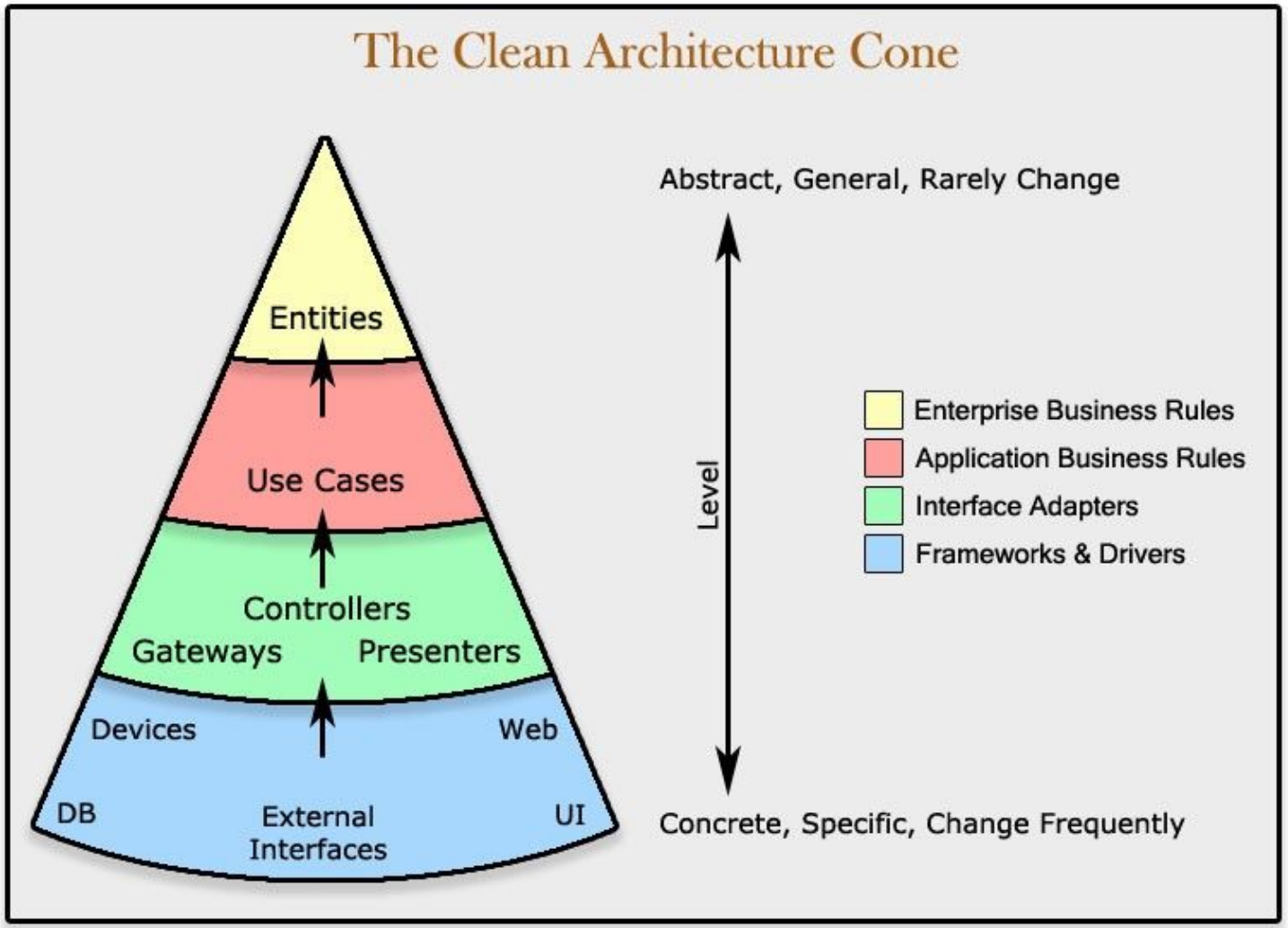
You can swap out Oracle or SQL Server, for Mongo, BigTable, CouchDB, or something else

Independent of any external agency

Your business rules should not know anything about the outside world



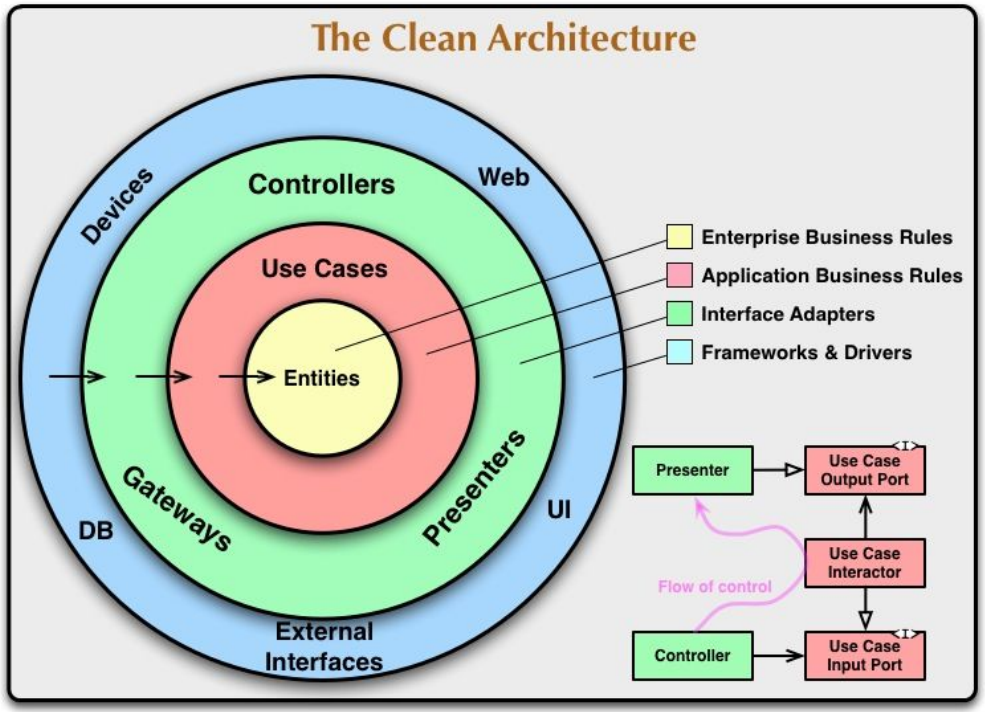
https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html


The Clean Architecture Cone
Entities
Use Cases
Controllers
Gateways
Presenters
Devices
Web
DB
External
Interfaces
UI
Abstract, General, Rarely Change
Level
Concrete, Specific, Change Frequently
Enterprise Business Rules
Application Business Rules
Interface Adapters
Frameworks & Drivers

Dependency Rule of Clean Architecture

Source code dependencies can only **point inwards**

Nothing in an inner circle can know anything at all about something in an outer circle



Layers

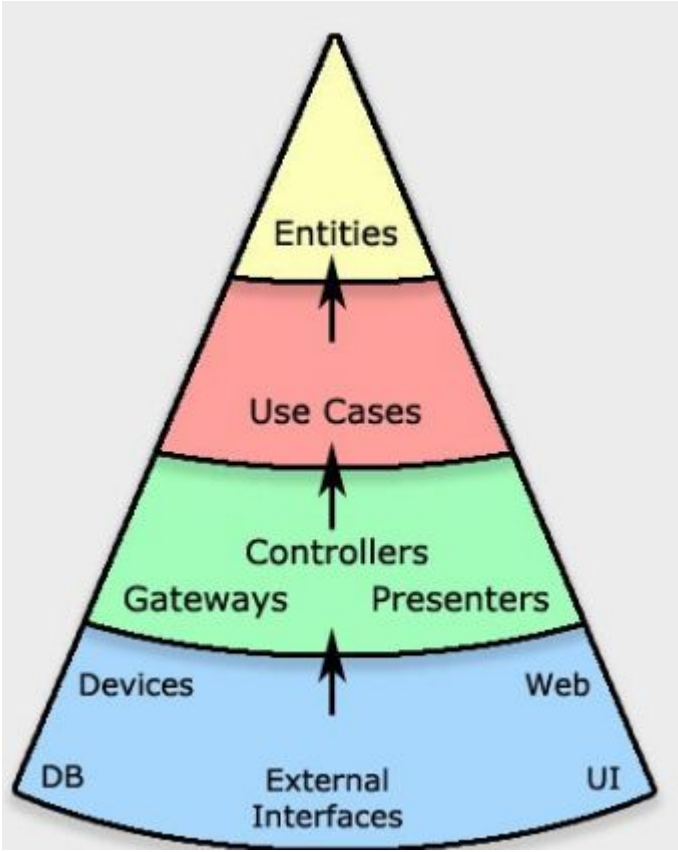
Entities

Encapsulate Enterprise wide business rules

They can be objects with methods, or a set of data structures and functions

If you don’t have an enterprise, and are just writing a single application, then these entities are the business objects of the application

They encapsulate the most general and high-level rules



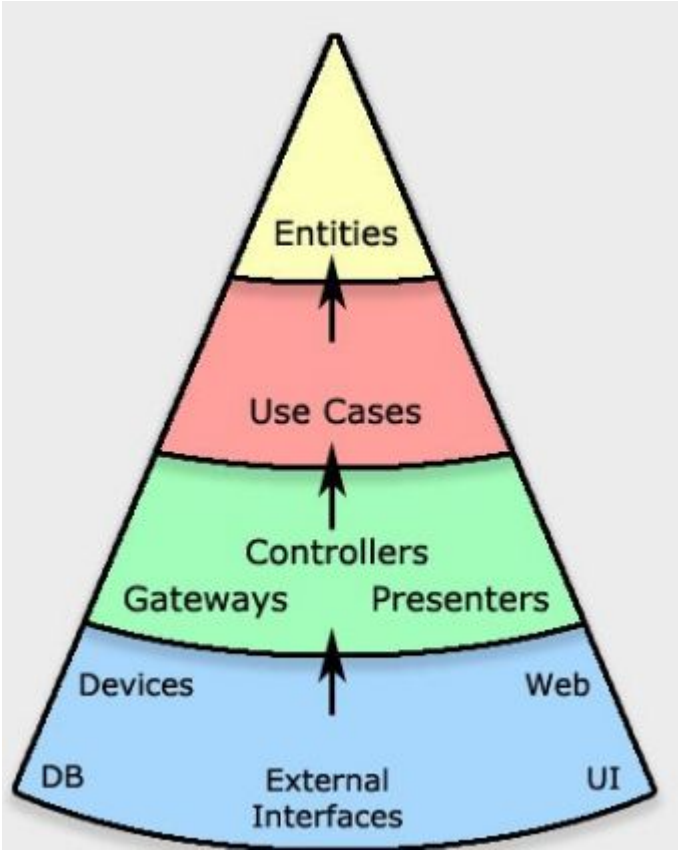
Layers

Use Cases

This layer contains application specific business rules

It encapsulates and implements all of the use cases of the system

These use cases orchestrate the flow of data to and from the entities, and direct those entities to use their enterprise wide business rules to achieve the goals of the use case



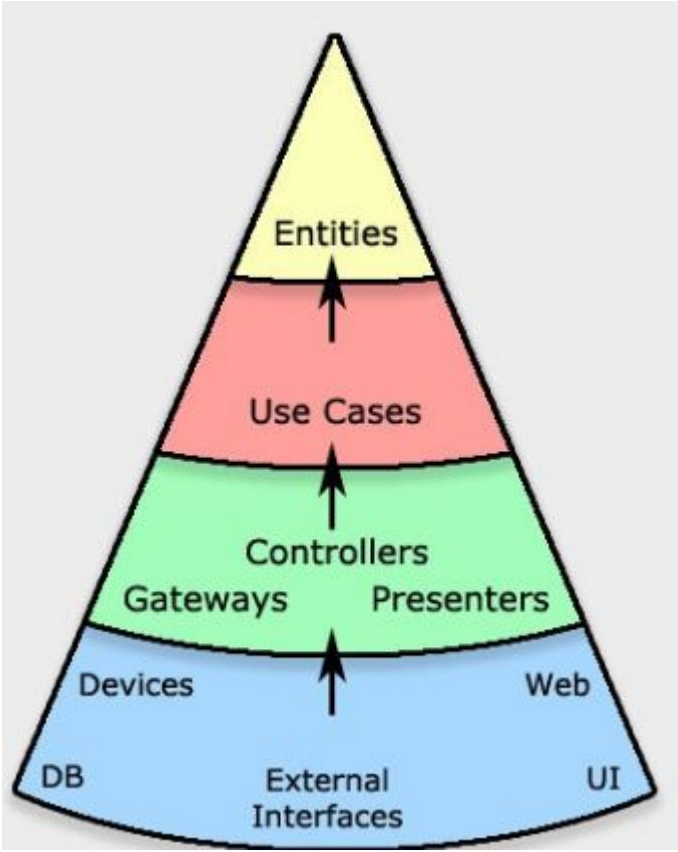
Layers

Use Cases

Changes in this layer should not affect the entities

This layer should not be affected by changes to externalities such as the database, the UI, or any of the common frameworks

We expect that changes to the operation of the application will affect the use-cases and therefore the software in this layer



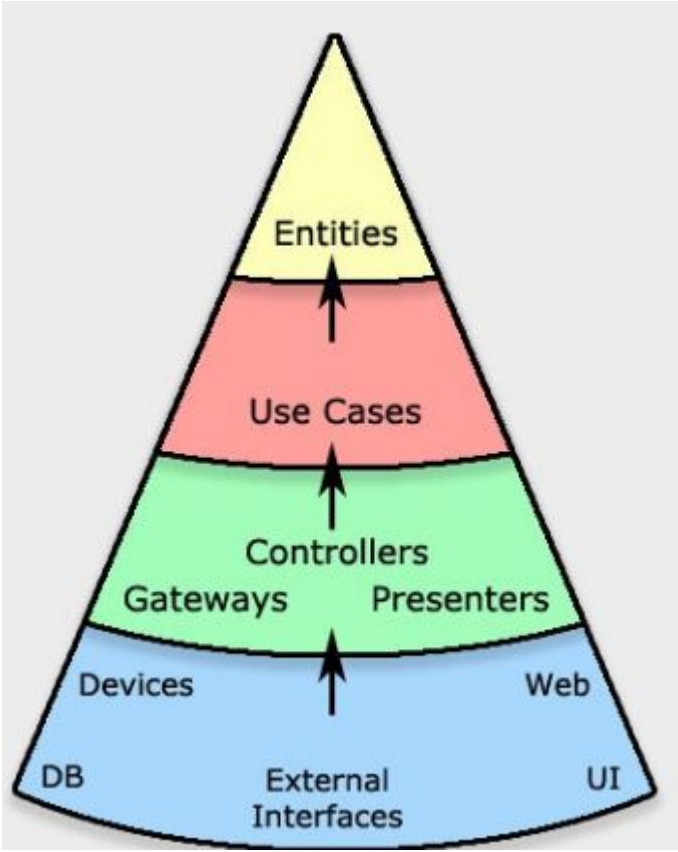
Layers

Interface Adapters

This layer is a set of adapters that convert data from the format most convenient for the use cases and entities, to the format most convenient for some external agency such as the Database or the Web

It is this layer, for example, that will wholly contain the MVC architecture of a GUI

The Presenters, Views, and Controllers all belong in here

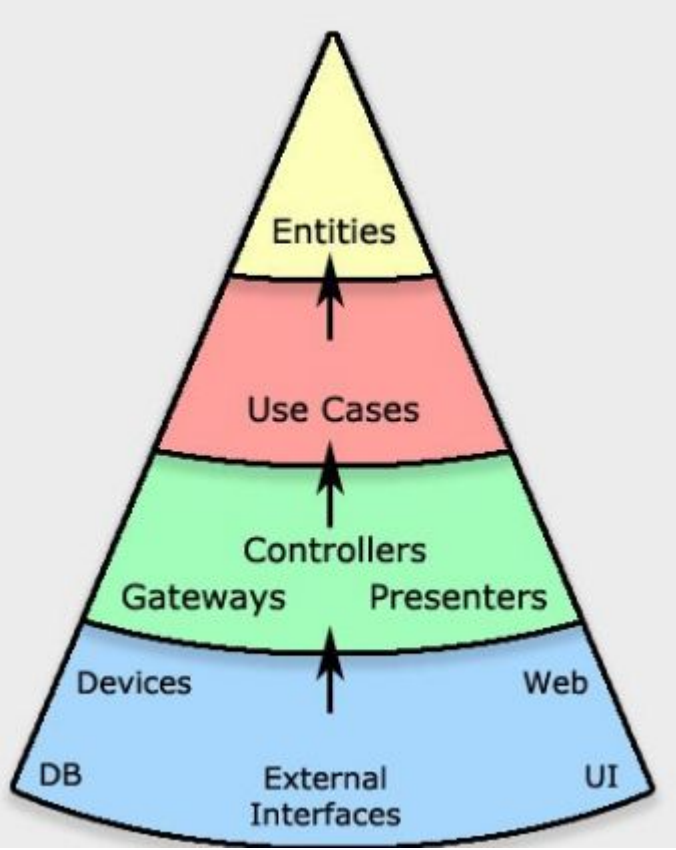


Layers

Interface Adapters

The models are data structures that are passed from the controllers to the use cases, and then back from the use cases to the presenters and views

No code inward of this circle should know anything at all about the database



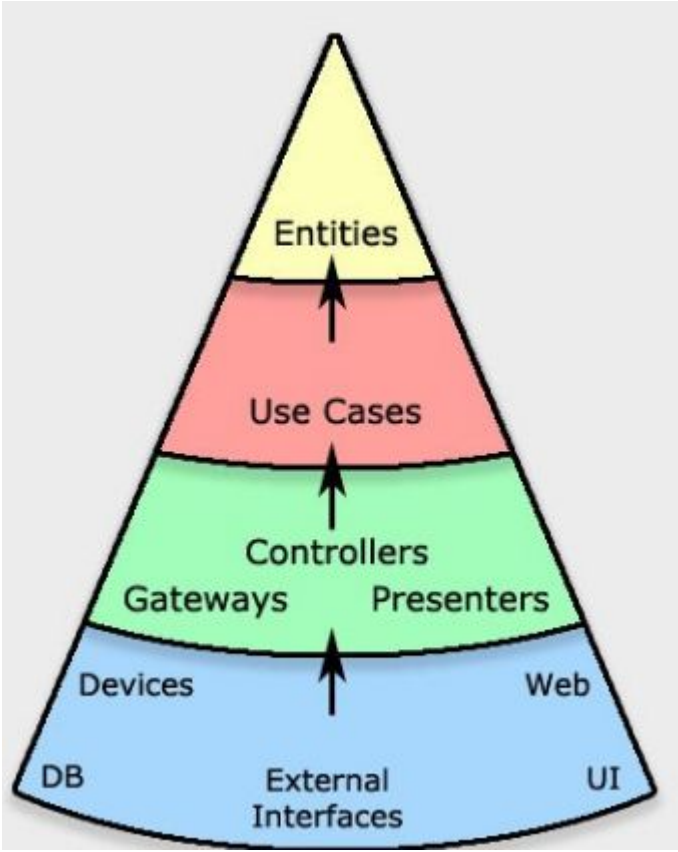
Layers

Frameworks and Drivers/Infrastructures

The outermost layer is generally composed of frameworks and tools such as the Database, the Web Framework, etc.

This layer is where all the details go. The Web is a detail. The database is a detail.

We keep these things on the outside where they can do little harm



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?

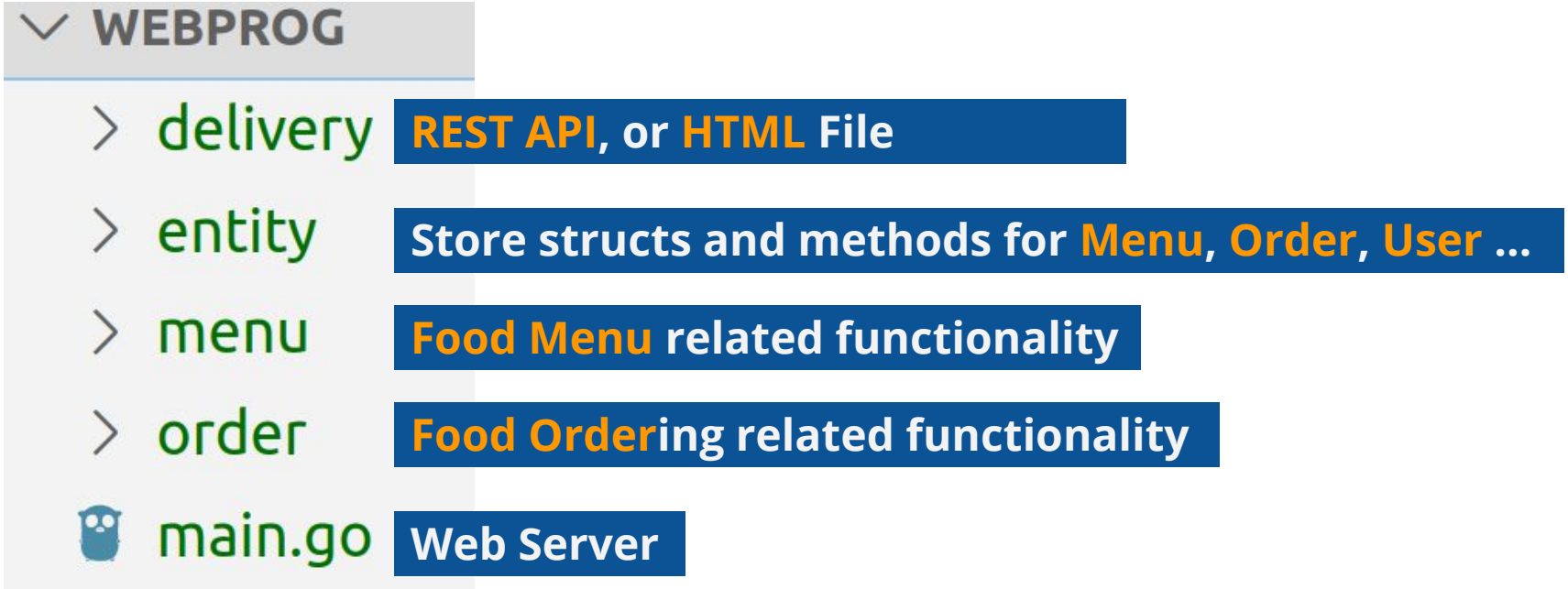
Exercise

Think about the features of your project

What are the functional and nonfunctional requirements?

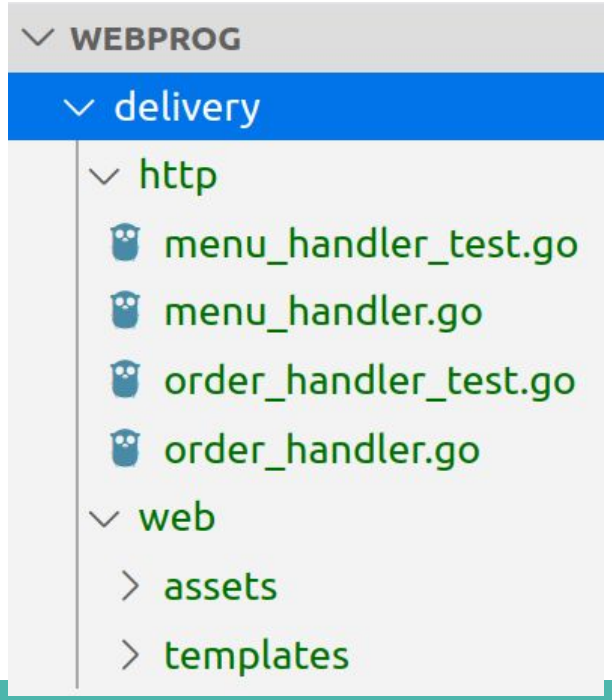
The Architecture of Sample Application

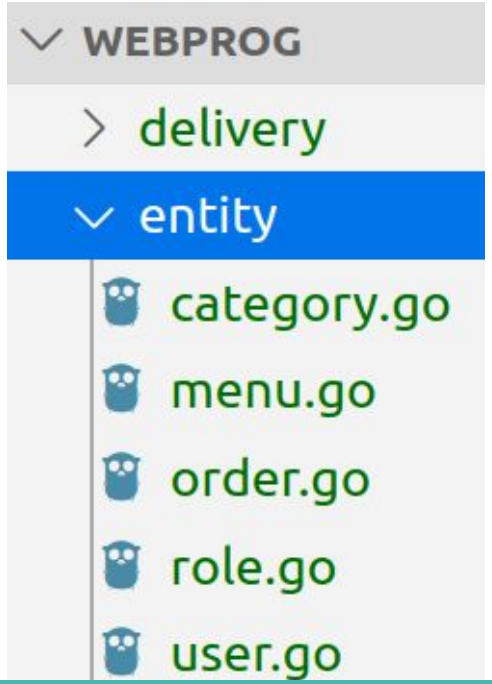
Overview



The Architecture of Sample Application

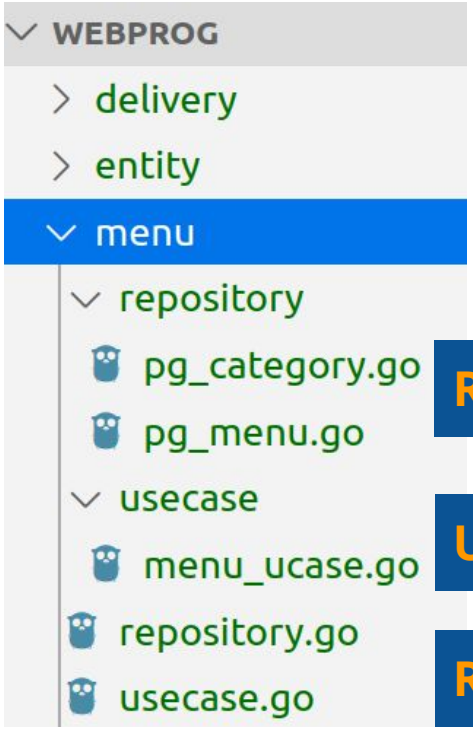
Details of Delivery and Entity layers





The Architecture of Sample Application

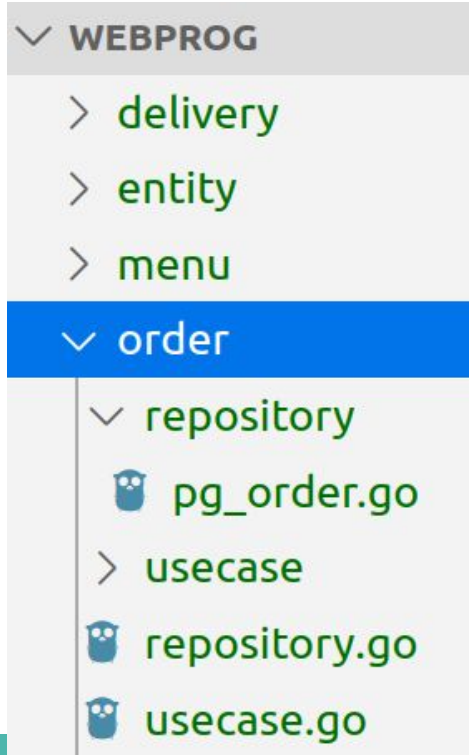
Details for Menu and Order



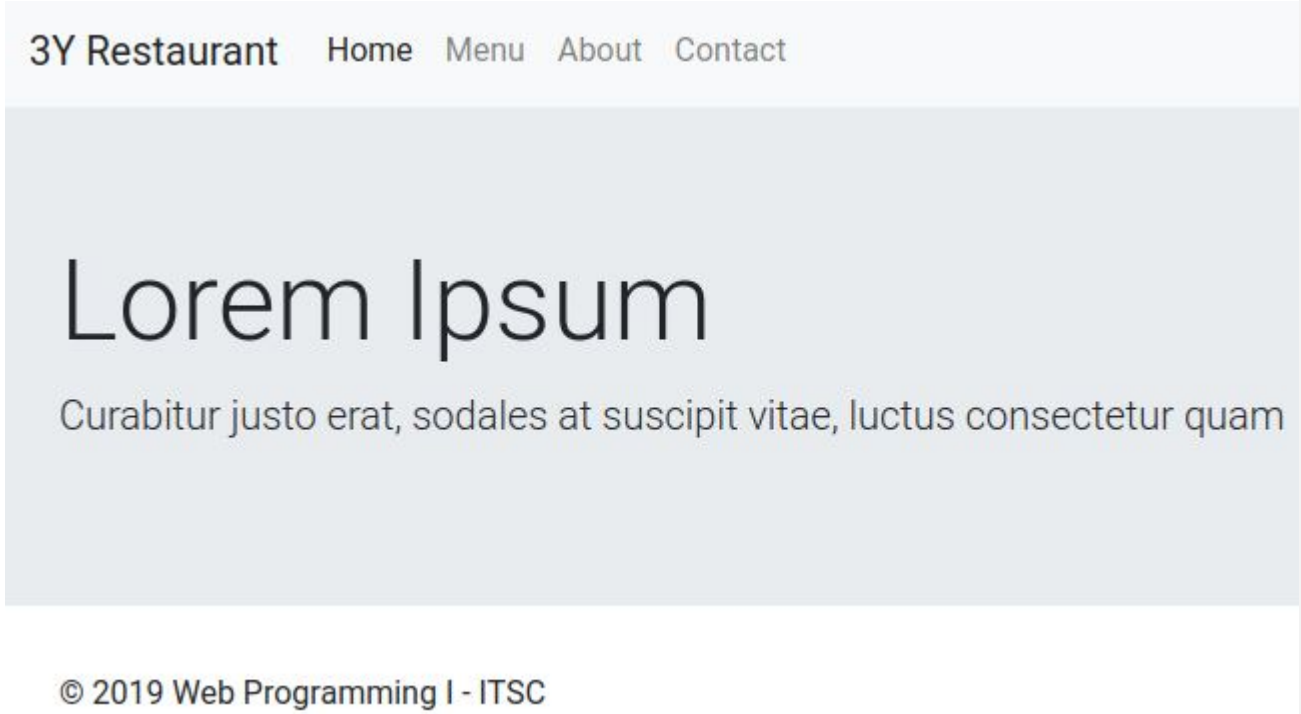
Repository implementation

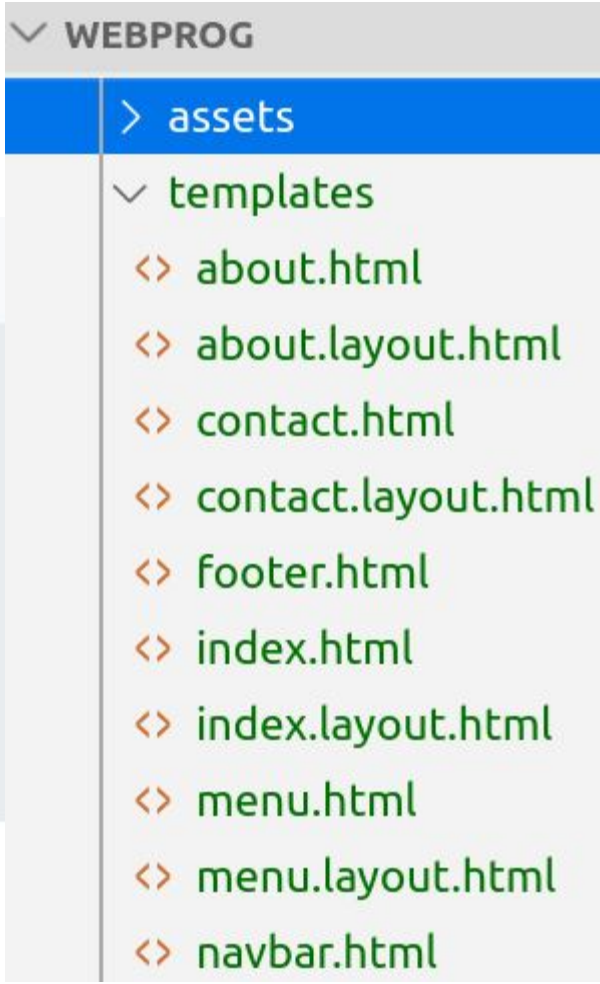
Use Case implementation

Repository and **Use Case** interfaces



Template Layout

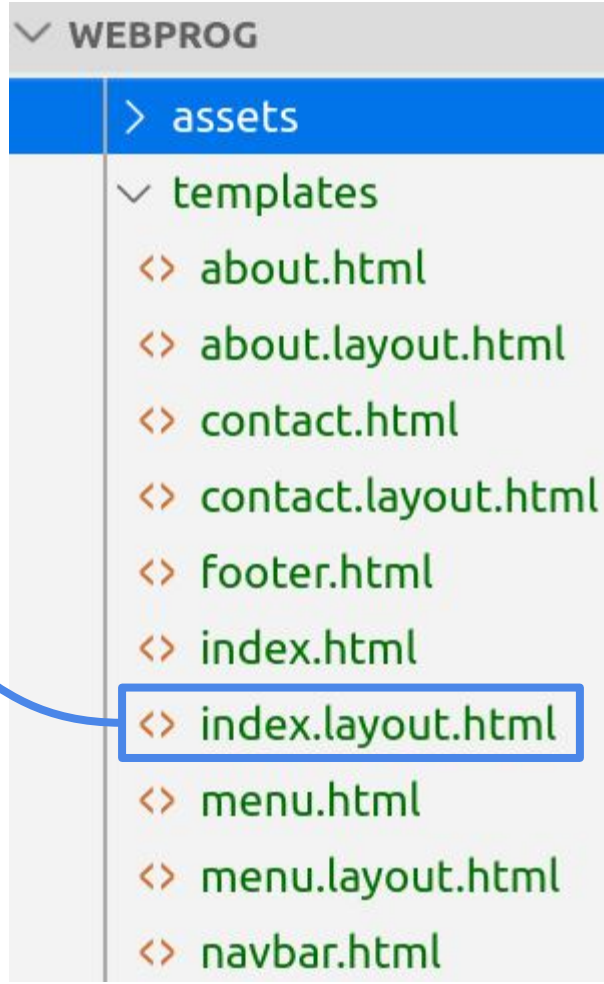




Template Layout

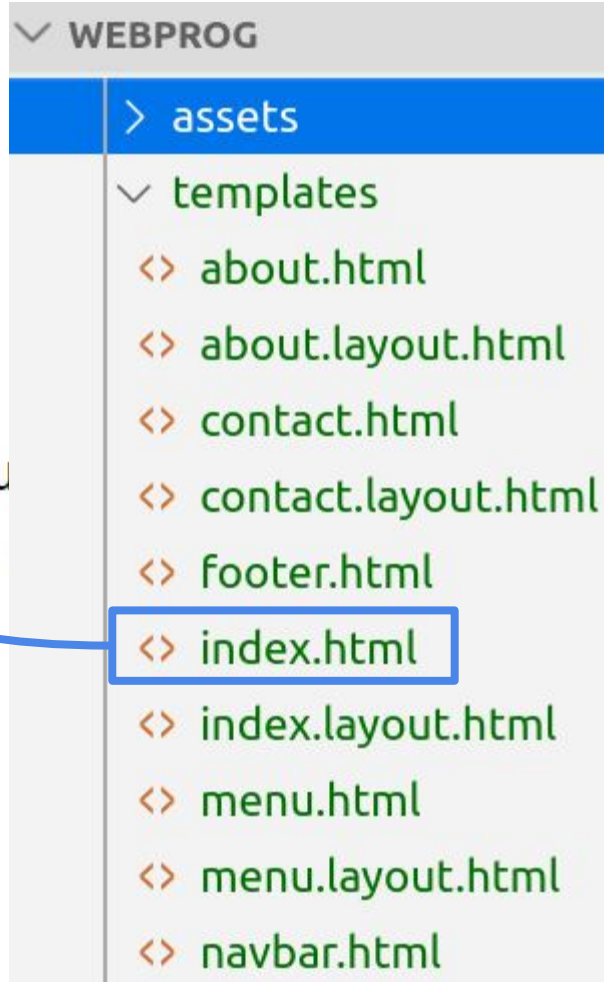
delivery > web > templates > index.layout.html

```
1   {{ define "index.layout" }}
2
3   {{ template "navbar" . }}
4   {{ template "index.content" . }}
5   {{ template "footer" . }}
6
7   {{ end }}
```



Template Layout

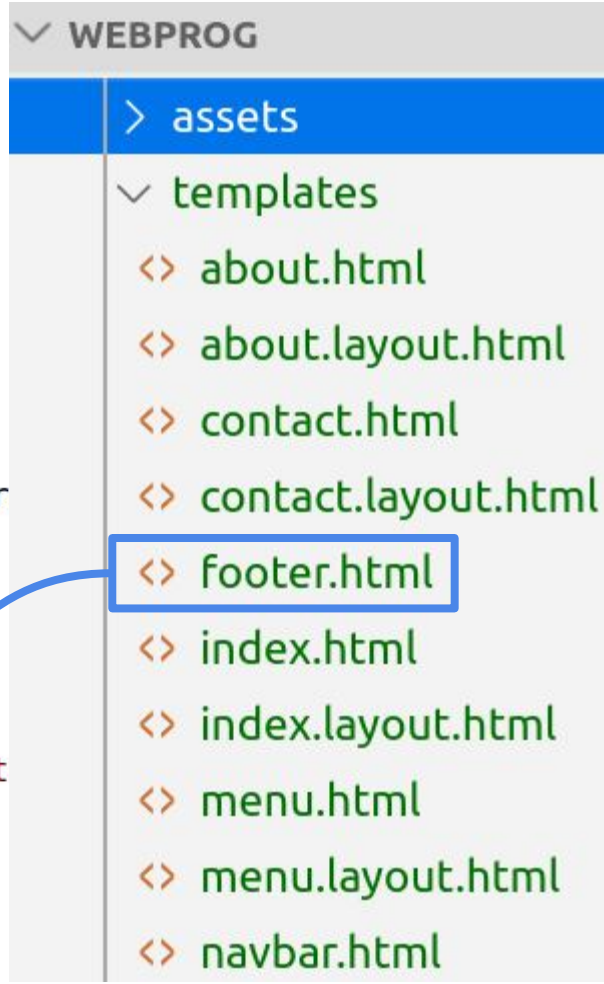
```
1   {{ define "index.content" }}
2   <div class="jumbotron jumbotron-fluid">
3       <div class="container">
4           <h1 class="display-4">Lorem Ipsu
5           <p class="lead">Curabitur justo
6       </div>
7   </div>
8   {{ end }}
```



Template Layout

delivery › web › templates › footer.html › ...

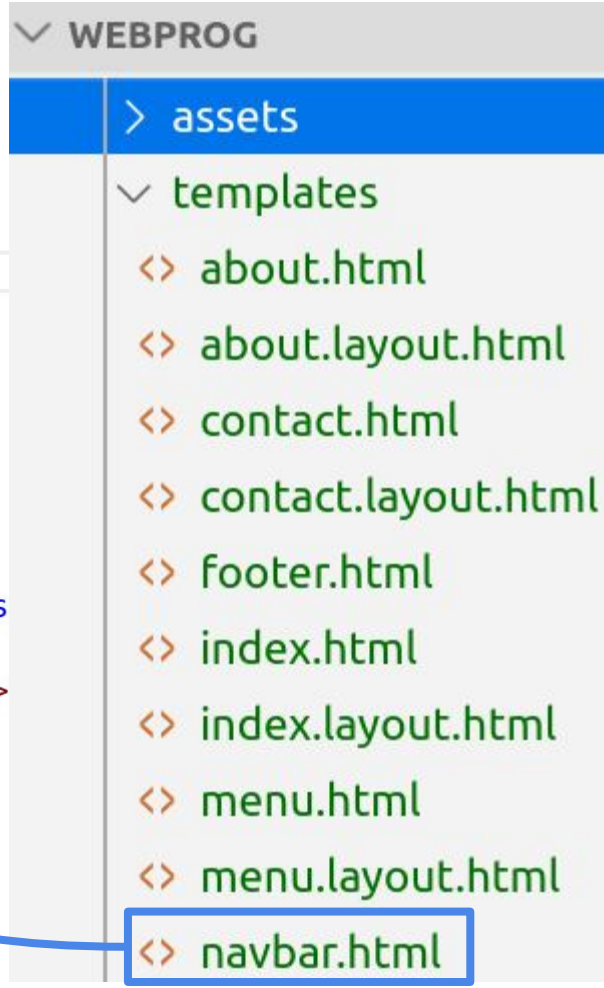
```
1   {{ define "footer" }}
2   <footer>
3       <div class="container">
4           <div class="row">
5               <div class="col-md-12">
6                   <p class="copyright">&copy; 2019 Web Progr
7               </div>
8           </div>
9       </div>
10  </footer>
11  <!-- js -->
12  <script src="/assets/js/jquery-3.2.1.slim.min.js"></script
13  <script src="/assets/js/bootstrap.min.js"></script>
14  </body>
15
16  </html>
17  {{ end }}
```



Template Layout

delivery > web > templates > navbar.html > ...

```
1   {{ define "navbar" }}
2
3   <!DOCTYPE html>
4   <html>
5
6   <head>
7       <title>{{ .Title }}</title>
8       <meta charset="UTF-8">
9       <meta name="viewport" content="width=device-width, initial-s
10      <!-- css -->
11      <link rel="stylesheet" href="/assets/css/bootstrap.min.css">
12  </head>
13
14  <body>
15      <nav class="navbar navbar-expand-lg navbar-light bg-light">
37      </nav>
38
39      {{ end }}
```



Template Layout

All other pages are also constructed in same manner

Parsing and Executing the Template files

```
7
8   var tmpl = template.Must(template.ParseGlob("delivery/web/templates/*.html"))
9
10  func index(w http.ResponseWriter, r *http.Request) {
11      tmpl.ExecuteTemplate(w, "index.layout", nil)
12  }
13
14  func main() {
15      fs := http.FileServer(http.Dir("delivery/web/assets"))
16      http.Handle("/assets/", http.StripPrefix("/assets/", fs))
17      http.HandleFunc("/", index)
18      http.ListenAndServe(":8181", nil)
19  }
```

Storing Data

Storing Data

In-memory storage with `structs`

File storage with `csv` and `gob` binary files

In-memory storage

In-memory data is usually stored in data structures such as `arrays`, `slices`, `maps`, and `structs`

We can utilize in-memory storage for caching the data that we retrieve from the database in order to improve performance

There are dedicated external in-memory database such as `Redis`

One approach for applying in-memory storage is to **use containers** for the individual structs such as **arrays**, **slices**, and **maps** rather than manipulating the individual structs themselves

In-memory storage: Example

```
5   // Category : models Food Menu category
6   type Category struct {
7       ID          int
8       Name        string
9       Description string
10      Image       string
11  }
12
13  // Map containers for storing collection of categories
14  var categoryByID map[int]*Category
15  var categoryByName map[string]*Category
```

In-memory storage: Example

```
17  // Stores a given category to the map containers
18  func store(category Category) {
19      categoryByID[category.ID] = &category
20      categoryByName[category.Name] = &category
21  }
23  func main() {
24
25      // Create the map containers
26      categoryByID = make(map[int]*Category)
27      categoryByName = make(map[string]*Category)
28
29      // ...
30  }
```

In-memory storage: Example

```
30  func main() {
31
32      // ...
33
34      // Create food categories
35      breakfast := Category{ID: 1, Name: "Breakfast", Description: "Breakfast Cateogry", Image: "breakfast.png"}
36      lunch := Category{ID: 2, Name: "Lunch", Description: "Lunch Cateogry", Image: "lunch.png"}
37      dinner := Category{ID: 3, Name: "Dinner", Description: "Dinner Cateogry", Image: "dinner.png"}
38      snack := Category{ID: 4, Name: "Snack", Description: "Snack Cateogry", Image: "snack.png"}
39
40      // ...
41
42  }
```

In-memory storage: Example

```
41  func main() {
42
43      // ...
44
45      // Store food categories to the container
46      store(breakfast)
47      store(lunch)
48      store(dinner)
49      store(snack)
50
51      // Access the map containers
52      fmt.Println(categoryByID[3])
53      fmt.Println(categoryByName["Snack"])
54  }
55
```

Exercise

For what purpose you can use in-memory storage?

What is the advantages of using in-memory storage?

Mention dedicated in-memory storage software

How do you cache data in memory using Go?

File storage

There are two ways to store data to files in Go

CSV (comma-separated values)

used for transferring large data from the user to the system

Using the **gob** package

binary format that can be saved in a file, providing a quick and effective means of serializing in-memory data to one or more files

Reading and writing to a file

Two ways of writing to and reading from a file

- Using **`WriteFile`** and **`ReadFile`** functions from the **`io/ioutil`** package

```
ioutil.WriteFile(filename string, data []byte, perm
os.FileMode) error

ioutil.ReadFile(filename string) ([]byte, error)
```

- Using **`File`** struct
  - You first need to use the **`Open/Create`** function in the **`os`** package

Reading and writing to a file: Example1

```
 8  func main() {
 9
10      error := ioutil.WriteFile("file01", []byte("Hello!\n"), 0644)
11
12      if error != nil {
13          panic(error)
14      }
15
16      data, _ := ioutil.ReadFile("file01")
17
18      fmt.Println(string(data))
19  }
```

Reading and writing to a file: Example1

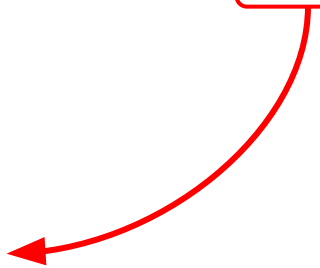
```
 8  func main() {
 9
10      error := ioutil.WriteFile("file01", []byte("Hello!\n"), 0644)
11
12      if error != nil {
13          panic(error)
14      }
15
16      data, _ := ioutil.ReadFile("file01")
17
18      fmt.Println(string(data))
19  }
```

Writing

Reading

Reading and writing to a file: Permission

```
ioutil.WriteFile("file01", []byte("Hello!\n"), 0644)
```



Permission Code Position Designation

File Type	Owner	Group	Other
0	6	4	4

Reading and writing to a file: Permission

Permission Code Position Designation

File Type	Owner	Group	Other
0	6	4	4

Three Permissions for each user

```
R Read (4),        W Write (2),      X Execute (1)
```

```
R (4) + W(2) = 6                Read Only = 4

R (4) + W(2) + X(1) = 7         No Permission = 0
```

Exercise

What is the access permission of `file01` in the following codes?

```
ioutil.WriteFile("file01", []byte("Hello!\n"), 0600)
```

```
ioutil.WriteFile("file01", []byte("Hello!\n"), 0777)
```

What is the effect of the following permission configuration?

```
ioutil.WriteFile("file01", []byte("Hello!\n"), 0555)
```

Reading and writing to a file: Example2

```
 8  func main() {
 9      file02, _ := os.Create("file02")
10      defer file02.Close()
11      nBytes, _ := file02.Write([]byte("Hello Again!\n"))
12
13      f02, _ := os.Open("file02")
14      defer f02.Close()
15      read02 := make([]byte, nBytes)
16      bytesRead, _ := f02.Read(read02)
17
18      fmt.Printf("%d bytes read from file02\n", bytesRead)
19      fmt.Println(string(read02))
20  }
```

Reading and writing to a file: Example2

```
 8  func main() {
 9      file02, _ := os.Create("file02")
10      defer file02.Close()
11      nBytes, _ := file02.Write([]byte("Hello Again!\n"))
12
13      f02, _ := os.Open("file02")
14      defer f02.Close()
15      read02 := make([]byte, nBytes)
16      bytesRead, _ := f02.Read(read02)
17
18      fmt.Printf("%d bytes read from file02\n", bytesRead)
19      fmt.Println(string(read02))
20  }
```

Creating

Writing

Opening

Reading

Exercise

What are the steps you should follow to Read and Write to a file using `file struct`?

Reading and writing CSV files

CSV is widely supported, and most spreadsheet programs, such as Microsoft Excel, Apple Numbers, and Google Sheet, support CSV

In Go, CSV is manipulated by the `encoding/csv` package

```
csv.NewWriter(w io.Writer) *Writer

csv.NewReader(r io.Reader) *Reader
```

Reading and writing CSV files: Example

Let us see how to **write** Food Menu categories into a CSV file

```
10  // Category : models Food Menu category
11  type Category struct {
12      ID          int
13      Name        string
14      Description string
15      Image       string
16  }
```

Reading and writing CSV files: Example

Let us see how to **write** Food Menu categories into a CSV file

```
41  func main() {
42      csvFile, _ := os.Create("categories.csv")
43      defer csvFile.Close()
44
45      writer := csv.NewWriter(csvFile)
46
47      categories := []Category{
48          Category{ID: 3, Name: "Dinner", Description: "Dinner Cateogry", Image: "dnr.png"},
49          Category{ID: 4, Name: "Snack", Description: "Snack Cateogry", Image: "snk.png"},
50      }
51
52      writeToCSVFile(writer, categories)
53  }
```

Next Slide

Reading and writing CSV files: Example

Let us see how to **write** list of Food Menu categories into a CSV file

```
18  func writeToCSVFile(wr *csv.Writer, ctgs []Category) {
19      for _, c := range ctgs {
20          line := []string{strconv.Itoa(c.ID), c.Name, c.Description, c.Image}
21          wr.Write(line)
22      }
23      wr.Flush()
24  }
```

Output file

categories.csv

```
3,Dinner,Dinner Cateogry,dnr.png
4,Snack,Snack Cateogry,snk.png
```

Reading and writing CSV files: Example

Let us see how to **write** Food Menu categories into a CSV file

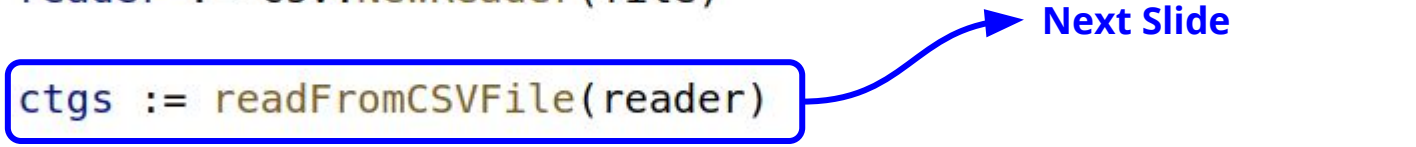
```
18  func writeToCSVFile(wr *csv.Writer, ctgs []Category) {
19      for _, c := range ctgs {
20          line := []string{strconv.Itoa(c.ID), c.Name, c.Description, c.Image}
21          wr.Write(line)
22      }
23      wr.Flush()
24  }
```

To make sure that buffered data is writen to the file

Reading and writing CSV files: Example

Let us see how to **read** Food Menu categories from a CSV file

```
41  func main() {
42
43      file, _ := os.Open("categories.csv")
44      defer file.Close()
45
46      reader := csv.NewReader(file)
47
48      ctgs := readFromCSVFile(reader)
49
50      fmt.Println(ctgs[0].ID, ctgs[0].Name, ctgs[0].Description)
51      fmt.Println(ctgs[1].ID, ctgs[1].Name, ctgs[1].Description)
52  }
```



Reading and writing CSV files: Example

```
26  func readFromCSVFile(rr *csv.Reader) []Category {
27      rr.FieldsPerRecord = -1
28      record, _ := rr.ReadAll()
29
30      var ctgs []Category
31
32      for _, item := range record {
33          id, _ := strconv.ParseInt(item[0], 0, 0)
34          category := Category{ID: int(id), Name: item[1],
35              Description: item[2], Image: item[3]}
36          ctgs = append(ctgs, category)
37      }
38      return ctgs
39  }
```

Reading and writing CSV files: Example

```
26  func readFromCSVFile(rr *csv.Reader) []Category {
27      rr.FieldsPerRecord = -1
28      record, _ := rr.ReadAll()
29
30      var ctgs []Category
31
32      for _, item := range record {
33          id, _ := strconv.ParseInt(item[0], 0, 0)
34          category := Category{ID: int(id), Name: item[1],
35              Description: item[2], Image: item[3]}
36          ctgs = append(ctgs, category)
37      }
38      return ctgs
39  }
```

the number of fields you expect from each record (-1 means any)

Exercise

What are the steps you should follow to Read and Write to a `CSV` file?

What does `writer.Flush()` function do?

What is the purpose of `reader.FieldsPerRecord` field?

The gob package

The `encoding/gob` package manages streams of gobs, which are binary data, exchanged between an encoder and a decoder

It's **designed for serialization and transporting data** but it can also be used for persisting data

Encoders and **decoders** wrap around **writers** and **readers**, which conveniently allows you to use them to write to and read from files

Reading and writing binary data: Example

Store categories of Food Menu using **`encoding/gob`**

```
10  // Category : models Food Menu category
11  type Category struct {
12      ID          int
13      Name        string
14      Description string
15      Image       string
16  }
```

Reading and writing binary data: Example

```
38  func main() {
39      categories := []Category{
40          Category{ID: 3, Name: "Dinner",
41              Description: "Dinner Cateogry", Image: "dnr.png"},
42          Category{ID: 4, Name: "Snack",
43              Description: "Snack Cateogry", Image: "snk.png"},
44      }
45
46      store(categories, "categories.gob")
47      var ctgs []Category
48      load(&ctgs, "categories.gob")
49
50      fmt.Println(ctgs)
51  }
```

Next Slide

Reading and writing binary data: Example

```
18  func store(data interface{}, fileName string) {
19
20      buffer := new(bytes.Buffer)
21
22      encoder := gob.NewEncoder(buffer)
23      encoder.Encode(data)
24
25      ioutil.WriteFile(fileName, buffer.Bytes(), 0600)
26  }
```

Reading and writing binary data: Example

```
18  func store(data interface{}, fileName string) {
19
20      buffer := new(bytes.Buffer)
21
22      encoder := gob.NewEncoder(buffer)
23      encoder.Encode(data)
24
25      ioutil.WriteFile(fileName, buffer.Bytes(), 0600)
26  }
```

variable buffer of bytes that has both Read/Write methods

Reading and writing binary data: Example

```
38  func main() {
39      categories := []Category{
40          Category{ID: 3, Name: "Dinner",
41              Description: "Dinner Cateogry", Image: "dnr.png"},
42          Category{ID: 4, Name: "Snack",
43              Description: "Snack Cateogry", Image: "snk.png"},
44      }
45
46      store(categories, "categories.gob")
47      var ctgs []Category
48      load(&ctgs, "categories.gob")
49
50      fmt.Println(ctgs)
51  }
```

Next Slide

Reading and writing binary data: Example

```
28  func load(data interface{}, fileName string) {
29
30      raw, _ := ioutil.ReadFile(fileName)
31
32      buffer := bytes.NewBuffer(raw)
33
34      dec := gob.NewDecoder(buffer)
35      dec.Decode(data)
36  }
```

Exercise

What is the difference between storing data using **csv** and **gob**?

For what scenario you can use csv or gob file storage?

What are the steps you should follow to save files using **gob** package?

References

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-archithecture-efd6d7c43047

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

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