GO and SQL

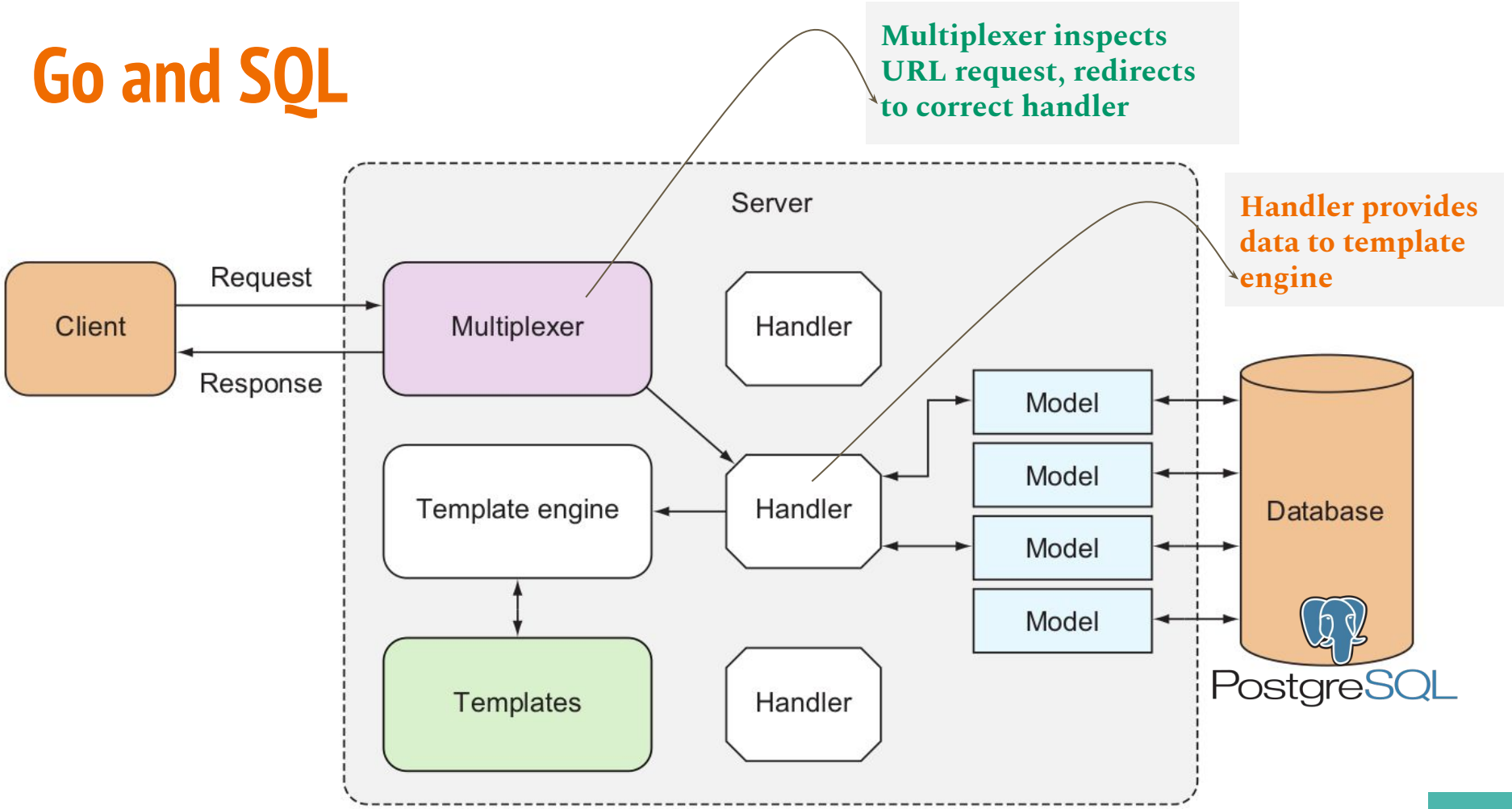
Lecture 07

Learning Outcomes

After Completing this lesson you should be able to

- Perform CRUD operations on PostgreSQL database
- Access form and URL request values
- Upload pictures to Go web server

Go and SQL


Multiplexer inspects URL request, redirects to correct handler
Handler provides data to template engine
Server
Client
Request
Response
Multiplexer
Handler
Handler
Handler
Template engine
Templates
Model
Model
Model
Model
Database
PostgreSQL

Connecting to PostgreSQL from Go code

To connect to PostgreSQL database you first need to download postgres driver. Type the following command inside your project directory

```
go get github.com/lib/pq
```

For this to work correctly, you need to have properly configured **`GOPATH`** environment variable (Refer to the first lab to check how to do)

Connecting to PostgreSQL from Go code

To connect to the database you can use either of the following approaches

```
db, err = sql.Open("postgres", "user=app_user
dbname=restaurantdb password='P@$$w0rdD1'
sslmode=disable"
```

OR

```
db, err := sql.Open("postgres",
"postgres://app_admin:P@$$w0rdD2@localhost/restaurantdb?s
slmode=disable")
```

Home Page

3Y Restaurant Home Menu About Contact

Lorem Ipsum

Curabitur justo erat, sodales at suscipit vitae, luctus consectetur quam

Breakfast

Breakfast Lorem Ipsum

Check Menu

Lunch

Lunch Lorem Ipsum

Check Menu

Dinner

Dinner Lorem ipsum

Check Menu

Snack

Snack Lorem Ipsum

Check Menu

© 2019 Web Programming I - ITSC

Admin Page: Dashboard

3Y Restaurant

Dashboard

Menus

Categories

Orders

Users

Comments

Dashboard

© 2019 Web Programming I - ITSC

Admin Page: Categories

3Y Restaurant

- Dashboard
- Menus
- Categories
- Orders
- Users
- Comments

Menu Categories

C R U D

New Category

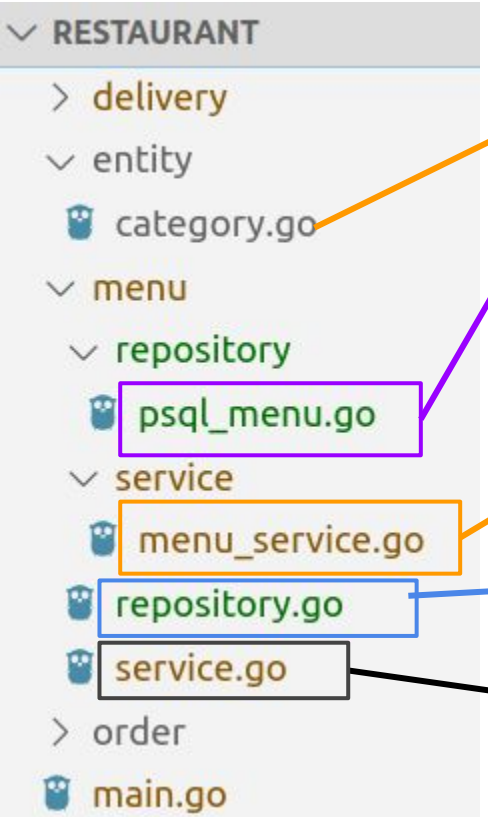
ID	Name	Description	Image		
15	Breakfast	Breakfast Lorem Ipsum		Edit	Delete
16	Lunch	Lunch Lorem Ipsum		Edit	Delete

Paths/Routes

List of Example Paths

Paths/Routes	Descriptions
/	Home page
/admin	Dashboard home
/admin/categories	List all categories
/admin/categories/new	Create new category
/admin/categories/update?id=3	Update a category with id 3
/admin/categories/delete?id=1	Delete a category with id 1

Code Structure



Business Objects/Entities

Implementation of the CategoryRepository interface using PostgreSQL database

Implementation of the service without depending on any particular persistency technology

Defines operations for persisting or accessing Category objects

Defines what the Category service provides

Category Entity

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

CategoryRepository


```go
// CategoryRepository specifies db operations for category
type CategoryRepository interface {
    Categories() ([]entity.Category, error)
    Category(id int) (entity.Category, error)
    UpdateCategory(category entity.Category) error
    DeleteCategory(id int) error
    StoreCategory(category entity.Category) error
}
```

Defines operations for persisting or accessing Category objects

CategoryService


```go
// CategoryService specifies food menu category services
type CategoryService interface {
    Categories() ([]entity.Category, error)
    Category(id int) (entity.Category, error)
    UpdateCategory(category entity.Category) error
    DeleteCategory(id int) error
    StoreCategory(category entity.Category) error
}
```

Defines what the Category service provides

Implementing CategoryRepository

Strategy

Step 1: Create a `struct` whose methods implement the methods defined in the `CategoryRepository` interfaces

Step 2: Create a **constructor** to initialize the **stuct**

Step 3: Implement the methods defined in the interface

Implementing CategoryRepository

Strategy

Step 1: Create a `struct` whose methods implement the database operations defined in the `CategoryRepository` interfaces

```
// PsqlCategoryRepository implements the
// menu.CategoryRepository interface
type PsqlCategoryRepository struct {
    conn *sql.DB
}
```

Implementing CategoryRepository

Strategy

Step 2: Create a **constructor** to initialize the **struct**

```
// NewPsqlCategoryRepository will create an object of PsqlCategoryRepository
func NewPsqlCategoryRepository(Conn *sql.DB) *PsqlCategoryRepository {
    return &PsqlCategoryRepository{conn: Conn}
}
```

Implementing CategoryRepository

Strategy

Step 3: Implement the methods (**Example**: `DeleteCategory`)

```go
// DeleteCategory removes a category from a database by its id
func (pr *PsqlCategoryRepository) DeleteCategory(id int) error {

    _, err := pr.conn.Exec("DELETE FROM categories WHERE id=$1", id)
    if err != nil {
        return errors.New("Delete has failed")
    }

    return nil
```

Executing the Query

Go provides three different methods for executing database queries

- `DB.Query()` is used for **SELECT** queries which return multiple rows
- `DB.QueryRow()` is used for **SELECT** queries which return a single row
- `DB.Exec()` is used for statements which don't return rows (like `INSERT`, `UPDATE` and `DELETE`)

Implementing CategoryService

Strategy

Step 1: Create a `struct` whose methods implement the services defined in the `CategoryService` interfaces

Step 2: Create a **constructor** to initialize the **struct**

Step 3: Implement the methods defined in the interface

Implementing CategoryService

Strategy

Step 1: Create a `struct` whose method implements the services defined in the `CategoryService` interfaces

```
// CategoryService implements menu.CategoryService interface
type CategoryService struct {
    categoryRepo menu.CategoryRepository
}
```

Implementing CategoryService

Strategy

Step 2: Create a **constructor** to initialize the **struct**

```
// NewCategoryService will create new CategoryService object
func NewCategoryService(CatRepo menu.CategoryRepository) *CategoryService {
    return &CategoryService{categoryRepo: CatRepo}
}
```

Implementing CategoryService

Strategy

Step 3: Implement the methods (**Example**: `DeleteCategory`)

```go
// DeleteCategory delete a category by its id
func (cs *CategoryService) DeleteCategory(id int) error {

    err := cs.categoryRepo.DeleteCategory(id)
    if err != nil {
        return err
    }
    return nil
}
```

Usage: `main` function

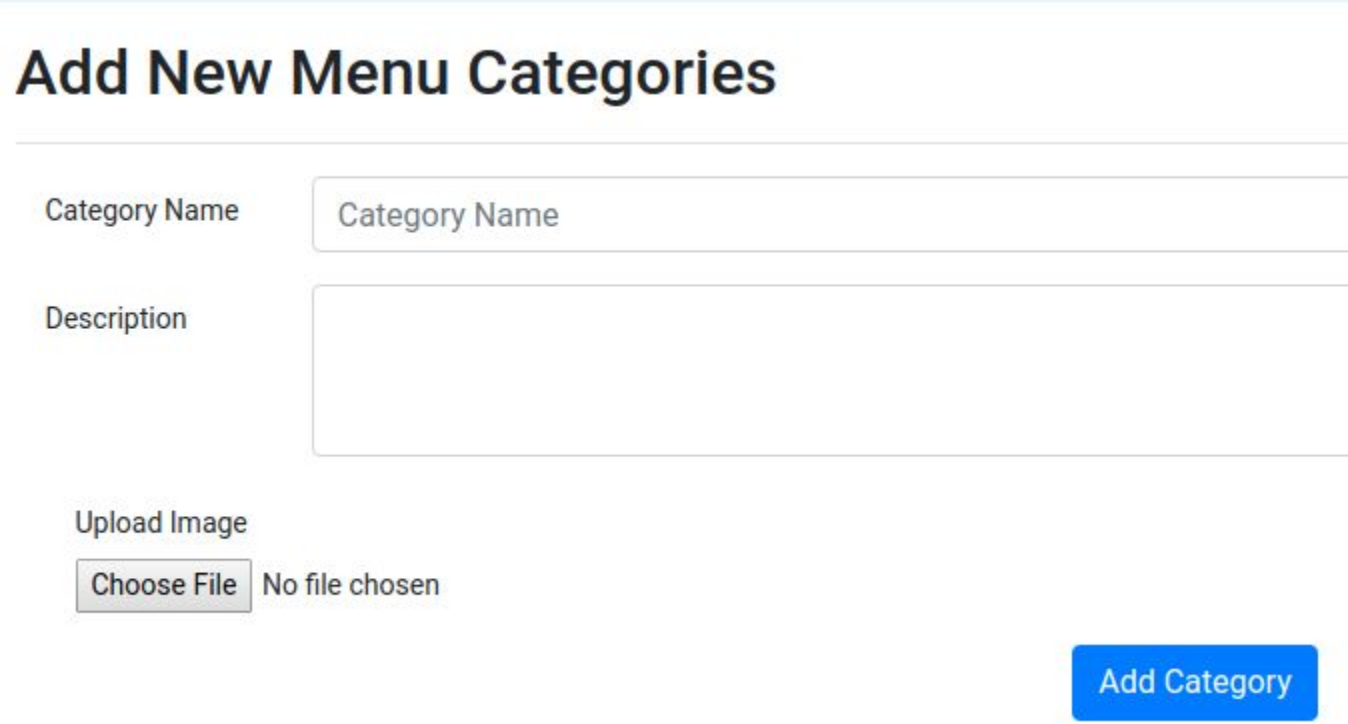
```go
func main() {

    dbconn, err := sql.Open("postgres",
    "postgres://app_admin:P@$$w0rdD2@localhost/restaurantdb")

    if err != nil {
        panic(err)
    }
    defer dbconn.Close()

    catRepo := repository.NewPsqlCategoryRepository(dbconn)
    categoryService = service.NewCategoryService(catRepo)

    http.HandleFunc("/admin/categories/delete", adminCategoriesDelete)
}
```

Usage: adminCategoriesDelete

```go
func adminCategoriesDelete(w http.ResponseWriter, r *http.Request) {

    var categoryService *service.CategoryService

    if r.Method == http.MethodGet {

        idRaw := r.URL.Query().Get("id")
        id, _ := strconv.Atoi(idRaw)

        categoryService.DeleteCategory(id)
    }

    http.Redirect(w, r, "/admin/categories", http.StatusSeeOther)
}
```

Accessing form request values

How to read the values of `Category Name` and `Description`



Accessing form request values

```
<form method="POST" action="/admin/categories/new" enctype="multipart/form-data">

  <div class="form-group row">
    <label for="name" class="col-sm-2 col-form-label">Category Name</label>
    <div class="col-sm-10">
      <input type="text" class="form-control" name="name" >
    </div>
  </div>

  <div class="form-group row">
    <label for="description" class="col-sm-2 col-form-label">Description</label>
    <div class="col-sm-10">
      <textarea class="form-control" name="description" rows="3"></textarea>
    </div>
  </div>
```

Accessing form request values

A handler that process new category post request

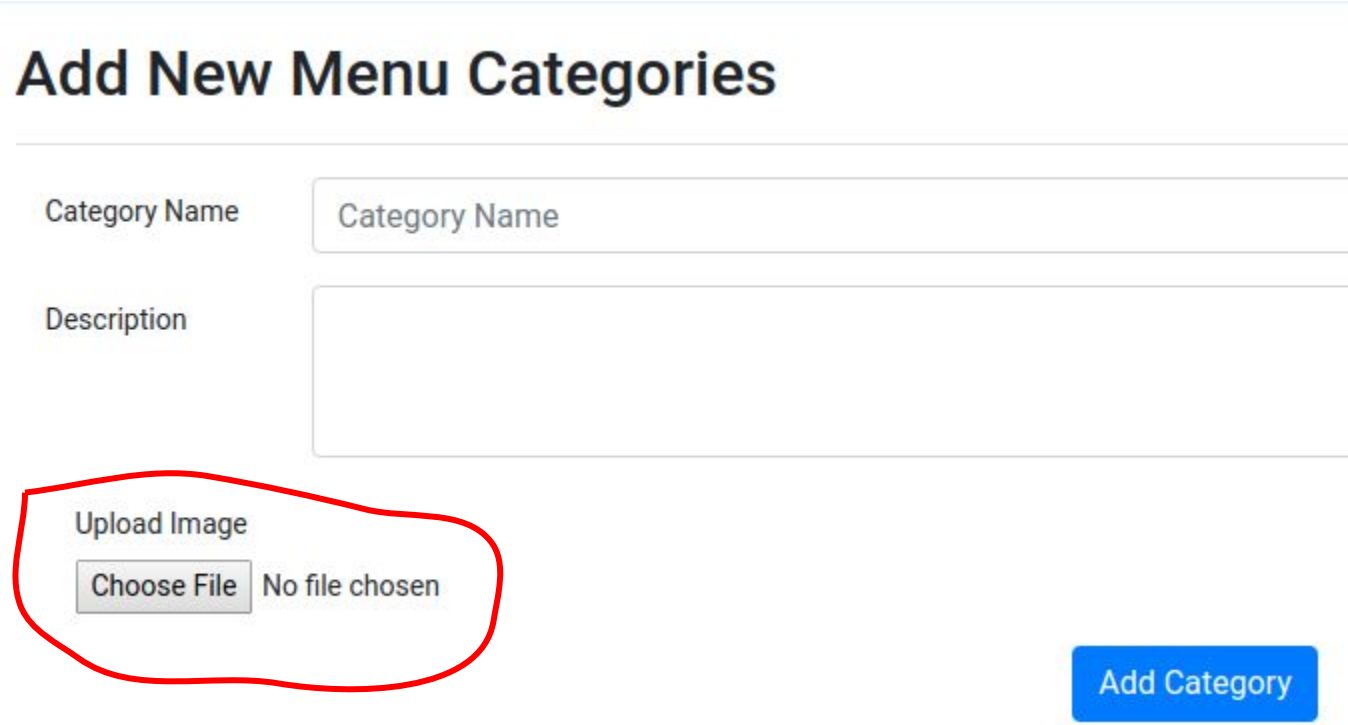
```
func adminCategoriesNew(w http.ResponseWriter, r *http.Request) {

    if r.Method == http.MethodPost {

        ctg := entity.Category{}
        ctg.Name = r.FormValue("name")
        ctg.Description = r.FormValue("description")

        categoryService.StoreCategory(ctg)
    }
        http.Redirect(w, r, "/admin/categories", http.StatusSeeOther)
}
```

Accessing URL values

```
func adminCategoriesDelete(w http.ResponseWriter, r *http.Request) {

    var categoryService *service.CategoryService

    if r.Method == http.MethodGet {

        idRaw := r.URL.Query().Get("id")
        id, _ := strconv.Atoi(idRaw)

        categoryService.DeleteCategory(id)
    }

    http.Redirect(w, r, "/admin/categories", http.StatusSeeOther)
}
```

Uploading pictures to Go web server

Upload file form element



Uploading pictures to Go web server

```
<form method="POST" action="/admin/categories/new" enctype="multipart/form-data">
  <div class="form-group row">…
  </div>
  <div class="form-group row">…
  </div>
  <div class="form-group">
    <label for="catimg" class="col-sm-2 col-form-label">Upload Image</label>
    <div class="col-sm-10">
      <input type="file" class="form-control-file" name="catimg" id="catimg">
    </div>
  </div>
  <div class="form-group row">…
```

Uploading pictures to Go web server

```go
if r.Method == http.MethodPost {
    mf, fh, _ := r.FormFile("catimg")
    defer mf.Close()

    fname := fh.Filename

    wd, err := os.Getwd()
    path := filepath.Join(wd, "delivery", "web", "assets", "img", fname)
    image, err := os.Create(path)
    defer image.Close()

    io.Copy(image, mf)
}
```

Name of the file upload input field

Get the Current Working Directory

Creating a file for storing the uploaded image

Constructing file path

Storing the uploaded file to the newly created file

More Code

You can check more codes of the examples used in the slides at the following url

https://github.com/betsegawlemma/web-prog-go-sample