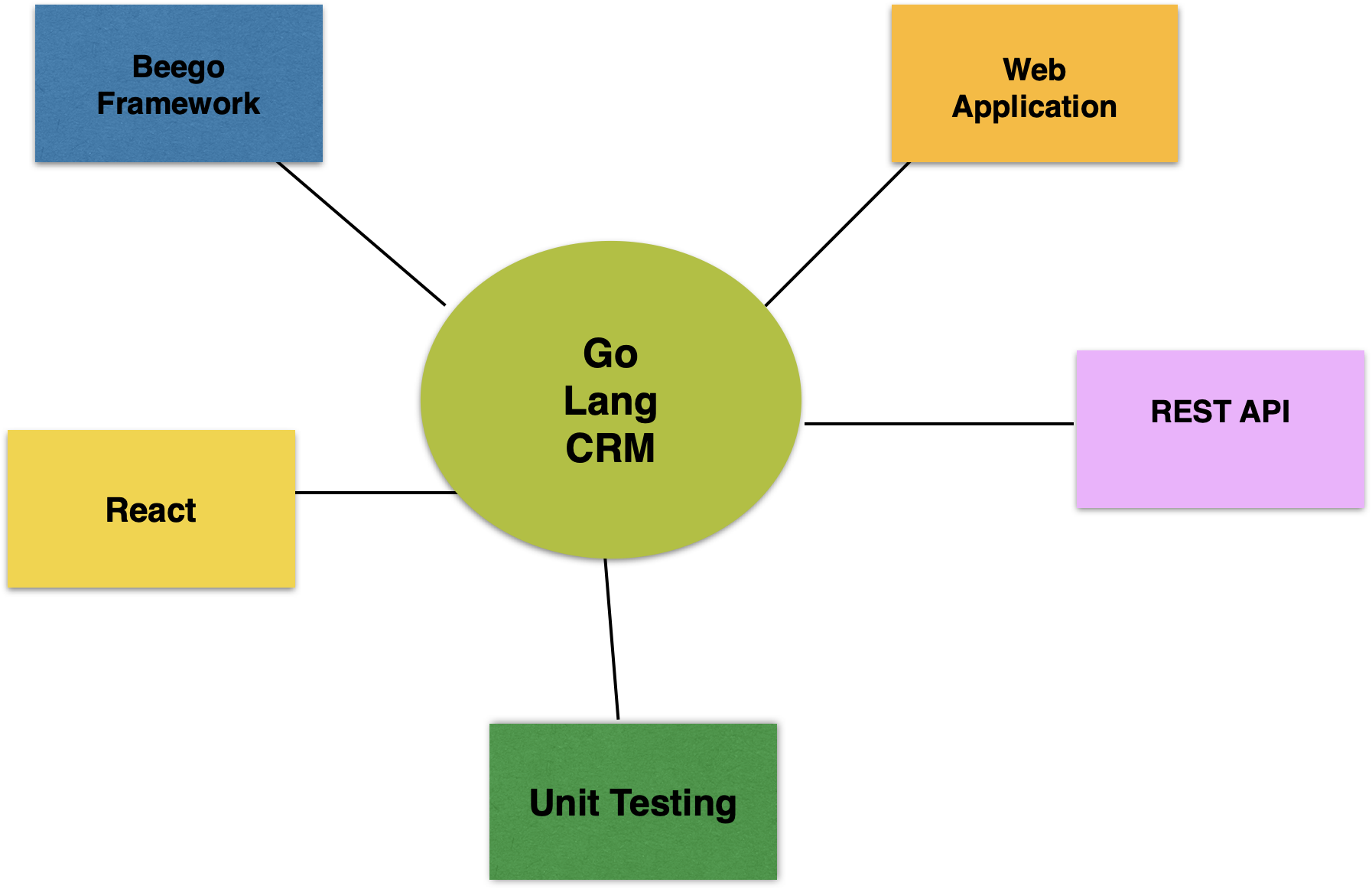
**C H A P T E R 12**

CHAPTER 12

# *Adding Security and IAM*

# Introduction

In this chapter, Reader will understand containers and Docker. Reader will be able to develop services which can be containerized using Docker. Docker Images can be created and posted on the docker Hub for different Go services



## Structure

The chapter covers the following topics:

## Basic Security Services

## Advanced Security Services

## Security Principles

## Securable Web Applications

## Objectives

In this chapter, we are going to build a customer relationship management application. We will be using Go Language. Beego Framework will be used to build the RESTAPI and web application. Readers will be presented with examples of unit testing Rest API in Go Language.

# SOLID Principles

We will be building Customer Relationship Management Application using Go. Customer Relationship Management application consists of creating customers, and managing the customer data. Campaigns will be created and targeted for publishing them to the customers. The leads are gathered as list of customers or segments of customers. Customer segments are created by an attribute like Age, Gender, Salary range, or geography. When the customer likes the product and buys the product, a transaction is logged in the back office of a retail store or an ecommerce company. The transactions are processed to check if the lead has realized and turned into revenue. This process helps in assessing the lead conversion.

The other important feature is the loyalty points which has different levels like gold, silver, and bronze levels. Customer buys the products and loyalty points are stored in the system to track the progress of the customer. Offers are targeted based on the loyalty to improve the purchasing rate of the customer. The discounted offers are shared based on where the customer is in terms of the loyalty points.

Let us start creating the REST API for this sytem.

**SOLID Principles**

We will be using Beego web framework to develop the CRM web application and REST API. Beego framework is an opensource framework which has application tools, Object Relationship Mapping framework, and other packages. ORM framework helps in cutting down the ordeal of developing the CRM relational schema and mapping it to object classes and names of the attributes. Beego is used to build CRM REST Apis which can be integrated with opensource JavaScript frontend frameworks. We can build CRM Beego web application using the Beego and SQLite database. Beego framework helps in developing web apps with affordable cost.

First, let us look at the code for Beego REST API.

**main.go**

package main

import (

"fmt"

"github.com/gorilla/handlers"

"go\_beego\_rest\_api/pkg/db"

handler "go\_beego\_rest\_api/pkg/handlers"

"log"

"net/http"

"github.com/gorilla/mux"

)

func main() {

r := mux.NewRouter()

DB := db.InitializeDB()

r.HandleFunc("/customers", handler.GetCustomers(DB)).Methods("GET")

r.HandleFunc("/create", handler.CreateCustomer(DB)).Methods("POST")

r.HandleFunc("/update", handler.UpdateCustomer(DB)).Methods("PUT")

r.HandleFunc("/delete", handler.DeleteCustomer(DB)).Methods("DELETE")

fmt.Println("Server at 8080")

log.Fatal(http.ListenAndServe(":8080", handlers.CORS()(r)))

}

The screenshot below shows the Beego Web app directory structure.

A screenshot of a computer

Description automatically generated with medium confidence

The above main.go has the code for initialization of ORM and registration of the model. Beego is started in the main method.

**db.go**

package db

import (

"fmt"

"github.com/jinzhu/gorm"

\_ "github.com/jinzhu/gorm/dialects/postgres"

)

func InitializeDB() \*gorm.DB {

db, err := gorm.Open("postgres", "user=newuser password=newuser dbname=crm sslmode=disable")

if err != nil {

fmt.Println(err)

} else {

fmt.Println("DB connected!")

}

return db

}

Handler.go has the code for tableName definition and the object class specification.

**handler.go**

package handler

import (

"encoding/json"

"fmt"

"net/http"

"github.com/jinzhu/gorm"

)

type CustomerBody struct {

Name string `json"name"`

}

type Customer struct {

Id int `json:"id"`

Name string `json:"name"`

Mobile string `json:"mobile"`

Address string `json:"address"`

}

func GetCustomers(db \*gorm.DB) http.HandlerFunc {

return func(w http.ResponseWriter, r \*http.Request) {

w.Header().Set("Content-Type", "application/json")

var customers []Customer

\_ = db.Table("customer").Select("id, name,mobile,address").Scan(&customers)

json.NewEncoder(w).Encode(customers)

}

}

func CreateCustomer(db \*gorm.DB) http.HandlerFunc {

return func(w http.ResponseWriter, r \*http.Request) {

w.Header().Set("Content-Type", "application/json")

var RequestBody CustomerBody

json.NewDecoder(r.Body).Decode(&RequestBody)

\_ = db.Table("customer").Create(&RequestBody)

fmt.Println("Created Customer")

json.NewEncoder(w).Encode(RequestBody)

}

}

func UpdateCustomer(db \*gorm.DB) http.HandlerFunc {

return func(w http.ResponseWriter, r \*http.Request) {

w.Header().Set("Content-Type", "application/json")

var PutBody Customer

json.NewDecoder(r.Body).Decode(&PutBody)

\_ = db.Table("customer").Where("id=?", PutBody.Id).Update("name", PutBody.Name).Scan(&PutBody)

fmt.Printf("Updated Customer with id %d\n", PutBody.Id)

json.NewEncoder(w).Encode(PutBody)

}

}

func DeleteCustomer(db \*gorm.DB) http.HandlerFunc {

return func(w http.ResponseWriter, r \*http.Request) {

w.Header().Set("Content-Type", "application/json")

var DeleteBody Customer

json.NewDecoder(r.Body).Decode(&DeleteBody)

\_ = db.Table("customer").Delete(&DeleteBody)

fmt.Printf("Deleted Customer with id %d\n", DeleteBody.Id)

json.NewEncoder(w).Encode(DeleteBody)

}

}

Now let us look at the database script to create a table in postgres crm database. You can use the below commands to create a postgres database crm and users.

psql -h localhost -d postgres

----

ALTER USER postgres PASSWORD 'postgres';

CREATE USER newuser with PASSWORD 'newuser' CREATEDB;

select \* from users;

\du

\l

\q

-----

createdb crm

psql -h localhost -d postgres

\c crm

CREATE TABLE "customer" (

"id" SERIAL,

"name" varchar(200) NOT NULL,

"mobile" varchar(100),

"address" varchar(400) DEFAULT NULL,

"notes" text,

UNIQUE (name)

);

\dt

\du

alter role newuser superuser;

create user newuser with password 'newuser';

grant all privileges on database crm to newuser;

alter role newuser superuser;

**Database**

CREATE TABLE "customer" (

"id" serial,

"name" varchar(200) NOT NULL,

"mobile" varchar(100),

"address" varchar(400) DEFAULT NULL,

"notes" text,

UNIQUE (name)

);

create user newuser with password 'newuser';

grant all privileges on database crm to newuser;

Now let us look at the routes configured in index.go.

The table below shows the routes/paths mentioned in the index.go.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Action** | **Controller Method** | **Http Method** | **URL Route** | **Description** |
| Create | Add | POST | /create | Create a New Customer |
| Read | getAll | GET | /customers | Retrieve list of customers |
| Delete | Delete | DELETE | /delete | Delete a Customer |
| Update | Update | PUT | /update | Update a Customer |

# Dependency Inversion

In this section, we will look into unit testing of the REST API. You can do the unit testing of the API using post man.

**Create Customer**

A screenshot of a computer

Description automatically generated with medium confidence

**Get Customers**

A screenshot of a computer

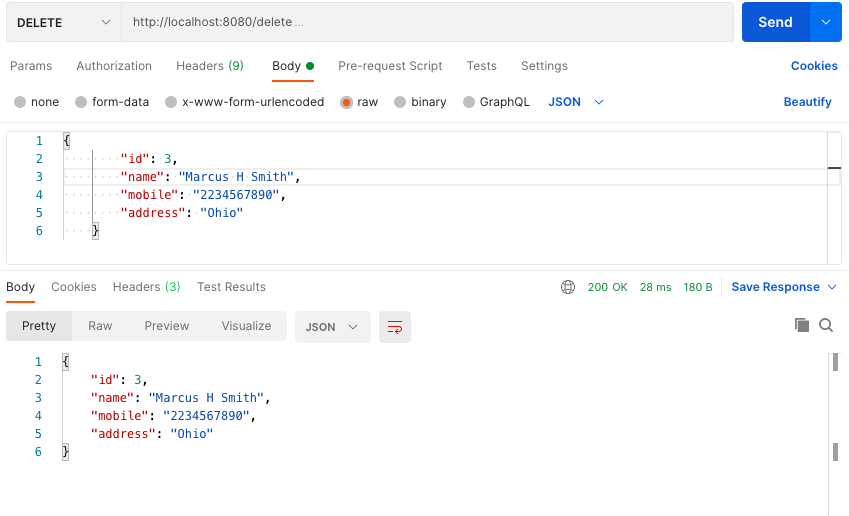
Description automatically generated with medium confidence

**Update Customer**

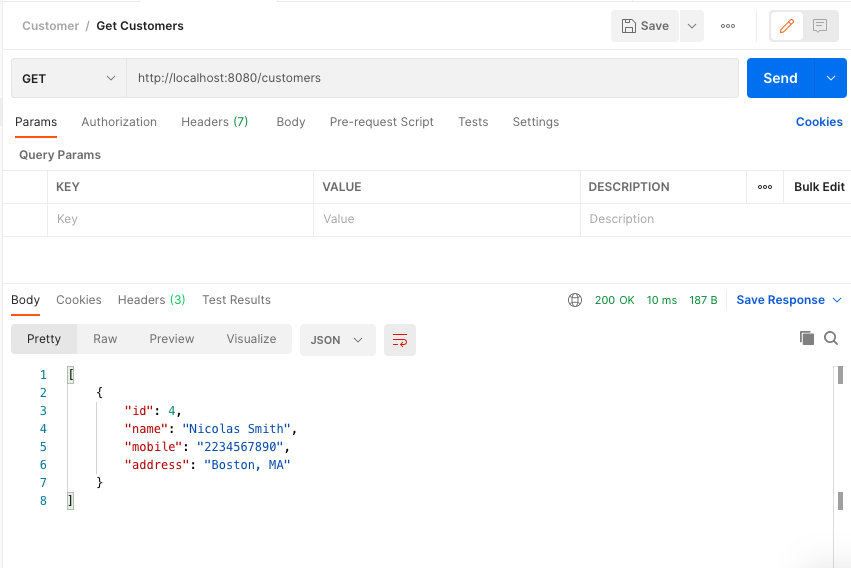
A screenshot of a computer

Description automatically generated with medium confidence

Delete Customer



Get Customers After Delete



# Dependency Injection

Now let us look at the web application and the web forms.

**home.tpl**

<div class="container">

<div class="row">

<div class="hero-text">

<h1>Portal Home</h1>

<p class="lead">This portal is used for managing products</p>

</div>

</div>

</div>

</div>

**view.tpl**

<div class="container">

<div class="row">

<div class="hero-text">

<h1>Customers List</h1>

{{if .flash.error}}

<blockquote>{{.flash.error}}</blockquote>

{{end}}

</div>

</div>

</div>

</div>

<div class="container">

<div class="row">

<table class="table">

<thead>

<tr>

<th>Id</th>

<th>Name</th>

<th>MobileNumber</th>

<th>Address</th>

<th></th>

</tr>

</thead>

<tbody>

{{range $record := .records}}

<tr>

<td>{{$record.Id}}</td>

<td>{{$record.Name}}</td>

<td>{{$record.MobileNumber}}</td>

<td>{{$record.Address}} {{urlfor "ManageController.Delete" ":id" "21"}}</td>

</tr>

{{end}}

</tbody>

<tfoot>

<tr>

<td colspan="4"><a href="{{urlfor "ManageController.Add"}}" title="add new customer">Add another Customer</a></td>

</tr>

</tfoot>

</table>

</div>

</div>

**add.tpl**

<div class="container">

<div class="row">

<div class="hero-text">

<h1>Add A Customer</h1>

</div>

</div>

</div>

</div>

<div class="container">

<div class="row">

<h2>Customer Details</h2>

{{if .flash.error}}

<blockquote>{{.flash.error}}</blockquote>

{{end}}

<p>

<form role="form" id="user" method="POST">

<div class="form-group {{if .Errors.Name}}has-error has-feedback{{end}}">

<label for="name">Customer name： {{if .Errors.Name}}({{.Errors.Name}}){{end}}</label>

<input name="name" type="text" value="{{.Customer.Name}}" class="form-control" tabindex="1" />

{{if .Errors.Name}}<span class="glyphicon glyphicon-remove form-control-feedback"></span>{{end}}

</div>

<div class="form-group">

<label for="mobilenumber">Mobile Number：</label>

<input name="mobilenumber" type="text" value="{{.Customer.MobileNumber}}" class="form-control" tabindex="2" />

</div>

<div class="form-group">

<label for="address">Address：</label>

<input name="address" type="text" value="{{.Customer.Address}}" class="form-control" tabindex="3" />

</div>

<input type="submit" value="Create Customer" class="btn btn-default" tabindex="4" /> &nbsp;

<a href="#" title="don't create the customer">Cancel</a>

</form>

</p>

</div>

</div>

**header.tpl**

<!DOCTYPE html>

<html lang="en">

<head>

<title>CRM</title>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8">

<script src="https://code.jquery.com/jquery-3.4.1.min.js" integrity="sha256-CSXorXvZcTkaix6Yvo6HppcZGetbYMGWSFlBw8HfCJo=" crossorigin="anonymous"></script>

<link rel="stylesheet" href="//netdna.bootstrapcdn.com/bootstrap/3.1.1/css/bootstrap.min.css">

<link rel="stylesheet" href="//netdna.bootstrapcdn.com/bootstrap/3.1.1/css/bootstrap-theme.min.css">

<link rel="stylesheet" href="/static/css/bootstrap.min.css">

<link rel="stylesheet" href="/static/css/bootstrap-theme.min.css">

<link href="/static/css/starter-template.css" rel="stylesheet">

</head>

<body>

<div class="navbar navbar-inverse navbar-fixed-top" role="navigation">

<div class="container">

<div class="navbar-header">

<button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">

<span class="sr-only">Toggle navigation</span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

</button>

<a class="navbar-brand" href="#">CRM</a>

</div>

<div class="collapse navbar-collapse">

<ul class="nav navbar-nav">

<li><a href="/">Home</a></li>

<li class="dropdown">

<a href="#" class="dropdown-toggle" data-toggle="dropdown">Manage Customers <b class="caret"></b></a>

<ul class="dropdown-menu">

<li><a href="/manage/add">Add Customer</a></li>

<li><a href="/manage/view">View All Customers</a></li>

</ul>

</li>

</ul>

</div>

</div>

</div>

**footer.tpl**

<div id="footer">

<div class="container">

<p class="text-muted">

Copyright &copy; 2023

</p>

</div>

</div>

<script src="//netdna.bootstrapcdn.com/bootstrap/3.1.1/js/bootstrap.min.js"></script>

</body>

</html>

You can now compile and run the main.go. The command to create a module go\_web\_app\_beego is shown as below:

go mod init go\_web\_app\_beego

go mod tidy

Then the module is built using command

go build

The output will be as shown below:

(base) apples-MacBook-Air:database bhagvan.kommadi$ go mod init go\_web\_app\_beego

go: creating new go.mod: module go\_web\_app\_beego

go: to add module requirements and sums:

go mod tidy

(base) apples-MacBook-Air:database bhagvan.kommadi$ go mod tidy

go: finding module for package github.com/mattn/go-sqlite3

go: found github.com/mattn/go-sqlite3 in github.com/mattn/go-sqlite3 v1.14.16

(base) apples-MacBook-Air:database bhagvan.kommadi$ go build

(base) apples-MacBook-Air:database bhagvan.kommadi$ ./ go\_web\_app\_beego

3.39.4

(base) apples-MacBook-Air:go\_web\_app\_beego bhagvan.kommadi$ ./go\_web\_app\_beego

2023/05/28 16:09:40.774 [I] [asm\_amd64.s:1571] http server Running on http://:8080

2023/05/28 16:09:46.627 [D] [manage.go:138] Customer name supplied: George Smith

2023/05/28 16:09:46.628 [D] [manage.go:140] Err: <QuerySeter> no row found

2023/05/28 16:09:46.628 [D] [manage.go:144] No customer found matching details supplied. Attempting to insert customer: {0 George Smith 7342129876 212 Oak wood Blvd, Fremont}

2023/05/28 16:09:46.632 [D] [manage.go:148] Customer inserted with id:%!(EXTRA int64=1)

2023/05/28 16:09:46.637 [D] [server.go:2916] | ::1| 200 | 11.83867ms| match| POST /manage/add r:/manage/add

2023/05/28 16:09:46.690 [D] [server.go:2916] | ::1| 404 | 1.354149ms| nomatch| GET /static/css/bootstrap.min.css

2023/05/28 16:09:46.698 [D] [server.go:2916] | ::1| 404 | 1.349423ms| nomatch| GET /static/css/starter-template.css

2023/05/28 16:09:46.698 [D] [server.go:2916] | ::1| 404 | 1.767545ms| nomatch| GET /static/css/bootstrap-theme.min.css

2023/05/28 16:09:51.087 [D] [server.go:2916] | ::1| 200 | 12.825254ms| match| GET /manage/view r:/manage/view

2023/05/28 16:09:51.135 [D] [server.go:2916] | ::1| 404 | 306.119µs| nomatch| GET /static/css/starter-template.css

2023/05/28 16:09:51.136 [D] [server.go:2916] | ::1| 404 | 215.149µs| nomatch| GET /static/css/bootstrap-theme.min.css

2023/05/28 16:09:51.137 [D] [server.go:2916] | ::1| 404 | 233.602µs| nomatch| GET /static/css/bootstrap.min.css

2023/05/28 16:09:59.884 [D] [server.go:2916] | ::1| 200 | 1.708895ms| match| GET / r:/

2023/05/28 16:10:13.116 [D] [server.go:2916] | ::1| 200 | 8.419187ms| match| GET /manage/add r:/manage/add

2023/05/28 16:10:13.152 [D] [server.go:2916] | ::1| 404 | 1.825546ms| nomatch| GET /static/css/bootstrap.min.css

2023/05/28 16:10:13.152 [D] [server.go:2916] | ::1| 404 | 172.6µs| nomatch| GET /static/css/bootstrap-theme.min.css

2023/05/28 16:10:13.153 [D] [server.go:2916] | ::1| 404 | 684.116µs| nomatch| GET /static/css/starter-template.css

2023/05/28 16:18:10.676 [D] [manage.go:138] Customer name supplied: Jessica Murphy

2023/05/28 16:18:10.702 [D] [manage.go:140] Err: <QuerySeter> no row found

2023/05/28 16:18:10.702 [D] [manage.go:144] No customer found matching details supplied. Attempting to insert customer: {0 Jessica Murphy 21958 78675 East Daniel, DE 48106}

2023/05/28 16:18:10.709 [D] [manage.go:148] Customer inserted with id:%!(EXTRA int64=2)

2023/05/28 16:18:10.730 [D] [server.go:2916] | ::1| 200 | 84.316965ms| match| POST /manage/add r:/manage/add

2023/05/28 16:18:10.868 [D] [server.go:2916] | ::1| 404 | 414.859µs| nomatch| GET /static/css/bootstrap.min.css

2023/05/28 16:18:10.884 [D] [server.go:2916] | ::1| 404 | 4.538847ms| nomatch| GET /static/css/bootstrap-theme.min.css

2023/05/28 16:18:10.886 [D] [server.go:2916] | ::1| 404 | 5.877629ms| nomatch| GET /static/css/starter-template.css

2023/05/28 16:18:14.185 [D] [server.go:2916] | ::1| 200 | 5.016121ms| match| GET /manage/view r:/manage/view

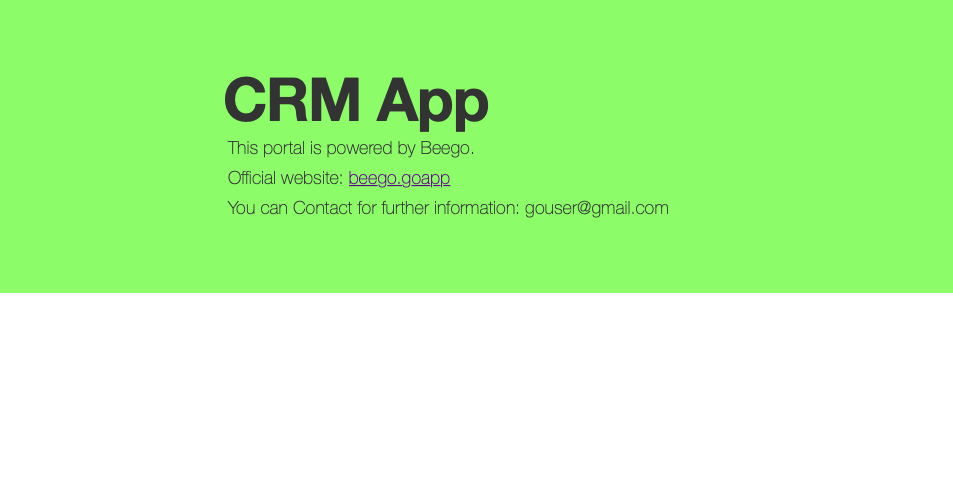
2023/05/28 16:18:14.267 [D] [server.go:2916] | ::1| 404 | 1.840564ms| nomatch| GET /static/css/bootstrap.min.css

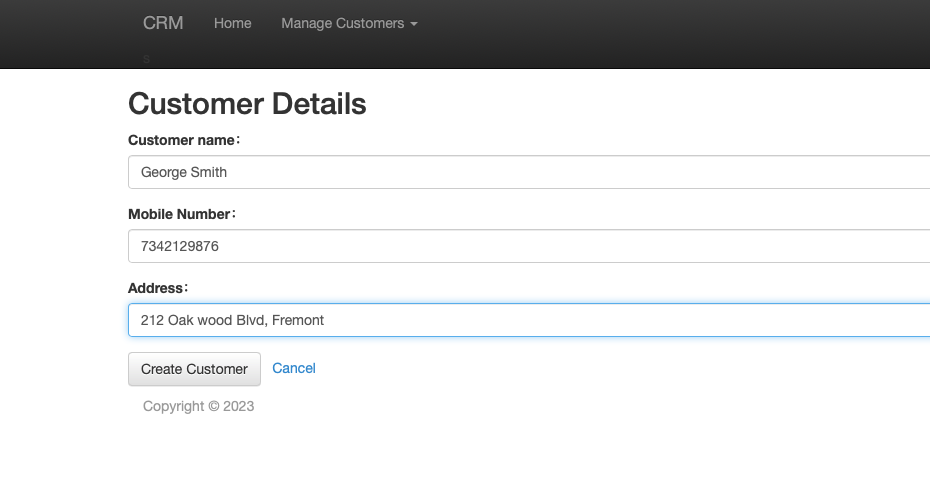
2023/05/28 16:18:14.267 [D] [server.go:2916] | ::1| 404 | 4.512847ms| nomatch| GET /static/css/bootstrap-theme.min.css

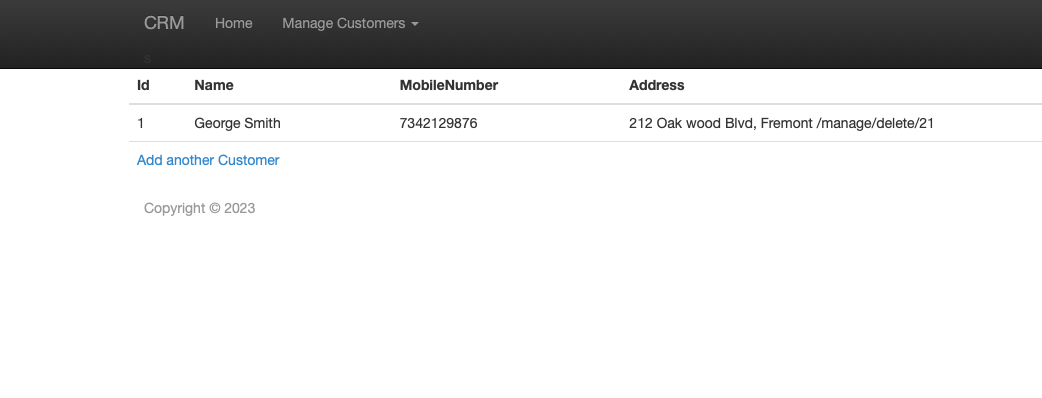
2023/05/28 16:18:14.268 [D] [server.go:2916] | ::1| 404 | 537.423µs| nomatch| GET /static/css/starter-template.css

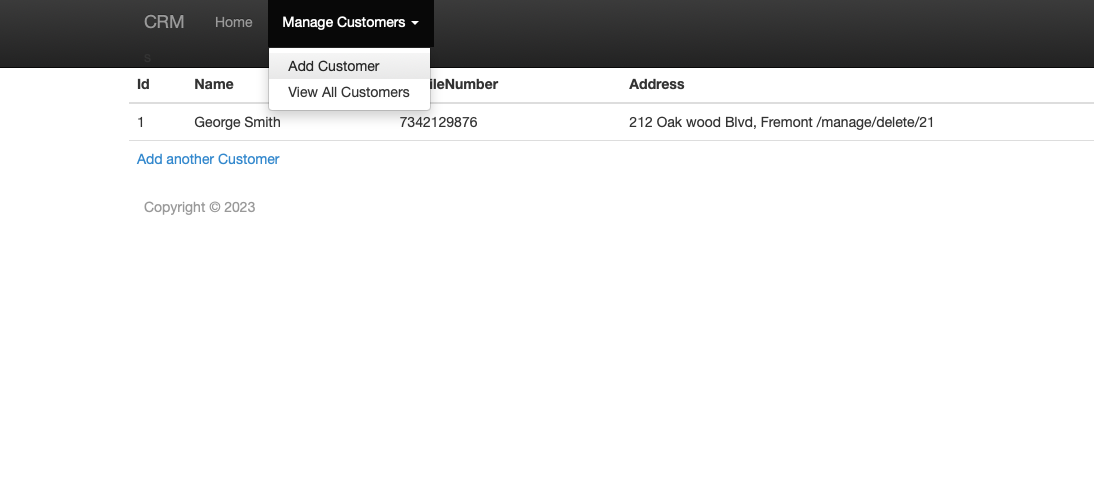
You can access the beego web app at : http://localhost:8080/

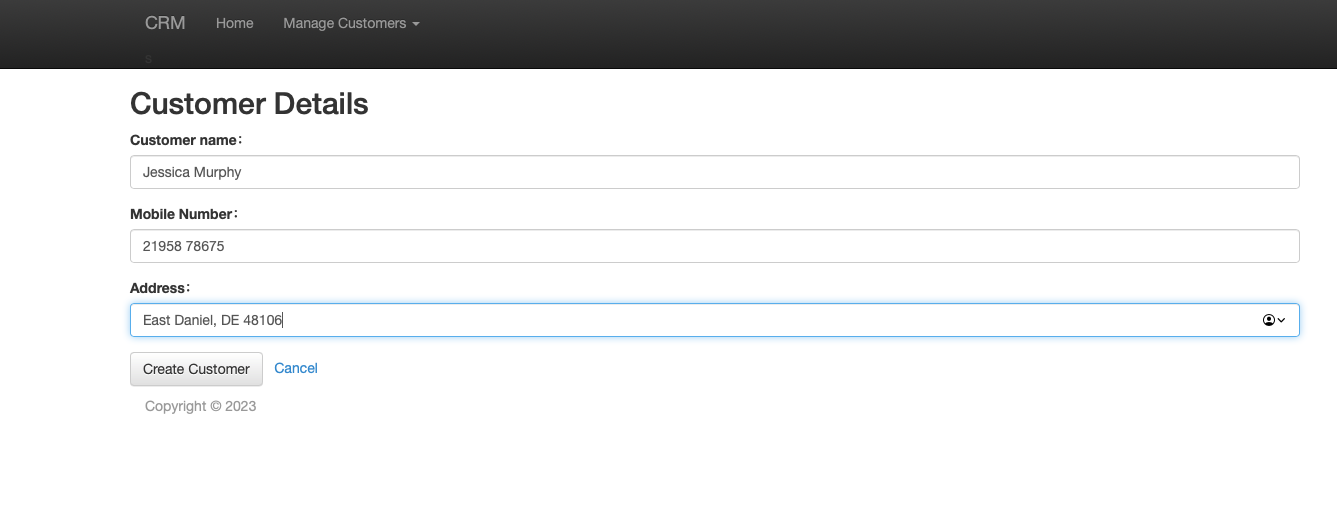
The screenshots below the web application.











A screenshot of a phone number

Description automatically generated with medium confidence

# Dependency Injection Types

# Conclusion

In this chapter we have covered topics related to Go Web Applications using Web Forms, Gin, and Beego. Examples were presented to demonstrate where web applications can be built using REST API using Vue.js, WebForms, and Beego Framework. Summary of the chapter is mentioned below.

* Web applications can be built using Go Lang, Web forms, and database specific packages in Go.
* REST Api can be developed using Gin. ORM based backend can be developed using GORM. Web applications can be built by having Vue.js views talking to REST Api using Gin.
* Beego Framework has capabilities to have REST APIs, ORM, and developing web application framework.