**Hospital Management System**

**Code Inspection Report**

**Team: Ignitors**

**Table of Contents**

1. Modules Inspected
   1. Front End Code
   2. Web API Code
   3. SQL Scripts

Appendix A. Coding Conventions for Angular

Appendix B – Coding Conventions for C#

**Front End Code - Angular**

* File Name - **base-http.service.ts**
* Class Name - **BaseHttpService**

constructor(private httpClient: HttpClient, protected apiConfig: ConfigDataProvider, ) { }

baseUrl: string = 'http://192.168.254.35:8087/'

  /\*\*

   \* Function used for HTTP get

   \* @param relativeUrl API endpoint

   \* @param options Optional parameters

   \*/

  public get<T, OT = any>(relativeUrl: string, options?: OT) {

    if (options) {

      return this.httpClient.get<T>(`${this.baseUrl}${relativeUrl}`, options );

    } else {

      return this.httpClient.get<T>(`${this.baseUrl}${relativeUrl}` );

    }

  }

  /\*\*

   \* Function used for HTTP get post

   \* @param relativeUrl API endpoint

   \* @param data Data object

   \* @param options Optional parameters

   \*/

  public post<T, OT = any>(relativeUrl: string, data: T, options?: OT) {

    if (options ) {

      return this.httpClient.post<T>(`${this.baseUrl}${relativeUrl}`, data, options);

    } else {

      return this.httpClient.post<T>(`${this.baseUrl}${relativeUrl}`, data);

    }

  }

  /\*\*

   \* Function used for HTTP put

   \* @param relativeUrl API endpoint

   \* @param data Data object

   \* @param options Optional parameters

   \*/

  public put<T, OT = any>(relativeUrl: string, data?: T, options?: OT) {

    if (options ) {

      return this.httpClient.put<T>(`${this.baseUrl}${relativeUrl}`, data, options);

    } else {

      return this.httpClient.put<T>(`${this.baseUrl}${relativeUrl}`, data);

    }

  }

  /\*\*

   \* Function used for HTTP delete

   \* @param relativeUrl: API endpoint

   \* @param options: Optional parameters

   \*/

  public delete<T, OT = any>(relativeUrl: string, options?: OT) {

    if (options) {

      return this.httpClient.delete<T>(`${this.baseUrl}${relativeUrl}`, options );

    } else {

      return this.httpClient.delete<T>(`${this.baseUrl}${relativeUrl}` );

    }

  }

* File Name **login.component.ts**
* Component Name – **LoginComponent**
* Screen – **Login Screen**

  /\*\*

  \* Method to called login user api

  \* @param requestData

  \*/

  callLoginUserApi(requestData: any) {

    this.isDataLoading = true;

    this.authService.loginExistingUser(requestData)

      .pipe(takeUntil(this.onDestroy$))

      .subscribe({

        next: (retData: any) => {

          if (retData.status) {

            this.parseResponse(retData,requestData);

          } else {

            this.toastService.errorMessage(retData.message);

          }

          this.isDataLoading = false;

        },

        error: (err: any) => {

          console.log(err);

          this.isDataLoading = false;

        },

        complete: () => {

          console.log('complete');

          this.isDataLoading = false;

        }

      });

  }

* File Name registration**.component.ts**
* Component Name – **RegistrationComponent**
* Screen –**Register Screen**

 /\*\*

   \* Method to called register user api

   \* @param respData

   \*/

  callRegisterUserApi(respData: any) {

    this.isDataLoading=true;

    this.authService.registerNewUser(respData)

      .pipe(takeUntil(this.onDestroy$))

      .subscribe({

        next: (retData: any) => {

          this.isDataLoading = false;

          if (retData.status) {

            this.toastService.successMessage(Messages.RegisterUserSuccess);

            this.router.navigate(['login']);

          } else {

            this.toastService.errorMessage(retData.message);

          }

        },

        error: (err: any) => {

          console.log(err);

          this.isDataLoading = false;

        },

        complete: () => {

          console.log('complete');

          this.isDataLoading = false;

        }

      });

  }

}

* **Constants File**

export enum Constants {

  RegexPatternAlphaNumeric = '^[a-zA-Z0-9 ]+$',

  RegexPatternOnlyNumeric = '^[0-9]\*$',

  RegexPatternVarCharDataType = '^[a-zA-Z0-9\_.\/-]\*$',

  RegexPatternNumberDataType = '[0-9]+',

  RegexPatternCodeValue = '^[9, 0-9]\*$',

  RegexPatternExponential = '^[0-9]+([eE][-+]?[0-9]+)?$',

  AccessToken = 'access\_token',

  UnauthorizedAccessEvent = 'unAuthorizedAccess',

  IsAuthenticated = 'authenticated',

  IsUserLoggedIn = 'loggedIn',

  ConnectionTimeoutErrorEvent = 'connectionTimeoutError',

  FormValid = 'VALID',

  FormInvalid = 'INVALID',

  FormDisabled = 'DISABLED',

  LoggedInUserData = 'loggedInUserData',

  SessionStorage = 'sessionStorage',

  Max\_Length\_1 = 1,

  Max\_Length\_2 = 2,

  Max\_Length\_3 = 3,

  Max\_Length\_4 = 4,

  Max\_Length\_5 = 5,

  Max\_Length\_6 = 6,

  Max\_Length\_10 = 10,

  Max\_Length\_11 = 11,

  Max\_Length\_22 = 22,

  Max\_Length\_30 = 30,

  Max\_Length\_50 = 50,

  Max\_Length\_100 = 100,

  Max\_Length\_200 = 200,

  Max\_Length\_255 = 255,

  DefaultDateTimeFormat = 'MM/DD/YYYY HH:mm:ss A',

  // --idle/keep-alive - constants -- //

  IdleTimeout = 1200, // 20 mins

  ForceTimeout = 120, // 2min

  DateFormat = 'MM/dd/yyyy',

  TimeFormat = 'hh:mm a',

  DefaultToastTimeout=5000,

}

export const ModuleConstants = {

  timeDelay: 500,

  exportTime: 5000,

  apiTimeout: 120000,  // 2 min

  userRoles: [{ key: 'Admin', value: 'Admin' }, { key: 'Doctor', value: 'Doctor' }, { key: 'Patient', value: 'Patient' }, { key: 'Staff', value: 'Staff' }],

};

* **Messages File**

export enum Messages {

    Dialog\_Message\_Error = 'Unable to process the request. Please try again after sometime',

    Dialog\_Authentication\_Error = 'Unable to process the request. Please try again after sometime. Do you want to logout?',

    Dialog\_Timeout\_Message\_Error = 'Network timeout. Please try again after sometime',

    Mandatory\_Fields\_Validation = 'Enter Mandatory Details',

    Dialog\_Title = 'Message',

    Dialog\_Button\_Ok = 'Ok',

    Dialog\_Button\_Close = 'Close',

    Dialog\_Button\_Cancel = 'Cancel',

    Dialog\_Confirmation\_Title = 'Confirmation',

    Dialog\_Confirmation\_Delete\_Message = 'Are you sure you want to delete?',

    Dialog\_Confirmation\_Button\_Title\_First = 'Yes',

    Dialog\_Confirmation\_Button\_Title\_Second = 'No',

    Unsaved\_Changes\_Alert = 'Unsaved changes present. Please save or discard them to continue',

    Unsaved\_Changes\_Confirmation = '<div>You have unsaved changes that will be lost.</div><div>Do you still want to continue?</div>',

    Only\_Number\_Validation = 'Enter only number in this field',

    Numeric\_Values\_Allowed = 'Only positive numbers are allowed',

    Email\_Validation = 'Enter proper email format',

    Max\_Length\_Validation = 'Maximum characters allowed is ',

    Min\_Length\_Validation = 'Minimum characters allowed is ',

    Max\_Length\_Validation\_int = 'Maximum digits allowed is ',

    Required\_Validation = 'This is a required field',

    Invalid\_Date\_Entered = 'Invalid date entered',

    UnAuthorizedAccess = 'You are not authorized to use this application',

    Session\_Timeout\_Warning = 'Your session will expire in 2 minutes. Would you like to continue your session?',

    SignOutButton = 'Sign Out',

    Invalid\_Browser = 'This site is accessible in Chrome browser only',

    Dialog\_Continue\_Button = 'Continue',

    Dialog\_Cancel\_Button = 'Cancel',

    Logout\_Dialog\_Button\_Cancel = 'Cancel',

    Logout\_Confirmation = 'Are you sure you want to logout?',

    Login\_Failure\_Message = 'Unable to login. Please try again after sometime',

    RegisterUserSuccess='User Registered Successfully',

    Password\_Validate\_Message='New Password and Confirm New Password should be same',

}

* **App Routing Module**
* **app-routing.module.ts**

import { NgModule } from '@angular/core';

import { PreloadAllModules, RouterModule, Routes } from '@angular/router';

const routes: Routes = [

  {

    path: '',

    loadChildren: () => import('./modules/authentication/authentication.module').then(m => m.AuthenticationModule)

  },

  {

    path: 'authentication',

    loadChildren: () => import('./modules/authentication/authentication.module').then(m => m.AuthenticationModule)

  },

  {

    path: 'home',

    loadChildren: () => import('./modules/home/home.module').then(m => m.HomeModule)

  },

];

@NgModule({

  imports: [RouterModule.forRoot(routes)],

  exports: [RouterModule]

})

export class AppRoutingModule { }

* **Toast Message Service**
* **toast-message.service.ts**

import { Injectable } from '@angular/core';

import { ToastrService } from 'ngx-toastr';

import { Constants } from '../../constants/constants';

@Injectable({

    providedIn: 'root'

})

export class ToastMessageService {

    toastStyleClass='toast-custom';

    private toastr: ToastrService;

    constructor(private toasterService1: ToastrService) {

        this.toastr = toasterService1;

    }

    /\*\*

     \* Method to show success toast message

     \* @param message: custome message

     \* @param timeOutVal:custome timeoutval

     \*/

    successMessage(message: string, timeOutVal?: number) {

        const title = '';

        let defaultTimeout = Constants.DefaultToastTimeout;

        if (timeOutVal) {

            defaultTimeout = timeOutVal;

        }

        this.toastr.success(message, title, {

            timeOut: defaultTimeout,

            closeButton:true,

            positionClass:this.toastStyleClass

        });

    }

    /\*\*

     \* Method to show error toast message

     \* @param message: custome message

     \* @param timeOutVal:custome timeoutval

     \*/

    errorMessage(message: string, timeOutVal?: number) {

        const title = '';

        let defaultTimeout = Constants.DefaultToastTimeout;

        if (timeOutVal) {

            defaultTimeout = timeOutVal;

        }

        this.toastr.error(message, title, {

            timeOut: defaultTimeout,

            positionClass:this.toastStyleClass

        });

    }

    /\*\*

     \* Method to show warning toast message

     \* @param message: custome message

     \* @param timeOutVal:custome timeoutval

     \*/

    warningMessage(message: string, timeOutVal?: number) {

        const title = '';

        let defaultTimeout = Constants.DefaultToastTimeout;

        if (timeOutVal) {

            defaultTimeout = timeOutVal;

        }

        this.toastr.warning(message, title, {

            timeOut: defaultTimeout,

            closeButton:true,

            positionClass:this.toastStyleClass

        });

    }

    /\*\*

     \* Method to show info toast message

     \* @param message: custome message

     \* @param timeOutVal:custome timeoutval

     \*/

    infoMessage(message: string, timeOutVal?: number) {

        const title = '';

        let defaultTimeout = Constants.DefaultToastTimeout;

        if (timeOutVal) {

            defaultTimeout = timeOutVal;

        }

        this.toastr.info(message, title, {

            timeOut: defaultTimeout,

            closeButton:true,

            positionClass:this.toastStyleClass

        });

    }

}

**ASP .NET ( WEB API CODE)**

**HospitalManagementController.ts**

using Newtonsoft.Json;

using Newtonsoft.Json.Linq;

using System;

using System.Collections;

using System.Collections.Generic;

using System.Data;

using System.Linq;

using System.Net;

using System.Net.Http;

using System.Threading.Tasks;

using System.Web.Http;

using WebApplication4.UtilityClasses;

using System.Web.Http.Cors;

namespace WebApplication4.Controllers

{

    [EnableCors(origins: "\*", headers: "\*", methods: "\*")]

    public class HospitalManagementController : ApiController

    {

        static string nodatafound = @"{""Message"": ""No data found""}";

        static string noaction = @"{""Message"": ""No action performed, please try again""}";

        #region API Health

        [HttpGet]

        [Route("api/Health")]

        public async Task<string> Get()

        {

            await Task.Delay(1);

            return "Service Running! Healthy!";

        }

        private void BadRequestValidation<T>(T RequestObject) where T : class

        {

            var modelStateJson = ModelState.SelectMany(x => x.Value.Errors.Select(e => (string.IsNullOrEmpty(e.ErrorMessage) ? e.Exception.Message : e.ErrorMessage))).Aggregate((a, b) => a + b);

            var message = "Model Validation faiiled, Required fields are not provided. More details : " + modelStateJson;

            int id = 0;

            if (typeof(T) == typeof(Doctor)) { Doctor newT1 = (Doctor)(object)RequestObject; id = newT1.DoctorId; }

            if (typeof(T) == typeof(Nurse)) { Nurse newT1 = (Nurse)(object)RequestObject; id = newT1.NurseId; }

            if (typeof(T) == typeof(HospitalBranch)) { HospitalBranch newT1 = (HospitalBranch)(object)RequestObject; id = newT1.HospitalId; }

            if (typeof(T) == typeof(Patient)) { Patient newT1 = (Patient)(object)RequestObject; id = newT1.PatientId; }

            if (typeof(T) == typeof(Medicine)) { Medicine newT1 = (Medicine)(object)RequestObject; id = newT1.MedicineId; }

            if (typeof(T) == typeof(Service)) { Service newT1 = (Service)(object)RequestObject; id = newT1.ServiceRoomId; }

            if (typeof(T) == typeof(OperativeRoom)) { OperativeRoom newT1 = (OperativeRoom)(object)RequestObject; id = newT1.OperativeRoomId; }

            if (typeof(T) == typeof(Medication)) { Medication newT1 = (Medication)(object)RequestObject; id = newT1.MedicationId; }

            if (typeof(T) == typeof(TreatmentNurse)) { TreatmentNurse newT1 = (TreatmentNurse)(object)RequestObject; id = newT1.TreatmentNurseId; }

            if (typeof(T) == typeof(Treatment)) { Treatment newT1 = (Treatment)(object)RequestObject; id = newT1.TreatmentId; }

            if (typeof(T) == typeof(Admission)) { Admission newT1 = (Admission)(object)RequestObject; id = newT1.AdmissionId; }

            if (typeof(T) == typeof(Payment)) { Payment newT1 = (Payment)(object)RequestObject; id = newT1.PaymentId; }

            var response = new HttpResponseMessage(HttpStatusCode.BadRequest);

            var responseModel = new Respone { ID = id, Message = message };

            response.Content = new StringContent(JsonConvert.SerializeObject(responseModel), System.Text.Encoding.UTF8, "application/json");

            throw new HttpResponseException(response);

        }

        private HttpResponseMessage PrepareDMLOutput(ResponseMessage action)

        {

            string Json = "";

            Json = JsonConvert.SerializeObject(action);

            return Request.CreateResponse(HttpStatusCode.OK, Json);

        }

        #endregion

        #region Login

        [HttpPost]

        [Route("api/Login/GetLoginAccess")]

        public async Task<HttpResponseMessage> GetLoginAccess(Users users)

        {

            ResponseMessage objres = new ResponseMessage();

            try

            {

                await Task.Yield();

                string Json = "";

                var response = new HttpResponseMessage();

                string command = "select UserRole from Users where UserName='" + users.Email + "' and Password='" + users.Password + "'";

                objres = Dal.GetDataforcommand(command);

                Json = JsonConvert.SerializeObject(objres);

                return Request.CreateResponse(HttpStatusCode.OK, Json);

            }

            catch (Exception ex)

            {

                return Request.CreateResponse(HttpStatusCode.ExpectationFailed, ex.InnerException.Message);

            }

        }

        [HttpPost]

        [Route("api/Login/CreateLogin")]

        public async Task<HttpResponseMessage> CreateLogin(Users users)

        {

            await Task.Yield();

            ResponseMessage objres = new ResponseMessage();

            string Json = "";

            objres = Dal.InsertRecord(users);

            Json = JsonConvert.SerializeObject(objres);

            return Request.CreateResponse(HttpStatusCode.OK, Json);

        }

        #endregion

           }

}

**SQL SCRIPTS**

-- MySQL Workbench Forward Engineering

-- create database hospitalmgmt;

use hospitalmgmt;

SET @OLD\_UNIQUE\_CHECKS=@@UNIQUE\_CHECKS, UNIQUE\_CHECKS=0;

SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0;

SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='ONLY\_FULL\_GROUP\_BY,STRICT\_TRANS\_TABLES,NO\_ZERO\_IN\_DATE,NO\_ZERO\_DATE,ERROR\_FOR\_DIVISION\_BY\_ZERO,NO\_ENGINE\_SUBSTITUTION';

-- -----------------------------------------------------

-- Schema mydb

-- -----------------------------------------------------

-- -----------------------------------------------------

-- Schema hospitalmgmt

-- -----------------------------------------------------

-- -----------------------------------------------------

-- Schema hospitalmgmt

-- -----------------------------------------------------

CREATE SCHEMA IF NOT EXISTS `hospitalmgmt` DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci ;

USE `hospitalmgmt` ;

-- -----------------------------------------------------

-- Table `hospitalmgmt`.`paymentmodes`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `hospitalmgmt`.`paymentmodes` (

  `PaymentModeId` INT NOT NULL AUTO\_INCREMENT,

  `Name` VARCHAR(50) NOT NULL,

  `ApplicableMethods` VARCHAR(100) NULL DEFAULT NULL,

  PRIMARY KEY (`PaymentModeId`))

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

-- -----------------------------------------------------

-- Table `hospitalmgmt`.`payments`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `hospitalmgmt`.`payments` (

  `PaymentId` INT NOT NULL AUTO\_INCREMENT,

  `PaymentCode` VARCHAR(20) NOT NULL,

  `PaymentmodeId` INT NOT NULL,

  `PayeeName` VARCHAR(50) NOT NULL,

  `TotalAmount` DECIMAL(12,2) NULL DEFAULT NULL,

  `DiscountAmount` DECIMAL(12,2) NULL DEFAULT NULL,

  `InsuranceAmount` DECIMAL(12,2) NULL DEFAULT NULL,

  `PayableAmount` DECIMAL(12,2) NULL DEFAULT NULL,

  PRIMARY KEY (`PaymentId`),

  UNIQUE INDEX `PaymentCode\_UNIQUE` (`PaymentCode` ASC) VISIBLE,

  INDEX `FK\_Payments\_ModeId` (`PaymentmodeId` ASC) VISIBLE,

  CONSTRAINT `FK\_Payments\_ModeId`

    FOREIGN KEY (`PaymentmodeId`)

    REFERENCES `hospitalmgmt`.`paymentmodes` (`PaymentModeId`)

    ON DELETE CASCADE

    ON UPDATE CASCADE)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

-- -----------------------------------------------------

-- Table `hospitalmgmt`.`doctors`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `hospitalmgmt`.`doctors` (

  `DoctorId` INT NOT NULL AUTO\_INCREMENT,

  `Name` VARCHAR(50) NOT NULL,

  `Address1` VARCHAR(100) NULL DEFAULT NULL,

  `Address2` VARCHAR(100) NULL DEFAULT NULL,

  `City` VARCHAR(20) NULL DEFAULT NULL,

  `Phone` VARCHAR(20) NULL DEFAULT NULL,

  `Mobile` VARCHAR(20) NULL DEFAULT NULL,

  `Email` VARCHAR(50) NOT NULL,

  `Salary` DECIMAL(12,2) NULL DEFAULT NULL,

  `Designation` VARCHAR(50) NULL DEFAULT NULL,

  `Gender` VARCHAR(50) NULL DEFAULT NULL,

  PRIMARY KEY (`DoctorId`))

ENGINE = InnoDB

AUTO\_INCREMENT = 25

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

-- -----------------------------------------------------

-- Table `hospitalmgmt`.`hospitals`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `hospitalmgmt`.`hospitals` (

  `HospitalId` INT NOT NULL AUTO\_INCREMENT,

  `HospitalCode` VARCHAR(20) NOT NULL,

  `Name` VARCHAR(50) NOT NULL,

  `Address1` VARCHAR(100) NULL DEFAULT NULL,

  `Address2` VARCHAR(100) NULL DEFAULT NULL,

  `City` VARCHAR(20) NULL DEFAULT NULL,

  `Phone1` VARCHAR(20) NULL DEFAULT NULL,

  `Phone2` VARCHAR(20) NULL DEFAULT NULL,

  `Email` VARCHAR(50) NOT NULL,

  `Description` VARCHAR(200) NULL DEFAULT NULL,

  PRIMARY KEY (`HospitalId`))

ENGINE = InnoDB

AUTO\_INCREMENT = 4

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

-- -----------------------------------------------------

-- Table `hospitalmgmt`.`roomtypes`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `hospitalmgmt`.`roomtypes` (

  `RoomTypeId` INT NOT NULL AUTO\_INCREMENT,

  `Name` VARCHAR(50) NOT NULL,

  `Description` VARCHAR(200) NULL DEFAULT NULL,

  PRIMARY KEY (`RoomTypeId`))

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

-- -----------------------------------------------------

-- Table `hospitalmgmt`.`operativerooms`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `hospitalmgmt`.`operativerooms` (

  `OperativeRoomId` INT NOT NULL AUTO\_INCREMENT,

  `OperativeRoomCode` VARCHAR(20) NOT NULL,

  `RoomTypeId` INT NOT NULL,

  `RoomNo` VARCHAR(10) NOT NULL,

  `BedNo` VARCHAR(10) NULL DEFAULT NULL,

  `StartDate` DATETIME NULL DEFAULT NULL,

  `EndDate` DATETIME NULL DEFAULT NULL,

  `HospitalId` INT NOT NULL,

  PRIMARY KEY (`OperativeRoomId`),

  UNIQUE INDEX `OperativeRoomCode\_UNIQUE` (`OperativeRoomCode` ASC) VISIBLE,

  INDEX `FK\_OpeRooms\_HospitalId` (`HospitalId` ASC) VISIBLE,

  INDEX `FK\_OpeRooms\_RoomTypeId` (`RoomTypeId` ASC) VISIBLE,

  CONSTRAINT `FK\_OpeRooms\_HospitalId`

    FOREIGN KEY (`HospitalId`)

    REFERENCES `hospitalmgmt`.`hospitals` (`HospitalId`)

    ON DELETE CASCADE

    ON UPDATE CASCADE,

  CONSTRAINT `FK\_OpeRooms\_RoomTypeId`

    FOREIGN KEY (`RoomTypeId`)

    REFERENCES `hospitalmgmt`.`roomtypes` (`RoomTypeId`)

    ON DELETE CASCADE

    ON UPDATE CASCADE)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

-- -----------------------------------------------------

-- Table `hospitalmgmt`.`patients`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `hospitalmgmt`.`patients` (

  `PatientId` INT NOT NULL AUTO\_INCREMENT,

  `PatientCode` VARCHAR(20) NOT NULL,

  `Name` VARCHAR(50) NOT NULL,

  `Address1` VARCHAR(100) NULL DEFAULT NULL,

  `Address2` VARCHAR(100) NULL DEFAULT NULL,

  `City` VARCHAR(50) NULL DEFAULT NULL,

  `Phone` VARCHAR(20) NULL DEFAULT NULL,

  `GuardianPhone` VARCHAR(10) NULL DEFAULT NULL,

  `Email` VARCHAR(50) NULL DEFAULT NULL,

  `Disease` VARCHAR(50) NULL DEFAULT NULL,

  `Gender` VARCHAR(50) NULL DEFAULT NULL,

  PRIMARY KEY (`PatientId`),

  UNIQUE INDEX `PatientCode\_UNIQUE` (`PatientCode` ASC) VISIBLE)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

-- -----------------------------------------------------

-- Table `hospitalmgmt`.`treatments`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `hospitalmgmt`.`treatments` (

  `TreatmentId` INT NOT NULL AUTO\_INCREMENT,

  `TreatmentCode` VARCHAR(20) NOT NULL,

  `Name` VARCHAR(50) NOT NULL,

  `Duration` INT NULL DEFAULT NULL,

  `DoctorId` INT NOT NULL,

  `PatientId` INT NOT NULL,

  `OperativeRoomId` INT NOT NULL,

  PRIMARY KEY (`TreatmentId`),

  UNIQUE INDEX `TreatmentCode\_UNIQUE` (`TreatmentCode` ASC) VISIBLE,

  INDEX `FK\_Treatments\_DoctorId` (`DoctorId` ASC) VISIBLE,

  INDEX `FK\_Treatments\_OpsRoomId` (`OperativeRoomId` ASC) VISIBLE,

  INDEX `FK\_Treatments\_PatientId` (`PatientId` ASC) VISIBLE,

  CONSTRAINT `FK\_Treatments\_DoctorId`

    FOREIGN KEY (`DoctorId`)

    REFERENCES `hospitalmgmt`.`doctors` (`DoctorId`)

    ON DELETE CASCADE

    ON UPDATE CASCADE,

  CONSTRAINT `FK\_Treatments\_OpsRoomId`

    FOREIGN KEY (`OperativeRoomId`)

    REFERENCES `hospitalmgmt`.`operativerooms` (`OperativeRoomId`)

    ON DELETE CASCADE

    ON UPDATE CASCADE,

  CONSTRAINT `FK\_Treatments\_PatientId`

    FOREIGN KEY (`PatientId`)

    REFERENCES `hospitalmgmt`.`patients` (`PatientId`)

    ON DELETE CASCADE

    ON UPDATE CASCADE)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

-- -----------------------------------------------------

-- Table `hospitalmgmt`.`admissions`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `hospitalmgmt`.`admissions` (

  `AdmissionId` INT NOT NULL AUTO\_INCREMENT,

  `AdmissionCode` VARCHAR(20) NOT NULL,

  `TreatmentId` INT NOT NULL,

  `AdmittedDate` DATETIME NULL DEFAULT NULL,

  `IsInsuranceApplicable` TINYINT NULL DEFAULT NULL,

  `InsurancePhotoId` VARCHAR(50) NULL DEFAULT NULL,

  `PaymentId` INT NOT NULL,

  `DischargeDate` DATETIME NULL DEFAULT NULL,

  PRIMARY KEY (`AdmissionId`),

  UNIQUE INDEX `AdmissionCode\_UNIQUE` (`AdmissionCode` ASC) VISIBLE,

  INDEX `FK\_Admissions\_PaymentId` (`PaymentId` ASC) VISIBLE,

  INDEX `FK\_Admissions\_TreatmentId` (`TreatmentId` ASC) VISIBLE,

  CONSTRAINT `FK\_Admissions\_PaymentId`

    FOREIGN KEY (`PaymentId`)

    REFERENCES `hospitalmgmt`.`payments` (`PaymentId`)

    ON DELETE CASCADE

    ON UPDATE CASCADE,

  CONSTRAINT `FK\_Admissions\_TreatmentId`

    FOREIGN KEY (`TreatmentId`)

    REFERENCES `hospitalmgmt`.`treatments` (`TreatmentId`)

    ON DELETE CASCADE

    ON UPDATE CASCADE)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

-- -----------------------------------------------------

-- Table `hospitalmgmt`.`appointments`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `hospitalmgmt`.`appointments` (

  `AppointmentId` INT NOT NULL AUTO\_INCREMENT,

  `AppointmentCode` VARCHAR(20) NOT NULL,

  `DoctorId` INT NOT NULL,

  `PatientId` INT NOT NULL,

  `HospitalId` INT NOT NULL,

  `Appt\_Date` DATETIME NULL DEFAULT NULL,

  `Next\_Appt\_Date` DATETIME NULL DEFAULT NULL,

  `DiseaseNotes` VARCHAR(50) NULL DEFAULT NULL,

  `Fee` DECIMAL(12,2) NULL DEFAULT NULL,

  PRIMARY KEY (`AppointmentId`),

  UNIQUE INDEX `AppointmentCode\_UNIQUE` (`AppointmentCode` ASC) VISIBLE,

  INDEX `FK\_Appointment\_DoctorId` (`DoctorId` ASC) VISIBLE,

  INDEX `FK\_Appointment\_HospitalId` (`HospitalId` ASC) VISIBLE,

  INDEX `FK\_Appointment\_PatientId` (`PatientId` ASC) VISIBLE,

  CONSTRAINT `FK\_Appointment\_DoctorId`

    FOREIGN KEY (`DoctorId`)

    REFERENCES `hospitalmgmt`.`doctors` (`DoctorId`)

    ON DELETE CASCADE

    ON UPDATE CASCADE,

  CONSTRAINT `FK\_Appointment\_HospitalId`

    FOREIGN KEY (`HospitalId`)

    REFERENCES `hospitalmgmt`.`hospitals` (`HospitalId`)

    ON DELETE CASCADE

    ON UPDATE CASCADE,

  CONSTRAINT `FK\_Appointment\_PatientId`

    FOREIGN KEY (`PatientId`)

    REFERENCES `hospitalmgmt`.`patients` (`PatientId`)

    ON DELETE CASCADE

    ON UPDATE CASCADE)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

-- -----------------------------------------------------

-- Table `hospitalmgmt`.`medicines`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `hospitalmgmt`.`medicines` (

  `MedicineId` INT NOT NULL AUTO\_INCREMENT,

  `MedicineCode` VARCHAR(20) NOT NULL,

  `Name` VARCHAR(50) NOT NULL,

  `Company` VARCHAR(50) NULL DEFAULT NULL,

  `Composition` VARCHAR(200) NULL DEFAULT NULL,

  `Dosage` VARCHAR(20) NULL DEFAULT NULL,

  `ExpiryDate` DATETIME NULL DEFAULT NULL,

  `Type` TINYINT NULL DEFAULT NULL,

  `Description` VARCHAR(200) NULL DEFAULT NULL,

  `Cost` DECIMAL(12,2) NULL DEFAULT NULL,

  `AvailableQuantity` INT NULL DEFAULT NULL,

  PRIMARY KEY (`MedicineId`),

  UNIQUE INDEX `MedicineCode\_UNIQUE` (`MedicineCode` ASC) VISIBLE)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

-- -----------------------------------------------------

-- Table `hospitalmgmt`.`appointmentmedicines`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `hospitalmgmt`.`appointmentmedicines` (

  `ApptMedicineId` INT NOT NULL AUTO\_INCREMENT,

  `AppointmentId` INT NOT NULL,

  `MedicineId` INT NOT NULL,

  `Limits` INT NULL DEFAULT NULL,

  PRIMARY KEY (`ApptMedicineId`),

  INDEX `FK\_ApptMedicines\_AppointmentId` (`AppointmentId` ASC) VISIBLE,

  INDEX `FK\_ApptMedicines\_MedicineId` (`MedicineId` ASC) VISIBLE,

  CONSTRAINT `FK\_ApptMedicines\_AppointmentId`

    FOREIGN KEY (`AppointmentId`)

    REFERENCES `hospitalmgmt`.`appointments` (`AppointmentId`)

    ON DELETE CASCADE

    ON UPDATE CASCADE,

  CONSTRAINT `FK\_ApptMedicines\_MedicineId`

    FOREIGN KEY (`MedicineId`)

    REFERENCES `hospitalmgmt`.`medicines` (`MedicineId`)

    ON DELETE CASCADE

    ON UPDATE CASCADE)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

-- -----------------------------------------------------

-- Table `hospitalmgmt`.`specializations`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `hospitalmgmt`.`specializations` (

  `SpecializationId` INT NOT NULL AUTO\_INCREMENT,

  `Name` VARCHAR(50) NOT NULL,

  `KeyAres` VARCHAR(100) NULL DEFAULT NULL,

  `Status` VARCHAR(10) NOT NULL,

  PRIMARY KEY (`SpecializationId`))

ENGINE = InnoDB

AUTO\_INCREMENT = 3

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

-- -----------------------------------------------------

-- Table `hospitalmgmt`.`doctorspecializations`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `hospitalmgmt`.`doctorspecializations` (

  `DocSpecId` INT NOT NULL AUTO\_INCREMENT,

  `DoctorId` INT NOT NULL,

  `SpecializationId` INT NOT NULL,

  PRIMARY KEY (`DocSpecId`),

  INDEX `FK\_DocSpecs\_DoctorId` (`DoctorId` ASC) VISIBLE,

  INDEX `FK\_DocSpecs\_SpecId` (`SpecializationId` ASC) VISIBLE,

  CONSTRAINT `FK\_DocSpecs\_DoctorId`

    FOREIGN KEY (`DoctorId`)

    REFERENCES `hospitalmgmt`.`doctors` (`DoctorId`)

    ON DELETE CASCADE

    ON UPDATE CASCADE,

  CONSTRAINT `FK\_DocSpecs\_SpecId`

    FOREIGN KEY (`SpecializationId`)

    REFERENCES `hospitalmgmt`.`specializations` (`SpecializationId`)

    ON DELETE CASCADE

    ON UPDATE CASCADE)

ENGINE = InnoDB

AUTO\_INCREMENT = 6

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

-- -----------------------------------------------------

-- Table `hospitalmgmt`.`nurses`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `hospitalmgmt`.`nurses` (

  `NurseId` INT NOT NULL AUTO\_INCREMENT,

  `NurseCode` VARCHAR(20) NOT NULL,

  `Name` VARCHAR(50) NOT NULL,

  `Address1` VARCHAR(100) NULL DEFAULT NULL,

  `Address2` VARCHAR(100) NULL DEFAULT NULL,

  `City` VARCHAR(20) NULL DEFAULT NULL,

  `Phone1` VARCHAR(10) NULL DEFAULT NULL,

  `Phone2` VARCHAR(10) NULL DEFAULT NULL,

  `Email` VARCHAR(50) NULL DEFAULT NULL,

  `Salary` DECIMAL(12,2) NULL DEFAULT NULL,

  `Gender` VARCHAR(15) NULL DEFAULT NULL,

  PRIMARY KEY (`NurseId`),

  UNIQUE INDEX `NurseCode\_UNIQUE` (`NurseCode` ASC) VISIBLE)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

-- -----------------------------------------------------

-- Table `hospitalmgmt`.`servicerooms`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `hospitalmgmt`.`servicerooms` (

  `ServiceRoomId` INT NOT NULL AUTO\_INCREMENT,

  `ServiceroomCode` VARCHAR(20) NOT NULL,

  `RoomTypeId` INT NOT NULL,

  `RoomNo` VARCHAR(10) NOT NULL,

  `Results` VARCHAR(200) NULL DEFAULT NULL,

  `Fee` DECIMAL(12,2) NULL DEFAULT NULL,

  `ServiceDate` DATETIME NULL DEFAULT NULL,

  `HospitalId` INT NOT NULL,

  `PatientId` INT NOT NULL,

  `DoctorId` INT NOT NULL,

  PRIMARY KEY (`ServiceRoomId`),

  UNIQUE INDEX `ServiceroomCode\_UNIQUE` (`ServiceroomCode` ASC) VISIBLE,

  INDEX `FK\_ServiceRooms\_DoctorId` (`DoctorId` ASC) VISIBLE,

  INDEX `FK\_ServiceRooms\_HospitalId` (`HospitalId` ASC) VISIBLE,

  INDEX `FK\_ServiceRooms\_PatientId` (`PatientId` ASC) VISIBLE,

  INDEX `FK\_ServiceRooms\_RoomTypeId` (`RoomTypeId` ASC) VISIBLE,

  CONSTRAINT `FK\_ServiceRooms\_DoctorId`

    FOREIGN KEY (`DoctorId`)

    REFERENCES `hospitalmgmt`.`doctors` (`DoctorId`)

    ON DELETE CASCADE

    ON UPDATE CASCADE,

  CONSTRAINT `FK\_ServiceRooms\_HospitalId`

    FOREIGN KEY (`HospitalId`)

    REFERENCES `hospitalmgmt`.`hospitals` (`HospitalId`)

    ON DELETE CASCADE

    ON UPDATE CASCADE,

  CONSTRAINT `FK\_ServiceRooms\_PatientId`

    FOREIGN KEY (`PatientId`)

    REFERENCES `hospitalmgmt`.`patients` (`PatientId`)

    ON DELETE CASCADE

    ON UPDATE CASCADE,

  CONSTRAINT `FK\_ServiceRooms\_RoomTypeId`

    FOREIGN KEY (`RoomTypeId`)

    REFERENCES `hospitalmgmt`.`roomtypes` (`RoomTypeId`)

    ON DELETE CASCADE

    ON UPDATE CASCADE)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

-- -----------------------------------------------------

-- Table `hospitalmgmt`.`treamentmedicines`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `hospitalmgmt`.`treamentmedicines` (

  `TreatmentMedicineId` INT NOT NULL AUTO\_INCREMENT,

  `TreatmentId` INT NOT NULL,

  `MedicineId` INT NOT NULL,

  `Limits` INT NULL DEFAULT NULL,

  PRIMARY KEY (`TreatmentMedicineId`),

  INDEX `FK\_TreamentMedicines\_NurseId` (`MedicineId` ASC) VISIBLE,

  INDEX `FK\_TreamentMedicines\_TreatmentId` (`TreatmentId` ASC) VISIBLE,

  CONSTRAINT `FK\_TreamentMedicines\_NurseId`

    FOREIGN KEY (`MedicineId`)

    REFERENCES `hospitalmgmt`.`medicines` (`MedicineId`)

    ON DELETE CASCADE

    ON UPDATE CASCADE,

  CONSTRAINT `FK\_TreamentMedicines\_TreatmentId`

    FOREIGN KEY (`TreatmentId`)

    REFERENCES `hospitalmgmt`.`treatments` (`TreatmentId`)

    ON DELETE CASCADE

    ON UPDATE CASCADE)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

-- -----------------------------------------------------

-- Table `hospitalmgmt`.`treamentnurses`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `hospitalmgmt`.`treamentnurses` (

  `TreatmentNurseId` INT NOT NULL AUTO\_INCREMENT,

  `TreatmentId` INT NOT NULL,

  `NurseId` INT NOT NULL,

  `Tr\_InTime` DATETIME NULL DEFAULT NULL,

  `Tr\_OutTime` DATETIME NULL DEFAULT NULL,

  `Notes` VARCHAR(200) NULL DEFAULT NULL,

  PRIMARY KEY (`TreatmentNurseId`),

  INDEX `FK\_TreamentNurses\_NurseId` (`NurseId` ASC) VISIBLE,

  INDEX `FK\_TreamentNurses\_TreatmentId` (`TreatmentId` ASC) VISIBLE,

  CONSTRAINT `FK\_TreamentNurses\_NurseId`

    FOREIGN KEY (`NurseId`)

    REFERENCES `hospitalmgmt`.`nurses` (`NurseId`)

    ON DELETE CASCADE

    ON UPDATE CASCADE,

  CONSTRAINT `FK\_TreamentNurses\_TreatmentId`

    FOREIGN KEY (`TreatmentId`)

    REFERENCES `hospitalmgmt`.`treatments` (`TreatmentId`)

    ON DELETE CASCADE

    ON UPDATE CASCADE)

ENGINE = InnoDB

AUTO\_INCREMENT = 2

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

CREATE TABLE IF NOT EXISTS `hospitalmgmt`.`Users` (

  `UserId` INT NOT NULL AUTO\_INCREMENT,

  `UserName` VARCHAR(50) NOT NULL,

  `FirstName` VARCHAR(50),

  `LastName` VARCHAR(50),

  `Email` VARCHAR(50),

  `UserRole` VARCHAR(50),

  `Password` VARCHAR(50),

  `UserroleId` VARCHAR(50),

  `PhoneNo` VARCHAR(20),

  PRIMARY KEY (`UserId`))

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

**STORED PROCEDURE**

DELIMITER $$

USE hospitalmgmt$$

CREATE DEFINER=root@localhost PROCEDURE InsertData(RequestData json, InsType varchar(50))

BEGIN

SET FOREIGN\_KEY\_CHECKS=0;

    If InsType = 'Specializations' Then

        Insert into Specializations(Name,KeyAres,Status)

        Select

            JSON\_VALUE(RequestData, '$.Name') AS Name,

            JSON\_VALUE(RequestData, '$.KeyArea') AS KeyAres,

            JSON\_VALUE(RequestData, '$.Status') AS Status;

         ElseIf InsType = 'paymentmodes' Then

        Insert into paymentmodes(Name,ApplicableMethods)

        Select

            JSON\_VALUE(RequestData, '$.Name') AS Name,

            JSON\_VALUE(RequestData, '$.ApplicableMethods') AS ApplicableMethods;

    elseif InsType = 'roomtype' Then

        Insert into roomtypes (Name,Description)

        Select

            JSON\_VALUE(RequestData, '$.Name') AS Name,

            JSON\_VALUE(RequestData, '$.Description') AS Description;

      elseif InsType = 'doctor' Then

        Insert into doctors (Name,Address1,Address2,City,Phone,mobile,Email,Salary,Designation,Gender)

        Select

            JSON\_VALUE(RequestData, '$.Name') AS Name,

            JSON\_VALUE(RequestData, '$.Address1') AS Address1,

            JSON\_VALUE(RequestData, '$.Address2') AS Address2,

            JSON\_VALUE(RequestData, '$.City') AS City,

            JSON\_VALUE(RequestData, '$.Phone') AS Phone1,

            JSON\_VALUE(RequestData, '$.Mobile') AS Mobile,

            JSON\_VALUE(RequestData, '$.Email') AS Email,

            JSON\_VALUE(RequestData, '$.Salary') AS Salary,

            JSON\_VALUE(RequestData, '$.Designation') AS Designation,

            JSON\_VALUE(RequestData, '$.Gender') AS Gender;

    Elseif InsType = 'hospital' then

        Select @MaxId := Max(HospitalId) from hospitals;

        if @MaxId is null then

            set @MaxId =1000;

        else

            SELECT @MaxId := cast((REPLACE((Select HospitalCode from hospitals where hospitalId =  @MaxId), "HSP", "")) as char(5));

        end if;

        Set @MaxId = @MaxId + 1;

        Select @DocCode := CONCAT('HSP', Cast(@MaxId as char(10)));

        Insert into hospitals(HospitalCode,Name,Address1,Address2,City,Phone1,Phone2,Email,Description)

        Select

            @DocCode AS HospitalCode,

            JSON\_VALUE(RequestData, '$.Name') AS Name,

            JSON\_VALUE(RequestData, '$.Address1') AS Address1,

            JSON\_VALUE(RequestData, '$.Address2') AS Address2,

            JSON\_VALUE(RequestData, '$.City') AS City,

            JSON\_VALUE(RequestData, '$.Phone1') AS Phone1,

            JSON\_VALUE(RequestData, '$.Phone2') AS Phone2,

            JSON\_VALUE(RequestData, '$.Email') AS Email,

            JSON\_VALUE(RequestData, '$.Description') AS Description;

    Elseif InsType = 'nurse' then

        If (Select 1 from nurses where Email = JSON\_VALUE(RequestData, '$.Email') limit 0,1) then

            SIGNAL SQLSTATE '45000'

            SET MESSAGE\_TEXT = 'Duplicate Entry with existing EmailID';

        end if;

        Select @MaxId := Max(NurseId) from nurses;

        if @MaxId is null then

            set @MaxId =1000;

        else

            SELECT @MaxId := cast((REPLACE((Select NurseCode from nurses where nurseId =  @MaxId), "NRS", "")) as char(5));

        end if;

        Set @MaxId = @MaxId + 1;

        Select @DocCode := CONCAT('NRS', Cast(@MaxId as char(10)));

        Insert into nurses(NurseCode,Name,Address1,Address2,City,Phone1,Phone2,Email,gender,Salary)

        Select

            @DocCode AS DoctorCode,

            JSON\_VALUE(RequestData, '$.Name') AS Name,

            JSON\_VALUE(RequestData, '$.Address1') AS Address1,

            JSON\_VALUE(RequestData, '$.Address2') AS Address2,

            JSON\_VALUE(RequestData, '$.City') AS City,

            JSON\_VALUE(RequestData, '$.Phone1') AS Phone1,

            JSON\_VALUE(RequestData, '$.Phone2') AS Phone2,

            JSON\_VALUE(RequestData, '$.Email') AS Email,

            JSON\_VALUE(RequestData, '$.Gender') AS Gender,

            JSON\_VALUE(RequestData, '$.Salary') AS Salary;

    Elseif InsType = 'patient' then

        If (Select 1 from patients where Email = JSON\_VALUE(RequestData, '$.Email') limit 0,1) then

            SIGNAL SQLSTATE '45000'

            SET MESSAGE\_TEXT = 'Duplicate Entry with existing EmailID';

        end if;

        Select @MaxId := Max(patientId) from patients;

        if @MaxId is null then

            set @MaxId =1000;

        else

            SELECT @MaxId := cast((REPLACE((Select patientCode from patients where patientId =  @MaxId), "PAT", "")) as char(5));

        end if;

        Set @MaxId = @MaxId + 1;

        Select @DocCode := CONCAT('PAT', Cast(@MaxId as char(10)));

        Insert into patients(patientCode,Name,Address1,Address2,City,Phone,GuardianPhone,Email,Disease,Gender)

        Select

            @DocCode AS HospitalCode,

            JSON\_VALUE(RequestData, '$.Name') AS Name,

            JSON\_VALUE(RequestData, '$.Address1') AS Address1,

            JSON\_VALUE(RequestData, '$.Address2') AS Address2,

            JSON\_VALUE(RequestData, '$.City') AS City,

            JSON\_VALUE(RequestData, '$.Phone') AS Phone1,

            JSON\_VALUE(RequestData, '$.GuardianPhone') AS GuardianPhone,

            JSON\_VALUE(RequestData, '$.Email') AS Email,

            JSON\_VALUE(RequestData, '$.Disease') AS Disease,

            JSON\_VALUE(RequestData, '$.Gender') AS Gender;

    Elseif InsType = 'medicine' then

        If (Select 1 from medicines where MedicineCode = JSON\_VALUE(RequestData, '$.MedicineCode') limit 0,1) then

            SIGNAL SQLSTATE '45000'

            SET MESSAGE\_TEXT = 'Duplicate Entry with existing MedicineCode';

        end if;

        Select @MaxId := Max(medicineId) from medicines;

        if @MaxId is null then

            set @MaxId =1000;

        else

            SELECT @MaxId := cast((REPLACE((Select MedicineCode from medicines where medicineId =  @MaxId), "MED", "")) as char(5));

        end if;

        Set @MaxId = @MaxId + 1;

        Select @DocCode := CONCAT('MED', Cast(@MaxId as char(10)));

        Insert into medicines(MedicineCode,Name,Company,Composition,Dosage,ExpiryDate,Type,Description,Cost,AvailableQuantity)

        Select

            @DocCode AS HospitalCode,

            JSON\_VALUE(RequestData, '$.Name') AS Name,

            JSON\_VALUE(RequestData, '$.Company') AS Company,

            JSON\_VALUE(RequestData, '$.Composition') AS Composition,

            JSON\_VALUE(RequestData, '$.Dosage') AS Dosage,

            JSON\_VALUE(RequestData, '$.ExpiryDate') AS ExpiryDate,

            JSON\_VALUE(RequestData, '$.Type') AS Type,

            JSON\_VALUE(RequestData, '$.Description') AS Description,

            JSON\_VALUE(RequestData, '$.Cost') AS Cost,

            JSON\_VALUE(RequestData, '$.AvailabelQuantity') AS AvailableQuantity;

    Elseif InsType = 'service' then

        Select @MaxId := Max(ServiceRoomId) from hospitals;

        if @MaxId is null then

            set @MaxId =1000;

        else

            SELECT @MaxId := cast((REPLACE((Select ServiceroomCode from servicerooms where ServiceRoomId =  @MaxId), "SVC", "")) as char(5));

        end if;

        Set @MaxId = @MaxId + 1;

        Select @DocCode := CONCAT('SVC', Cast(@MaxId as char(10)));

        Insert into servicerooms(ServiceroomCode, RoomTypeId, RoomNo, Results, Fee, ServiceDate, HospitalId, PatientId, DoctorId)

        Select

            @DocCode AS HospitalCode,

            JSON\_VALUE(RequestData, '$.RoomTypeId') AS RoomTypeId,

            JSON\_VALUE(RequestData, '$.RoomNo') AS RoomNo,

            JSON\_VALUE(RequestData, '$.Results') AS Results,

            JSON\_VALUE(RequestData, '$.Fee') AS Fee,

            JSON\_VALUE(RequestData, '$.ServiceDate') AS ServiceDate,

            JSON\_VALUE(RequestData, '$.HospitalId') AS HospitalId,

            JSON\_VALUE(RequestData, '$.PatientId') AS PatientId,

            JSON\_VALUE(RequestData, '$.DoctorId') AS DoctorId;

    Elseif InsType = 'operativerooms' then

        Select @MaxId := Max(OperativeRoomId) from operativerooms;

        if @MaxId is null then

            set @MaxId =1000;

        else

            SELECT @MaxId := cast((REPLACE((Select OperativeRoomCode from operativerooms where OperativeRoomId =  @MaxId), "OPS", "")) as char(5));

        end if;

        Set @MaxId = @MaxId + 1;

        Select @DocCode := CONCAT('OPS', Cast(@MaxId as char(10)));

        Insert into operativerooms(OperativeRoomCode, RoomTypeId, RoomNo, BedNo, StartDate, EndDate, HospitalId)

        Select

            @DocCode AS HospitalCode,

            JSON\_VALUE(RequestData, '$.RoomTypeId') AS RoomTypeId,

            JSON\_VALUE(RequestData, '$.RoomNo') AS RoomNo,

            JSON\_VALUE(RequestData, '$.BedNo') AS BedNo,

            JSON\_VALUE(RequestData, '$.StartDate') AS StartDate,

            JSON\_VALUE(RequestData, '$.EndDate') AS EndDate,

            JSON\_VALUE(RequestData, '$.HospitalId') AS HospitalId;

    Elseif InsType = 'medication' then

        Select

            @Tabletype := JSON\_VALUE(RequestData, '$.Type') AS Type,

            @App\_Treatment\_Id := JSON\_VALUE(RequestData, '$.App\_Treatment\_Id') AS App\_Treatment\_Id,

            @MedicineId := JSON\_VALUE(RequestData, '$.MedicineId') AS MedicineId,

            @Limit := JSON\_VALUE(RequestData, '$.Limit') AS Limit1;

        If @Tabletype = 'Appointment'   then

            Delete from appointmentmedicines where AppointmentId = @App\_Treatment\_Id;

            Insert into appointmentmedicines(AppointmentId, MedicineId)

            SELECT @App\_Treatment\_Id,ids FROM JSON\_TABLE(@MedicineId,"$[\*]" COLUMNS(ids BIGINT(20) PATH "$")) AS tt;

        ELSE

            Delete from treamentmedicines where TreatmentId = @App\_Treatment\_Id;

            Insert into treamentmedicines(TreatmentId, MedicineId)

            SELECT @App\_Treatment\_Id,ids FROM JSON\_TABLE(@MedicineId,"$[\*]" COLUMNS(ids BIGINT(20) PATH "$")) AS tt;

        End If;

    Elseif InsType = 'treatmentNurse' then

        Select

            @TreatmentId := JSON\_VALUE(RequestData, '$.TreatmentId') AS TreatmentId,

            @NurseId := JSON\_VALUE(RequestData, '$.NurseId') AS NurseId,

            @Tr\_InTime := JSON\_VALUE(RequestData, '$.Tr\_InTime') AS Tr\_InTime,

            @Tr\_OutTime := JSON\_VALUE(RequestData, '$.Tr\_OutTime') AS Tr\_OutTime,

            @Notes := JSON\_VALUE(RequestData, '$.Notes') AS Notes;

        Insert into treamentnurses(TreatmentId, NurseId, Tr\_InTime, Tr\_OutTime, Notes)

        Select @TreatmentId, @NurseId,@Tr\_InTime, @Tr\_OutTime, @Notes ;

    Elseif InsType = 'doctorspecializations' then

        Select

            @DoctorId := JSON\_VALUE(RequestData, '$.DoctorId') AS DoctorId,

            @SpecializationId := JSON\_VALUE(RequestData, '$.SpecializationId') AS SpecializationId;

        Delete from doctorspecializations where DoctorId = @DoctorId;

        Insert into doctorspecializations(DoctorId, SpecializationId)

        SELECT @DoctorId,ids FROM JSON\_TABLE(@SpecializationId,"$[\*]" COLUMNS(ids BIGINT(20) PATH "$")) AS tt;

    Elseif InsType = 'treatment' then

        Select @MaxId := Max(TreatmentId) from treatments;

        if @MaxId is null then

            set @MaxId =1000;

        else

            SELECT @MaxId := cast((REPLACE((Select TreatmentCode from treatments where TreatmentId =  @MaxId), "TRT", "")) as char(5));

        end if;

        Set @MaxId = @MaxId + 1;

        Select @DocCode := CONCAT('TRT', Cast(@MaxId as char(10)));

        Insert into treatments(TreatmentCode, Name, Duration, DoctorId, PatientId, OperativeRoomId)

        Select

            @DocCode AS HospitalCode,

            JSON\_VALUE(RequestData, '$.Name') AS Name,

            JSON\_VALUE(RequestData, '$.Duration') AS Duration,

            JSON\_VALUE(RequestData, '$.DoctorId') AS DoctorId,

            JSON\_VALUE(RequestData, '$.PatientId') AS PatientId,

            JSON\_VALUE(RequestData, '$.OperativeRoomId') AS OperativeRoomId;

    Elseif InsType = 'admission' then

        Select @MaxId := Max(admissionId) from admissions;

        if @MaxId is null then

            set @MaxId =1000;

        else

            SELECT @MaxId := cast((REPLACE((Select AdmissionCode from admissions where admissionId =  @MaxId), "ADM", "")) as char(5));

        end if;

        Set @MaxId = @MaxId + 1;

        Select @DocCode := CONCAT('ADM', Cast(@MaxId as char(10)));

        Insert into admissions(AdmissionCode, TreatmentId, AdmittedDate, IsInsuranceApplicable, InsurancePhotoId, PaymentId, DischargeDate)

        Select

            @DocCode AS HospitalCode,

            JSON\_VALUE(RequestData, '$.TreatmentId') AS TreatmentId,

            JSON\_VALUE(RequestData, '$.AdmittedDate') AS AdmittedDate,

            JSON\_VALUE(RequestData, '$.IsInsuranceApplicable') AS IsInsuranceApplicable,

            JSON\_VALUE(RequestData, '$.InsurancePhotoId') AS InsurancePhotoId,

            JSON\_VALUE(RequestData, '$.PaymentId') AS PaymentId,

            JSON\_VALUE(RequestData, '$.DishargeDate') AS DischargeDate;

    Elseif InsType = 'payment' then

        Select @MaxId := Max(PaymentId) from Payments;

        if @MaxId is null then

            set @MaxId =1000;

        else

            SELECT @MaxId := cast((REPLACE((Select PaymentCode from Payments where PaymentId =  @MaxId), "PAY", "")) as char(5));

        end if;

        Set @MaxId = @MaxId + 1;

        Select @DocCode := CONCAT('PAY', Cast(@MaxId as char(10)));

        Insert into Payments(PaymentCode, PaymentmodeId, PayeeName, TotalAmount, DiscountAmount, InsuranceAmount, PayableAmount)

        Select

            @DocCode AS PaymentCode,

            JSON\_VALUE(RequestData, '$.PaymentmodeId') AS PaymentmodeId,

            JSON\_VALUE(RequestData, '$.PayeeName') AS PayeeName,

            JSON\_VALUE(RequestData, '$.TotalAmount') AS TotalAmount,

            JSON\_VALUE(RequestData, '$.DiscountAmount') AS DiscountAmount,

            JSON\_VALUE(RequestData, '$.InsuranceAmount') AS InsuranceAmount,

            JSON\_VALUE(RequestData, '$.PayableAmount') AS PayableAmount;

    Elseif InsType = 'appointment' then

        Select @MaxId := Max(AppointmentId) from appointments;

        if @MaxId is null then

            set @MaxId =1000;

        else

            SELECT @MaxId := cast((REPLACE((Select AppointmentCode from appointments where AppointmentId =  @MaxId), "PAY", "")) as char(5));

        end if;

        Set @MaxId = @MaxId + 1;

        Select @DocCode := CONCAT('PAY', Cast(@MaxId as char(10)));

        Insert into appointments(AppointmentCode, DoctorId, PatientId, HospitalId, Appt\_Date, Next\_Appt\_Date, DiseaseNotes, Fee)

        Select

            @DocCode AS HospitalCode,

            JSON\_VALUE(RequestData, '$.DoctorId') AS DoctorId,

            JSON\_VALUE(RequestData, '$.PatientId') AS PatientId,

            JSON\_VALUE(RequestData, '$.HospitalId') AS HospitalId,

            JSON\_VALUE(RequestData, '$.Appt\_Date') AS Appt\_Date,

            JSON\_VALUE(RequestData, '$.Next\_Appt\_Date') AS Next\_Appt\_Date,

            JSON\_VALUE(RequestData, '$.DiseaseNotes') AS DiseaseNotes,

            JSON\_VALUE(RequestData, '$.Fee') AS Fee;

        Elseif InsType = 'treamentmedicines' then

         Insert into treamentmedicines(TreatmentId, MedicineId,Limits)

        Select

            JSON\_VALUE(RequestData, '$.TreatmentId') AS TreatmentId,

            JSON\_VALUE(RequestData, '$.MedicineId') AS MedicineId,

            JSON\_VALUE(RequestData, '$.Limit') AS Limits;

            Elseif InsType = 'appointmentmedicine' then

         Insert into appointmentmedicines(AppointmentId, MedicineId,Limits)

        Select

            JSON\_VALUE(RequestData, '$.AppointmentId') AS AppointmentId,

            JSON\_VALUE(RequestData, '$.MedicineId') AS MedicineId,

            JSON\_VALUE(RequestData, '$.Limit') AS Limits;

    Elseif InsType = 'Users' then

         Insert into Users(UserName,FirstName,LastName,Email,UserRole,Password,PhoneNo)

        Select

            JSON\_VALUE(RequestData, '$.UserName') AS UserName,

            JSON\_VALUE(RequestData, '$.FirstName') AS FirstName,

            JSON\_VALUE(RequestData, '$.LastName') AS LastName,

            JSON\_VALUE(RequestData, '$.Email') AS Email,

            JSON\_VALUE(RequestData, '$.UserRole') AS UserRole,

            JSON\_VALUE(RequestData, '$.Password') AS Password,

            JSON\_VALUE(RequestData, '$.PhoneNo') AS PhoneNo;

    End If;

END$$

DELIMITER ;

**Appendix A**

**Angular Coding Standards**

**And**

**Best Programming Practices**

Table of Contents

1. [Basic Principles 38](#_Toc24726753)

[2. Common Guidelines 38](#_Toc24726755)

3.  [Naming Conventions and Standards 38](#_Toc24726756)

[4. Application Design Guidelines 41](#_Toc24726757)

[5. Performance Guidelines 42](#_Toc24726758)

[6. Programming practices 42](#_Toc24726759)

[7. Comments 44](#_Toc24726760)

# Basic Principles

Set of basic principles that apply to all situations, regardless of context. These include:

* **Single responsibility**: A class should have one and only one reason to change meaning that a class should have only one job
* **The Principle of Least Surprise** (or Astonishment): Choose a solution that everyone can understand, and that keeps them on the right track.
* **Keep It Simple Stupid** (a.k.a. KISS): the simplest solution is more than enough.
* **You Ain't Gonna Need It** (a.k.a. YAGNI): create a solution for the problem at hand, not for the ones you think may happen later on.
* **Don't Repeat Yourself** (a.k.a. DRY): avoid duplication within a component, a source control repository or a bounded context, without forgetting the Rule of Three heuristic.
* The four principles of object-oriented programming: encapsulation, abstraction, inheritance and polymorphism.

# Common Guidelines

* Use codelyzer (<https://www.npmjs.com/package/codelyzer>) to maintain consistent coding style in the application.
* Use Lazy loading and AuthGuard, along with Routing.
* Use Reactive Forms for all data entry screens.
* Use errorLogService to log errors in log file.

# Naming Conventions and Standards

|  |
| --- |
| Note:  The terms Pascal Casing and Camel Casing are used throughout this document.  **Pascal Casing** - First character of all words are Upper Case and other characters are lower case.  Example: BackColor  **Camel Casing -** First character of all words, except the first word are Upper Case and other characters are lower case.  Example: backColor |

3.1 General Naming Guidelines

* Use consistent names for all symbols. Recommended naming pattern is feature.type.ts.
  1. Separate filenames with dots and dashes
* Use conventional type names including .service, .component, .pipe, .module, and .directive
* Bad practice: using Abbreviations such as .srv, .svc, and .serv in place of .service

3.3 Symbols and file names

* Do use pascal case for class names
* Do match the name of the symbol to the name of the file.
* Do give the filename the conventional suffix (such as .component.ts, .directive.ts, .module.ts, .pipe.ts, or .service.ts) for a file of that type.

Examples

|  |  |
| --- | --- |
| @Component({ ... })  export class AppComponent { } | app.component.ts |
| @Directive({ ... })  export class ValidationDirective { } | validation.directive.ts |
| @NgModule({ ... })  export class AppModule | app.module.ts |
| @Pipe({ name: 'initCaps' })  export class InitCapsPipe implements PipeTransform { } | init-caps.pipe.ts |

1. Service names

* Use consistent names for all services named after their feature.
* Do suffix a service class name with Service. For example, something that gets data or heroes should be called a DataService or a HeroService.

Examples

|  |  |
| --- | --- |
| @Injectable()  export class HeroDataService { } | hero-data.service.ts |
| @Injectable()  export class CreditService { } | credit.service.ts |
| @Injectable()  export class Logger { } | logger.service.ts |

1. Angular NgModule names

* Append the symbol name with the suffix Module.
* Give the filename the .module.ts extension.
* Do name the module after the feature and folder it resides in.
* Do suffix a RoutingModule class name with RoutingModule.
* Do end the filename of a RoutingModule with -routing.module.ts.

Examples

|  |  |
| --- | --- |
| @NgModule({ ... })  export class AppModule { } | app.module.ts |
| @NgModule({ ... })  export class VillainsModule { } | villains.module.ts |
| @NgModule({ ... })  export class AppRoutingModule { } | app-routing.module.ts |
| @NgModule({ ... })  export class HeroesRoutingModule { } | heroes-routing.module.ts |

1. Classes

* Use PascalCase, when naming classes.
* Good: ExceptionService
* Bad: exceptionService

1. Constants

* Declare variables with constant if their values should not change during the application lifetime.

1. Interfaces

* Name interface using PascalCase.
* Consider naming an interface without an I prefix.
* Consider using a class instead of an interface for services and declarable (components, directives, and pipes).

1. Properties and methods

* Do use camel case to name properties and methods.
* Avoid prefixing private properties and methods with an underscore.

1. Spacing

* Consider leaving one empty line between third party imports and application imports.

1. Do not use single character variable names like i, n, s etc. Use names like index, temp

One exception in this case would be variables used for iterations in loops:

* + 1. for ( let i = 0; i < count; i++ )
    2. {
       1. ...
    3. }

If the variable is used only as a counter for iteration and is not used anywhere else in the loop, many people still like to use a single char variable (i) instead of inventing a different suitable name.

1. Don't prefix fields. For example, don't use g\_ or s\_ to distinguish static from non-static fields.  
   Examples of incorrect identifier names are: \_currentUser, mUserName, m\_loginTime.
2. Don't repeat the name of a class or enumeration in its members.
3. Do not use variable names that resemble keywords.

# Application Design Guidelines

* A class or interface should have a single purpose.
* An interface should be small and focused. Interfaces should have a name that clearly explains their purpose or role in the system.
* It should be possible to treat a derived object as if it were a base class object
* A property, method or local function should do only one thing
* **SharedModules**: Do declare components, directives, and pipes in a shared module when those items will be re-used and referenced by the components declared in other feature modules.
* Extract templates and styles into a separate file, when more than 3 lines. Name the template file [component-name].component.html, where [component-name] is the component name.
* Giving components an element selector, as opposed to attribute or class selectors.
* Limit logic in a component to only that required for the view. All other logic should be delegated to services.
* Move reusable logic to services and keep components simple and focused on their intended purpose.
* Put presentation logic in the component class, and not in the template.
* Services are ideal for sharing methods across a feature area or an app.

# Performance Guidelines

* When using ngFor to loop over an array in templates, use it with a trackByfunction which will return an unique identifier for each item.
* When subscribing to observables, always make sure you unsubscribe from them appropriately by using operators like take, takeUntil, etc.
* Avoid Function Calls in Views

# Programming practices

* Avoid writing very long methods. A method should typically have 1~25 lines of code. If a method has more than 25 lines of code, you must consider refactoring into separate methods.
* Method name should tell what it does. Do not use mis-leading names. If the method name is obvious, there is no need of documentation explaining what the method does.

Good:

function divide(divident, divisor) {

const quotient = divident / divisor;

return quotient;

}

Not Good:

function div(x, y)) {

const val = x / y;

return val;

}

* A method should do only 'one job'. Do not combine more than one job in a single method, even if those jobs are very small.

Good:

addData(data: Data) {

return this.http.post<Data>(url, data)

.pipe(this.catchHttpErrors());

}

updateData(data: Data) {

return this.http.put<Data>(`${url}/${data.id}`, data)

.pipe(this.catchHttpErrors());

}

Not Good:

addOrUpdateData(data: Data, status: boolean) {

if (status) {

return this.http.post<Data>(url, data)

.pipe(this.catchHttpErrors());

}

return this.http.put<Data>(`${url}/${data.id}`, data)

.pipe(this.catchHttpErrors());

}

}

* Don’t use nested subscriptions

Avoid:

this.returnsObservable1(...)

.subscribe(

success => {

this.returnsObservable2(...)

.subscribe(

success => {

this.returnsObservable3(...)

.subscribe(

success => {

...

},

Good:

this.returnsObservable1(...)

.pipe(

flatMap(success => this.returnObservable2(...),

flatMap(success => this.returnObservable3(...)

)

.subscribe(success => {...});

* Do not hardcode numbers. Use constants instead.
* Do not hardcode strings. Use resource files or constant file.
* The event handler should not contain the code to perform the required action. Rather call another method from the event handler.
* Error messages should help the user to solve the problem. Never give error messages like "Error in Application", "There is an error" etc. Instead give specific messages like "Failed to update database. Please make sure the login id and password are correct."
* When displaying error messages, in addition to telling what is wrong, the message should also tell what the user should do to solve the problem. Instead of message like "Failed to update database.", suggest what should the user do: "Failed to update database. Please make sure the login id and password are correct."
* Show short and friendly message to the user. But log the actual error with all possible information. This will help a lot in diagnosing problems.
* Avoid having very large files. If a single file has more than 1000 lines of code, it is a good candidate for refactoring. Split them logically into two or more classes.
* Avoid passing too many parameters to a method. If you have more than 4~5 parameters, it is a good candidate to define a class or structure.
* Logically organize all your files within appropriate folders. Use 2 level folder hierarchies. You can have up to 10 folders in the root folder and each folder can have up to 5 sub folders. If you have too many folders than cannot be accommodated with the above mentioned 2 level hierarchy, you may need re factoring into multiple assemblies.
* Don’t use var, use let.
* If value is not to change, use const.
* Use separate environment files for each environment.
* Avoid declaring datatype in component itself. Write this in a separate class/interface file.
* Use interface instead of ‘any’ if data structure is known.
* Create a custom http service and use that for all http calls, it will help you to maintain code.

# Comments

Good and meaningful comments make code more maintainable. However,

* Do not write comments for every line of code and every variable declared.
* Write comments wherever required. But good readable code will require very less comments. If all variables and method names are meaningful, that would make the code very readable and will not need many comments.
* Do not write comments if the code is easily understandable without comment. The drawback of having lot of comments is, if you change the code and forget to change the comment, it will lead to more confusion.
* If you have to use some complex or weird logic for any reason, document it very well with sufficient comments.
* Perform spelling check on comments and also make sure proper grammar and punctuation is used.

APPENDIX B

**C# Coding Standards**

**References**

1. C# Coding Conventions - C# Programming Guide | Microsoft Docs - <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/inside-a-program/coding-conventions>
2. Naming Guidelines | Microsoft Docs –

<https://docs.microsoft.com/en-us/dotnet/standard/design-guidelines/naming-guidelines>