**Shop Management System**

**(ET Departmentals)**

**Prepared by**

**Narendula Nithin**

**Problem Statement**

Design simple application for Shop Management System website with Asp.net core, Web API, Angular 6+, Angular and dotnet unit testing using Jasmine &Karma and Ms Test, proper normalization, collection, proper exception handling, EF, Log4net and C#. Implements all the checks so that there are no errors when Records are added, removed, updated, searched from collections, and use Entity Framework. Write effective unit test cases using MS Test framework Follow AGILE principles and SCRUM model to create the following. For the above requirements, prepare epic, user stories, task, and sub tasks relevant to the client requirements. Integrate GIT with IDE and manage files.

**ET Departmental stores** looking to automate their day-to-day activity of the shop like customer management, Supplier management, sale and invoice management, payments, employee management and their attendance.

Create a Nice and error free application to support ET Departmental store, use required technology and create a responsive UI and stable Entity framework and faster database support.

**Requirement Specifications**

**Technology Stack:**

* SQL Server
* .Net (Entity Framework)
* Angular CLI
* Bootstrap
* Visual Studio
* Visual Studio Code
* Jasmine & Karma
* Git

**SQL Server: (SSMS used in this scenario)**

Used to create tables in the database as per the client/customer requirements.

**.Net: (Implemented in Visual Studio)**

Used to create Web API’s (Repositories and Controllers) and models with respective to the database created.

**Angular CLI: (Implemented in visual studio code)**

User Interface design for the created Web API’s in .Net is done using angular CLI. **(Note: Each Web API created in .net can be considered as components in Angular)**

**Bootstrap:**

Used in designing the user interface in Angular.

**Jasmine & Karma:**

Used for testing Angular code.

**Git:**

Can be used to publish the work, get feedback from the other developers around the world. **(Note: Can also be used as a backup)**

**Input Design:**

The input design is the link between the information system and the user. It comprises the developing specification and procedures for data preparation.

The Input design comprises of a nav bar in which buttons(components) are included to get the respected details from the database.

Each component when opened shows the respective data and 3 buttons with functionalities add data , edit data, and delete data/record and also a search bar to search for a particular record.

When Add or Edit buttons are clicked a popup window shows up for entering the details and saving in the database.

When delete button is clicked a popup window appears to confirm that they want to delete the record or not.

**Modules/Components:**

* Customer
* Customer Ledger
* Employee
* Product
* Product Purchase
* Employee Designation
* Sales
* Purchase
* Supplier
* Supplier Ledger

**Use case Diagram:**

**Diagram

Description automatically generated**

**Sample Code: (Employee Details Web API)**

**Employee Model:**

using System;

using System.Collections.Generic;

#nullable disable

namespace ShopManagementSystem.WebApi.Models

{

public partial class EmployeeDetail

{

public string EmployeeId { get; set; }

public string EmployeeName { get; set; }

public string PhoneNumber { get; set; }

public string MailId { get; set; }

public string Address { get; set; }

public string Department { get; set; }

}

}

**Employee Repository:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

using ShopManagementSystem.WebApi.Models;

namespace ShopManagementSystem.WebApi.Repositories

{

public class EmployeeRepository : IEmployeeRepository

{

private readonly ETsroreDBContext \_context;

public EmployeeRepository(ETsroreDBContext context)

{

\_context = context;

}

public async Task<IEnumerable<EmployeeDetail>> GetEmployeeDetailsAsync()

{

var records = await \_context.EmployeeDetails.ToListAsync();

return records;

}

public async Task<EmployeeDetail> GetEmployee(string employeeId)

{

var book = await \_context.EmployeeDetails.FindAsync(employeeId);

return book;

}

public async Task AddEmployee(EmployeeDetail employee)

{

await \_context.EmployeeDetails.AddAsync(employee);

await \_context.SaveChangesAsync();

}

public async Task<EmployeeDetail> UpdateEmployeeDetailsAsync(EmployeeDetail employee)

{

var emp = new EmployeeDetail()

{

EmployeeId = employee.EmployeeId,

EmployeeName = employee.EmployeeName,

PhoneNumber = employee.PhoneNumber,

MailId = employee.MailId,

Address = employee.Address,

Department = employee.Department

};

\_context.EmployeeDetails.Update(emp);

await \_context.SaveChangesAsync();

return emp;

}

public async Task DeleteEmployeeAsync(string employeeId)

{

var emp = new EmployeeDetail()

{

EmployeeId = employeeId

};

\_context.EmployeeDetails.Remove(emp);

await \_context.SaveChangesAsync();

}

}

}

**Employee Repository:**

**Interface:**

using ShopManagementSystem.WebApi.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace ShopManagementSystem.WebApi.Repositories

{

public interface IEmployeeRepository

{

Task<IEnumerable<EmployeeDetail>> GetEmployeeDetailsAsync();

Task<EmployeeDetail> GetEmployee(string employeeId);

Task AddEmployee(EmployeeDetail employee);

Task<EmployeeDetail> UpdateEmployeeDetailsAsync( EmployeeDetail employee);

Task DeleteEmployeeAsync(string employeeId);

}

}

**Interface Methods Implementation:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

using ShopManagementSystem.WebApi.Models;

namespace ShopManagementSystem.WebApi.Repositories

{

public class EmployeeRepository : IEmployeeRepository

{

private readonly ETsroreDBContext \_context;

public EmployeeRepository(ETsroreDBContext context)

{

\_context = context;

}

public async Task<IEnumerable<EmployeeDetail>> GetEmployeeDetailsAsync()

{

var records = await \_context.EmployeeDetails.ToListAsync();

return records;

}

public async Task<EmployeeDetail> GetEmployee(string employeeId)

{

var book = await \_context.EmployeeDetails.FindAsync(employeeId);

return book;

}

public async Task AddEmployee(EmployeeDetail employee)

{

await \_context.EmployeeDetails.AddAsync(employee);

await \_context.SaveChangesAsync();

}

public async Task<EmployeeDetail> UpdateEmployeeDetailsAsync(EmployeeDetail employee)

{

var emp = new EmployeeDetail()

{

EmployeeId = employee.EmployeeId,

EmployeeName = employee.EmployeeName,

PhoneNumber = employee.PhoneNumber,

MailId = employee.MailId,

Address = employee.Address,

Department = employee.Department

};

\_context.EmployeeDetails.Update(emp);

await \_context.SaveChangesAsync();

return emp;

}

public async Task DeleteEmployeeAsync(string employeeId)

{

var emp = new EmployeeDetail()

{

EmployeeId = employeeId

};

\_context.EmployeeDetails.Remove(emp);

await \_context.SaveChangesAsync();

}

}

}

**Employee Controller:**

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using ShopManagementSystem.WebApi.Repositories;

using ShopManagementSystem.WebApi.Models;

using Microsoft.AspNetCore.Cors;

using System.Data;

namespace ShopManagementSystem.WebApi.Controllers

{

[EnableCors("AllowOrigin")]

[Route("api/[controller]")]

[ApiController]

public class EmployeeController : ControllerBase

{

private readonly IEmployeeRepository \_employeeRepository;

public EmployeeController(IEmployeeRepository employeeRepository)

{

\_employeeRepository = employeeRepository;

}

[HttpGet("")]

public async Task<IEnumerable<EmployeeDetail>> GetEmployeeDetails()

{

var details = await \_employeeRepository.GetEmployeeDetailsAsync();

return details;

}

[HttpGet("{employeeId}")]

[ActionName("GetEmployeeAsync")]

public async Task<IActionResult> GetEmployeeAsync([FromRoute] string employeeId)

{

var detail = await \_employeeRepository.GetEmployee(employeeId);

if (detail != null)

{

return Ok(detail);

}

return NotFound("Card Not Found");

}

[HttpPost("")]

public async Task<IActionResult> AddEmployeeAsync([FromBody] EmployeeDetail employee)

{

await \_employeeRepository.AddEmployee(employee);

return CreatedAtAction(nameof(GetEmployeeAsync), new { employeeID = employee.EmployeeId }, employee);

}

[HttpPut("")]

public async Task<IActionResult> UpdateEmployeeDetailsAsync( [FromBody]EmployeeDetail employee)

{

var detail = await \_employeeRepository.UpdateEmployeeDetailsAsync(employee);

if (detail != null)

{

return Ok(detail);

}

else

{

return NotFound("Card not found");

}

}

[HttpDelete("{employeeId}")]

public async Task<IActionResult> DeleteEmployeeAsync([FromRoute]string employeeId)

{

await \_employeeRepository.DeleteEmployeeAsync(employeeId);

return Ok();

}

}

}