**CAREER HUB C# CODING CHALLENGE**

**GOKUL T M**

**1.Create and implement the mentioned class and the structure in your application.**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CodeChallenge2.model

{

class Applicant

{

public int ApplicantId { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

public string Email { get; set; }

public string Phone { get; set; }

public string Resume { get; set; }

public Applicant(int id, string fname, string lname, string email, string phone, string resume)

{

ApplicantId = id;

FirstName = fname;

LastName = lname;

Email = email;

Phone = phone;

Resume = resume;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CodeChallenge2.model

{

class Company

{

public int CompanyId { get; set; }

public string CompanyName { get; set; }

public string Location { get; set; }

public Company(int companyId, string companyName, string location)

{

CompanyId = companyId;

CompanyName = companyName;

Location = location;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CodeChallenge2.model

{

class JobApplication

{

public int ApplicationId { get; set; }

public int JobId { get; set; }

public int ApplicantId { get; set; }

public DateTime ApplicationDate { get; set; }

public string CoverLetter { get; set; }

public JobApplication(int id, int jobId, int applicantId, DateTime date, string coverLetter)

{

ApplicationId = id;

JobId = jobId;

ApplicantId = applicantId;

ApplicationDate = date;

CoverLetter = coverLetter;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CodeChallenge2.model

{

class JobListing

{

public int JobId { get; set; }

public int CompanyId { get; set; }

public string JobTitle { get; set; }

public string JobDescription { get; set; }

public string JobLocation { get; set; }

public decimal Salary { get; set; }

public string JobType { get; set; }

public DateTime PostedDate { get; set; }

public JobListing(int jobId, int companyId, string title, string description, string location, decimal salary, string jobType, DateTime postedDate)

{

JobId = jobId;

CompanyId = companyId;

JobTitle = title;

JobDescription = description;

JobLocation = location;

Salary = salary;

JobType = jobType;

PostedDate = postedDate;

}

}

}

**2.DatabaseManager Class: Methods**

**InitializeDatabase(): Initializes the database schema and tables.**

**• InsertJobListing(job: JobListing): Inserts a new job listing into the "Jobs" table.**

**• InsertCompany(company: Company): Inserts a new company into the "Companies" table.**

**• InsertApplicant(applicant: Applicant): Inserts a new applicant into the "Applicants" table.**

**• InsertJobApplication(application: JobApplication): Inserts a new job application into the "Applications" table.**

**• GetJobListings(): List: Retrieves a list of all job listings.**

**• GetCompanies(): List: Retrieves a list of all companies**

**• GetApplicants(): List: Retrieves a list of all applicants.**

**• GetApplicationsForJob(jobID: int): List: Retrieves a list of job applications for a specific job listing.**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using CodeChallenge2.model;

namespace CodeChallenge2.dao

{

interface ICareerHubService

{

void InsertCompany(Company company);

void InsertJobListing(JobListing job);

void InsertApplicant(Applicant applicant);

void InsertJobApplication(JobApplication application);

List<Company> GetCompanies();

List<JobListing> GetJobListings();

List<Applicant> GetApplicants();

List<JobApplication> GetApplicationsForJob(int jobId);

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using CodeChallenge2.exception;

using CodeChallenge2.model;

using CourierManagementSystem.util;

using Microsoft.Data.SqlClient;

namespace CodeChallenge2.dao

{

class CareerHubServiceImpl : ICareerHubService

{

private SqlConnection con;

public CareerHubServiceImpl()

{

con = DBConnUtil.GetConnection("appsettings.json");

}

public void InsertCompany(Company company)

{

try

{

con.Open();

string query = "insert into tblcompanies values (@companyId, @companyName, @location)";

SqlCommand cmd = new SqlCommand(query, con);

cmd.Parameters.AddWithValue("@companyId", company.CompanyId);

cmd.Parameters.AddWithValue("@companyName", company.CompanyName);

cmd.Parameters.AddWithValue("@location", company.Location);

cmd.ExecuteNonQuery();

}

catch (SqlException ex)

{

Console.WriteLine("SQL Error while inserting company: " + ex.Message);

}

finally

{

con.Close();

}

}

public void InsertJobListing(JobListing job)

{

try

{

con.Open();

string query = "insert into tbljobs values (@jobId, @companyId, @title, @desc, @loc, @sal, @type, @date)";

SqlCommand cmd = new SqlCommand(query, con);

cmd.Parameters.AddWithValue("@jobId", job.JobId);

cmd.Parameters.AddWithValue("@companyId", job.CompanyId);

cmd.Parameters.AddWithValue("@title", job.JobTitle);

cmd.Parameters.AddWithValue("@desc", job.JobDescription);

cmd.Parameters.AddWithValue("@loc", job.JobLocation);

cmd.Parameters.AddWithValue("@sal", job.Salary);

cmd.Parameters.AddWithValue("@type", job.JobType);

cmd.Parameters.AddWithValue("@date", job.PostedDate);

cmd.ExecuteNonQuery();

}

catch (SqlException ex)

{

Console.WriteLine("SQL Error while inserting job listing: " + ex.Message);

}

finally

{

con.Close();

}

}

public void InsertApplicant(Applicant applicant)

{

try

{

con.Open();

string query = "insert into tblapplicants values (@id, @fname, @lname, @email, @phone, @resume)";

SqlCommand cmd = new SqlCommand(query, con);

cmd.Parameters.AddWithValue("@id", applicant.ApplicantId);

cmd.Parameters.AddWithValue("@fname", applicant.FirstName);

cmd.Parameters.AddWithValue("@lname", applicant.LastName);

cmd.Parameters.AddWithValue("@email", applicant.Email);

cmd.Parameters.AddWithValue("@phone", applicant.Phone);

cmd.Parameters.AddWithValue("@resume", applicant.Resume);

cmd.ExecuteNonQuery();

}

catch (SqlException ex)

{

Console.WriteLine("SQL Error while inserting applicant: " + ex.Message);

}

finally

{

con.Close();

}

}

public void InsertJobApplication(JobApplication application)

{

try

{

con.Open();

string query = "insert into tblapplications values (@id, @jobId, @appId, @date, @letter)";

SqlCommand cmd = new SqlCommand(query, con);

cmd.Parameters.AddWithValue("@id", application.ApplicationId);

cmd.Parameters.AddWithValue("@jobId", application.JobId);

cmd.Parameters.AddWithValue("@appId", application.ApplicantId);

cmd.Parameters.AddWithValue("@date", application.ApplicationDate);

cmd.Parameters.AddWithValue("@letter", application.CoverLetter);

cmd.ExecuteNonQuery();

}

catch (SqlException ex)

{

Console.WriteLine("SQL Error while inserting application: " + ex.Message);

}

finally

{

con.Close();

}

}

public List<Company> GetCompanies()

{

List<Company> list = new List<Company>();

try

{

con.Open();

SqlCommand cmd = new SqlCommand("select \* from tblcompanies", con);

SqlDataReader dr = cmd.ExecuteReader();

while (dr.Read())

{

list.Add(new Company(

Convert.ToInt32(dr["companyId"]),

Convert.ToString(dr["companyName"]),

Convert.ToString(dr["companyLocation"])

));

}

}

catch (SqlException ex)

{

Console.WriteLine("Error fetching companies: " + ex.Message);

}

finally

{

con.Close();

}

return list;

}

public List<JobListing> GetJobListings()

{

List<JobListing> list = new List<JobListing>();

try

{

con.Open();

SqlCommand cmd = new SqlCommand("select \* from tbljobs", con);

SqlDataReader dr = cmd.ExecuteReader();

while (dr.Read())

{

list.Add(new JobListing(

Convert.ToInt32(dr["jobId"]),

Convert.ToInt32(dr["companyId"]),

Convert.ToString(dr["jobTitle"]),

Convert.ToString(dr["jobDescription"]),

Convert.ToString(dr["jobLocation"]),

Convert.ToDecimal(dr["jobSalary"]),

Convert.ToString(dr["jobType"]),

Convert.ToDateTime(dr["postedDate"])

));

}

}

catch (SqlException ex)

{

Console.WriteLine("Error fetching jobs: " + ex.Message);

}

finally

{

con.Close();

}

return list;

}

public List<Applicant> GetApplicants()

{

List<Applicant> list = new List<Applicant>();

try

{

con.Open();

SqlCommand cmd = new SqlCommand("select \* from tblapplicants", con);

SqlDataReader dr = cmd.ExecuteReader();

while (dr.Read())

{

list.Add(new Applicant(

Convert.ToInt32(dr["applicantId"]),

Convert.ToString(dr["firstName"]),

Convert.ToString(dr["lastName"]),

Convert.ToString(dr["email"]),

Convert.ToString(dr["phone"]),

Convert.ToString(dr["applicantResume"])

));

}

}

catch (SqlException ex)

{

Console.WriteLine("Error fetching applicants: " + ex.Message);

}

finally

{

con.Close();

}

return list;

}

public List<JobApplication> GetApplicationsForJob(int jobId)

{

List<JobApplication> list = new List<JobApplication>();

try

{

con.Open();

SqlCommand cmd = new SqlCommand("select \* from tblapplications where jobId = @jobId", con);

cmd.Parameters.AddWithValue("@jobId", jobId);

SqlDataReader dr = cmd.ExecuteReader();

while (dr.Read())

{

list.Add(new JobApplication(

Convert.ToInt32(dr["applicationId"]),

Convert.ToInt32(dr["jobId"]),

Convert.ToInt32(dr["applicantId"]),

Convert.ToDateTime(dr["applicationDate"]),

Convert.ToString(dr["coverLetter"])

));

}

}

catch (SqlException ex)

{

Console.WriteLine("Error fetching applications: " + ex.Message);

}

finally { con.Close(); }

return list;

}

}

}

**3.Exceptions handling Create and implement the following exceptions in your application.**

**• Invalid Email Format Handling: o In the Job Board application, during the applicant registration process, users are required to enter their email addresses.**

**Write a program that prompts the user to input an email address and implement exception handling to ensure that the email address follows a valid format (e.g., contains "@" and a valid domain).**

**If the input is not valid, catch the exception and display an error message. If it is valid, proceed with registration.**

**• Salary Calculation Handling: o Create a program that calculates the average salary offered by companies for job listings. Implement exception handling to ensure that the salary values are non-negative when computing the average. If any salary is negative or invalid, catch the exception and display an error message, indicating the problematic job listings.**

**• File Upload Exception Handling: o In the Job Board application, applicants can upload their resumes as files. Write a program that handles file uploads and implements exception handling to catch and handle potential errors, such as file not found, file size exceeded, or file format not supported. Provide appropriate error messages in each case.**

**• Application Deadline Handling: o Develop a program that checks whether a job application is submitted before the application deadline. Implement exception handling to catch situations where an applicant tries to submit an application after the deadline has passed. Display a message indicating that the application is no longer accepted.**

**• Database Connection Handling: o In the Job Board application, database connectivity is crucial. Create a program that establishes a connection to the database to retrieve job listings. Implement exception handling to catch database-related exceptions, such as connection errors or SQL query errors. Display appropriate error messages and ensure graceful handling of these exceptions.**

namespace CodeChallenge2.exception

{

class ApplicationDeadlinePassedException : Exception

{

public ApplicationDeadlinePassedException(string message) : base(message)

{

}

}

}

namespace CodeChallenge2.exception

{

class DatabaseConnectionException : Exception

{

public DatabaseConnectionException(string message) : base(message)

{

}

}

}

namespace CodeChallenge2.exception

{

class FileUploadException : Exception

{

public FileUploadException(string message) : base(message)

{

}

}

}

class InvalidEmailFormatException : Exception

{

public InvalidEmailFormatException(string message) : base(message)

{

}

}

class NegativeSalaryException : Exception

{

public NegativeSalaryException(string message) : base(message)

{

}

}

**4.Database Connectivity Create and implement the following tasks in your application.**

**• Job Listing Retrieval: Write a program that connects to the database and retrieves all job listings from the "Jobs" table. Implement database connectivity using Entity Framework and display the job titles, company names, and salaries.**

**• Applicant Profile Creation: Create a program that allows applicants to create a profile by entering their information. Implement database connectivity to insert the applicant's data into the "Applicants" table. Handle potential database-related exceptions.**

**• Job Application Submission: Develop a program that allows applicants to apply for a specific job listing. Implement database connectivity to insert the job application details into the "Applications" table, including the applicant's ID and the job ID. Ensure that the program handles database connectivity and insertion exceptions.**

**• Company Job Posting: Write a program that enables companies to post new job listings. Implement database connectivity to insert job listings into the "Jobs" table, including the company's ID. Handle database-related exceptions and ensure the job posting is successful.**

**• Salary Range Query: Create a program that allows users to search for job listings within a specified salary range. Implement database connectivity to retrieve job listings that match the user's criteria, including job titles, company names, and salaries. Ensure the program handles database connectivity and query exceptions.**

using CodeChallenge2.dao;

using CodeChallenge2.exception;

using CodeChallenge2.model;

namespace CodeChallenge2

{

internal class Program

{

static ICareerHubService service = new CareerHubServiceImpl();

static void Main(string[] args)

{

while (true)

{

Console.WriteLine("\n---- CareerHub Menu ----");

Console.WriteLine("1. Insert Company");

Console.WriteLine("2. Insert Job Listing");

Console.WriteLine("3. Insert Applicant");

Console.WriteLine("4. Insert Job Application");

Console.WriteLine("5. Get Job Listings");

Console.WriteLine("6. Get Companies");

Console.WriteLine("7. Get Applicants");

Console.WriteLine("8. Get Applications for Job");

Console.WriteLine("9. Exit");

Console.Write("Enter your choice: ");

int choice;

bool isNumber = int.TryParse(Console.ReadLine(), out choice);

if (!isNumber)

{

Console.WriteLine("Enter valid number!");

continue;

}

switch (choice)

{

case 1:

InsertCompany();

break;

case 2:

try

{

InsertJobListing();

}

catch(Exception e)

{

Console.WriteLine(e.Message);

}

break;

case 3:

try

{

InsertApplicant();

}

catch (Exception e)

{

Console.WriteLine(e.Message);

}

break;

case 4:

try

{

InsertJobApplication();

}

catch (Exception e)

{

Console.WriteLine(e.Message);

}

break;

case 5:

GetJobListings();

break;

case 6:

GetCompanies();

break;

case 7:

GetApplicants();

break;

case 8:

GetApplicationsForJob();

break;

case 9:

Console.WriteLine("Exit");

return;

default:

Console.WriteLine("Invalid choice! Please enter a valid choice");

break;

}

}

}

private static void InsertCompany()

{

Console.Write("Enter Company ID: ");

int id = int.Parse(Console.ReadLine());

Console.Write("Enter Company Name: ");

string name = Console.ReadLine();

Console.Write("Enter Location: ");

string location = Console.ReadLine();

Company company = new Company(id, name, location);

service.InsertCompany(company);

}

private static void InsertJobListing()

{

Console.Write("Enter Job ID: ");

int jobId = int.Parse(Console.ReadLine());

Console.Write("Enter Company ID: ");

int companyId = int.Parse(Console.ReadLine());

Console.Write("Enter Job Title: ");

string title = Console.ReadLine();

Console.Write("Enter Job Description: ");

string desc = Console.ReadLine();

Console.Write("Enter Job Location: ");

string location = Console.ReadLine();

Console.Write("Enter Salary: ");

decimal salary = decimal.Parse(Console.ReadLine());

if (salary < 0) throw new NegativeSalaryException("Salary can't be negative!");

Console.Write("Enter Job Type: ");

string type = Console.ReadLine();

DateTime posted = DateTime.Now;

JobListing job = new JobListing(jobId, companyId, title, desc, location, salary, type, posted);

service.InsertJobListing(job);

}

private static void InsertApplicant()

{

Console.Write("Enter Applicant ID: ");

int id = int.Parse(Console.ReadLine());

Console.Write("Enter First Name: ");

string fname = Console.ReadLine();

Console.Write("Enter Last Name: ");

string lname = Console.ReadLine();

Console.Write("Enter Email: ");

string email = Console.ReadLine();

if (!email.Contains("@") || !email.Contains("."))

throw new InvalidEmailFormatException("Invalid email format!");

Console.Write("Enter Phone: ");

string phone = Console.ReadLine();

Console.Write("Enter Resume Text: ");

string resume = Console.ReadLine();

Applicant applicant = new Applicant(id, fname, lname, email, phone, resume);

service.InsertApplicant(applicant);

}

private static void InsertJobApplication()

{

Console.Write("Enter Application ID: ");

int appId = int.Parse(Console.ReadLine());

Console.Write("Enter Job ID: ");

int jobId = int.Parse(Console.ReadLine());

Console.Write("Enter Applicant ID: ");

int applicantId = int.Parse(Console.ReadLine());

DateTime deadline = new DateTime(2024, 1, 1);

if (DateTime.Now > deadline)

throw new ApplicationDeadlinePassedException("Deadline Passed. Cannot Apply.");

Console.Write("Enter Cover Letter: ");

string cover = Console.ReadLine();

DateTime applied = DateTime.Now;

JobApplication jobApp = new JobApplication(appId, jobId, applicantId, applied, cover);

service.InsertJobApplication(jobApp);

}

private static void GetJobListings()

{

List<JobListing> jobs = service.GetJobListings();

Console.WriteLine("\n--- All Jobs ---");

Console.WriteLine("JobId | CompanyId | JobTitle | JobLocation | Salary | JobType | PostedDate");

foreach (var j in jobs)

{

Console.WriteLine($"{j.JobId} | {j.CompanyId} | {j.JobTitle} | {j.JobLocation} | {j.Salary} | {j.JobType} | {j.PostedDate}");

}

}

private static void GetCompanies()

{

List<Company> companies = service.GetCompanies();

Console.WriteLine("\n--- All Companies ---");

Console.WriteLine("CompanyId | CompanyName | Location");

foreach (var c in companies)

{

Console.WriteLine($"{c.CompanyId} | {c.CompanyName} | {c.Location}");

}

}

private static void GetApplicants()

{

List<Applicant> applicants = service.GetApplicants();

Console.WriteLine("\n--- All Applicants ---");

Console.WriteLine("ApplicantId | FullName | Email | Phone | Resume");

foreach (var a in applicants)

{

Console.WriteLine($"{a.ApplicantId} | {a.FirstName} {a.LastName} | {a.Email} | {a.Phone} | {a.Resume}");

}

}

private static void GetApplicationsForJob()

{

Console.Write("Enter Job ID to see applications: ");

int jobId = int.Parse(Console.ReadLine());

List<JobApplication> apps = service.GetApplicationsForJob(jobId);

Console.WriteLine($"\n--- Applications for Job ID {jobId} ---");

Console.WriteLine("ApplicationId | ApplicantId | CoverLetter");

foreach (var app in apps)

{

Console.WriteLine($"{app.ApplicationId} | {app.ApplicantId} | {app.CoverLetter}");

}

}

}

}

































