Project Description

GENERAL INSTRUCTIONS:

Please carefully read the below instructions

The objective of this assessment is to check your ability to complete a project as per the provided "Project Design".

You are expected to –

1. Write the source code for the classes, methods and packages EXACTLY as mentioned in the "Project Design" section.

2. Ensure that the names of the packages, classes, methods and variables EXACTLY MATCH with the names specified in the "Project Design" section.

3. Understand the project requirements and ACCORDINGLY WRITE the code and logic in the classes and methods so as to meet all given requirements.

Creating the project and testing it –

1. Download the Project Template from the Assessment Portal and extract it on your local machine
2. Extract the Downloaded ZIP file. It imports the Visual Studio Solution containing the Project(s) and the related source files. **You need not to create a separate application on your location machine, instead you are supposed to work on the Solution / Project / Source downloaded / extracted from the downloaded ZIP File.**
3. Define the methods as mentioned in the Project Design Below.
4. Build and test your application locally before uploading it for evaluation.
5. **Before uploading, make sure to delete all the PDB and DLL files from the Project directory. For this delete the content from BIN and OBJ directories.**
6. Close the IDE, Create a .ZIP file by right clicking on the Root directory of the Project and Send to Compressed Folder option.
7. To upload, select the File, and click on Upload.

IMPORTANT NOTE 1: The extension of the zip file should be ONLY .zip (any other zip formats such as .7z will produce unexpected results)

7. After uploading the zip file, you can click on "Compile & Test" button to start the evaluation, and the assessment engine will compile your source code and test it using its pre-defined test cases.

8. If some of the test-cases fail, you can make the fixes in your source code locally on your desktop, and again repeat the above two steps.

9. Once you are finished with all the fixes, you can click on "Submit Test" button, which will show you the final result / score.

NOTE that –

10. The assessment engine will create objects and invoke methods as per the project design, and while doing so, it will use your packages, classes and methods. If your namespaces, classes and methods have a name mismatch or method prototype mismatch w.r.t the expected "Project Design", the tool will show it as an ERROR. If your packages OR classes OR methods OR Script match as per the names but do not perform the expected functionality, the tool will show it as a FAILURE.

11. Unless specified in the Project Design, DO NOT use System. Environment. Exit (0) anywhere in your code. Using System. Environment. Exit (0) in your project code will cause the CPC test engine to exit and it will not be able to run all test-cases.

ONLINE MOVIE TICKETING SYSTEM

The XYZ Inc. have a chain of world class multiplexes. They have decided to automate the process of movie management, ticket selling and movie casting across their theatres in their multiplexes. As a developer you are supposed to write an application for supporting their functional activities. The below requirements are to be implemented in the application.

1. Add a Movie and theatre.
2. Schedule a movie.
3. Manage ticket sale.

Depending on the requirement implementation and need the additional supportive methods like, search etc. can be added.

Database Design: Create a database in SQL Server on your machine locally with the following Specifications.

1. Database Name : MovieTicketing
2. Create the below table, “Movies” in the above database as per the below specifications.

Table Name: Movies			
Column Name	Data Type	Specification	Description
MOVIEID	INT	PRIMARY KEY, IDENTITY(1000,1)	ID of the Movie. This should be auto generated
MOVIENAME	VARCHAR(30)	NOT NULL	Name of the view.
DIRECTOR	VARCAHR(30)		Name of the director.
PLAYSPERDAY	INT	Should be more than Zero (0)	Indicates no of shows / plays per day
TICKETPRICE	MONEY	Should be more than Zero (0)	Price per ticket.

3. Create a table, “Theatres” in the above database.

Table Name: Theatres			
Column Name	Data Type	Specification	Description
THEATREID	INT	PRIMARY KEY, IDENTITY(1000,1)	ID of the Theatre. This should be auto generated
THEATRENAME	VARCHAR(30)	NOT NULL	Name of the Theatre
SEATINGCAPACITY	INT	Should be more than Zero (0)	Total number of seats

4. Create a table, “Shows” in the above database.

Table Name: Shows			
Column Name	Data Type	Specification	Description
SHOWID	INT	PRIMARY KEY, IDENTITY(1000,1)	ID of the Show. This should be auto generated
THEATREID	INT	FOREIGN KEY. Should refer from Theatres table	Theatre ID where the show is casted
MOVIEID	INT	FOREIGN KEY. Should refer from Movies Table.	ID of the Movie to be casted
STARTDATE	DATE	Default System Date	Date from which the Show should start. This must take the default system date. Value should not be inserted explicitly.

ENDDATE	DATE	Not Null	Date on which the show should end.
STARTTIME	TIME	Not Null	The time at which the show starts
ENDTIME	TIME	Not Null	The time at which the show ends

5. Create a table, "Tickets" in the above database.

Table Name: Tickets			
Column Name	Data Type	Specification	Description
TICKETID	INT	PRIMARY KEY, IDENTITY(1000,1)	ID of the Ticket. This should be auto generated
SHOWID	INT	FOREIGN KEY, refers from Shows Table	
REFERENCECODE	VARCHAR(20)	NOT NULL	The reference code should be generated as mentioned in the Appendix 1.
CUSTOMERNAME	VARCHAR(30)	Not Null	Name of the customer
NUMBEROFPERSONS	INT	Not Null	Number of persons allowed per ticket
BOOKINGDATE	DATE	Not Null	Date on which ticket is booked
AMOUNT	MONEY	Not Null	Total amount to be paid
TICKETSTATUS	VARCHAR(10)	Not Null	Status of the ticket.

Application design

The below table gives the overview of the namespaces and classes present in the provided source template. For every class, there will be a separate source file (.CS) in the project. The name of the namespace is "MovieTicketing". This namespace is provided with the below classes

Class Name	**Description**
Movies	This class contains the Properties of the Movie
Theatres	This class contains the Properties of the Theatre
Shows	This class contains the properties of the Show
Tickets	This class contains the properties of the Ticket

The above classes are supposed to contain only the data. These classes must be defined with the required properties as mentioned below.

Class Name	**Description**
TicketingDataAccess	This class contains the methods to perform the database related operations like, Create, Read, Update and Delete. The methods in this class takes the reference of the Movies, Theatres, Shows, and Tickets to perform these operations. The reference to be passed must be chosen as per the method specifications.

Below is the detailed list of features / functionalities to be implemented in the solution.

1. Define the below properties in Movies class.
    - MovieID as int
    - MovieName as string
    - Director as string
    - PlaysPerDay as int
    - TicketPrice as decimal
2. Define the below properties in Theatres class
    - TheatreID as int
    - TheatreName as string
    - SeatingCapacity as int
3. Define the below properties in Shows class
    - ShowID as int
    - TheatreID as int
    - MovieID as int
    - StartDate as datetime
    - EndDate as datetime
    - StartTime as datetime
    - EndTime as datetime
4. Define the below properties in Tickets class
    - TicketID as int
    - ShowID as int
    - CustomerName as string
    - ReferenceCode as string
    - BookingDate as Datetime
    - NumberofPersons as int
    - Amount as decimal
    - TicketStatus as string
5. Define the below methods in **TicketingDataAccess** class.

Method	**Description / Specification**
public bool AddMovie(Movies obj)	This method should take the reference of the movie object containing the movie details to be added into the database. The movie details should be inserted into the Movies table in the database. If the insertion is successful, return "true" else return "false". **Note: If the object reference is null return "false"**.
public bool AddTheatre(Theatres obj)	This method should take the reference of the Theatres object containing the theatre details to be added into the database. The theatre details should added into the Theatres table in the database. If the insertion successful, return "true", else return "false". **Note: If the object reference is null return "false"**
public bool AddShow(Shows obj)	This method should take the reference of the Shows object containing the show details to be added into the database. The show details should be added to the Shows table in the database. If the insertion is successful, must return "true", else return "false". **Note: If the object reference is null return "false"**
public string AddTicket(Tickets obj)	This method should take the reference of the of the Tickets object containing the ticket information to be added into the database. **The REFERENCECODE should be generated as per the procedure mentioned in Appendix 1. The amount to be calculated as [Numberofpersons * Ticker price of the movie]** The ticket details should be added into the Tickets table. If the insertion is successful, should return "reference code" else return "null". **Note: If the object reference is null return "null"**

public int DeleteMovie(int intMovieID)	This method should delete the movie details and all its related information needs to be deleted from the database. This method should return the number of rows deleted from both the movies table and all other related tables. **Note: if the movie doesn't exists return zero (0)**

Appendix 1

- Procedure of generating the reference code is mentioned below.

  Step 1: Concatenate first two characters of the customer name with the number of persons.

  Step 2: To the string generated in Step 1, concatenate first two characters of the Movie name.

  Step 3: To the string generated in Step 2, concatenate Day and month part of the Booking Date.

  Step 4: Generate a random number between 1 and 1000. Concatenate this number to the string generated in Step 3.

  Step 5: To the resultant string generated till Step 3, if any lower case alphabets are found, they should be converted to Upper case.

Example:

Consider the name of the customer as "John", number of persons are 2, name of the movie is, "Journey to the center of the Earth", and the booking date is, "5/19/17", then

Step 1: The first two letters of the customer name, "Jo". This should be concatenated with number of persons which is "2". So string generated is "Jo2".

Step 2: To the string generated in step 1, "Jo2", first two characters of the movie name should be concatenated. The first two characters of the movie name are, "Jo". The resultant string after concatenation is, "Jo2Jo".

Step 3: To the string generated in step 2, which is "Jo2Jo", concatenate Day part (19) and month part of the booking date, so the resultant string will be "Jo2Jo195".

Step 4: Assume 100 is the random number generated between 1 and 1000. This should be concatenated to the string generated in Step 3. The resultant string will be, "Jo2Jo195100".

Step 5: The resultant string in Step 4 Is having lower case alphabets, so they should be converted into upper case. Hence the reference code will be, "JO2JO195100".