🚀 FLIGHT SEARCH ENGINE - Assesment

ASP.NET Core MVC Assignment (2 Hours)

📌 PROBLEM OVERVIEW

Create a **Flight Search Engine** web application that allows users to search for flights based on source and destination. The application should provide two search options: **Flights Only** and **Flights + Hotels**. The data should be retrieved from SQL Server database using stored procedures.

🎯 BUSINESS REQUIREMENTS

Functional Requirements:

1. **Search Criteria Selection**

- User can select Source Location from DropDownList
- User can select Destination Location from DropDownList
- User can enter Number of Persons in TextBox

2. **Search Options**

- **Option 1:** Search Flight Only (Button)
- **Option 2:** Search Flight + Hotel (Button)

3. **Display Results**

- **Flight Only Results:** Show FlightID, FlightName, FlightType, Source, Destination, TotalCost
- **Flight + Hotel Results:** Show FlightID, FlightName, Source, Destination, HotelName, TotalCost

4. **Database Requirements**

- Create Flight table with columns: FlightId, FlightName, FlightType, Source, Destination, PricePerSeat
- Create Hotel table with columns: HotelId, HotelName, HotelType, Location, PricePerDay
- Note: Each city has only one hotel

5. **Technical Requirements**

- Follow MVC Architecture
- Use Stored Procedures for all database operations
- Implement proper validation
- Follow coding standards
- Include unit tests

📊 DATABASE DESIGN

Table: Flights

Column Name	Data Type	Constraints
FlightId	INT	PRIMARY KEY, IDENTITY(1,1)
FlightName	NVARCHAR(100)	NOT NULL
FlightType	NVARCHAR(50)	NOT NULL
Source	NVARCHAR(100)	NOT NULL
Destination	NVARCHAR(100)	NOT NULL
PricePerSeat	DECIMAL(18,2)	NOT NULL

Table: Hotels

Column Name	Data Type	Constraints
HotelId	INT	PRIMARY KEY, IDENTITY(1,1)
HotelName	NVARCHAR(100)	NOT NULL
HotelType	NVARCHAR(50)	NOT NULL
Location	NVARCHAR(100)	NOT NULL
PricePerDay	DECIMAL(18,2)	NOT NULL

🗂 MVC STRUCTURE TO IMPLEMENT

text

FlightSearchEngine/

├── Controllers/

| └── FlightController.cs

├── Models/

│ ├── SearchViewModel.cs

│ ├── FlightResult.cs

│ └── FlightHotelResult.cs

├── Views/

│ └── Flight/

│ ├── Index.cshtml

│ └── Results.cshtml

├── Data/

│ └── DatabaseHelper.cs

└── appsettings.json

📝 **TODO: METHODS AND CLASSES TO IMPLEMENT**

🔷 **TASK 1: Database Stored Procedures (SQL)**

Create the following stored procedures:

#	Stored Procedure Name	Parameters	Return Data
1	sp_GetSources	None	List of distinct Source cities
2	sp_GetDestinations	None	List of distinct Destination cities
3	sp_SearchFlights	@Source, @Destination, @Persons	Flight details with TotalCost
4	sp_SearchFlightsWithHotels	@Source, @Destination, @Persons	Flight+Hotel details with TotalCost

---

🔷 **TASK 2: Model Classes (C#)**

📌 **Class: SearchViewModel (Models/SearchViewModel.cs)**

Purpose: Handle search form data and dropdown lists

Property Name	Data Type	Attributes	Description
Source	string	[Required]	Selected source location
Destination	string	[Required]	Selected destination location
NumberOfPersons	int	[Required][Range(1,10)]	Number of travelers
SourceList	SelectList	-	Dropdown data for sources
DestinationList	SelectList	-	Dropdown data for destinations

Methods to Implement: None (Property-only class)

📌 **Class: FlightResult (Models/FlightResult.cs)**

Purpose: Store flight-only search results

Property Name	**Data Type**	**Description**
FlightId	int	Flight ID from database
FlightName	string	Name of the flight
FlightType	string	Domestic/International
Source	string	Source city
Destination	string	Destination city
TotalCost	decimal	Calculated cost (PricePerSeat × Persons)

Methods to Implement: None (Property-only class)

📌 **Class: FlightHotelResult (Models/FlightHotelResult.cs)**

Purpose: Store flight+hotel package results

Property Name	Data Type	Description
FlightId	int	Flight ID from database
FlightName	string	Name of the flight
Source	string	Source city
Destination	string	Destination city
HotelName	string	Hotel name at destination
TotalCost	decimal	Calculated cost (Flight + Hotel)

Methods to Implement: None (Property-only class)

◈ TASK 3: Database Helper Class (C#)

📌 Class: DatabaseHelper (Data/DatabaseHelper.cs)

Constructor:

csharp

public DatabaseHelper(IConfiguration configuration)

- **Purpose:** Initialize database connection using connection string from appsettings.json
- **Parameters:** IConfiguration
- **Returns:** None

📌 Method 1: GetSourcesAsync

csharp

public async Task<List<string>> GetSourcesAsync()

- **Purpose:** Retrieve all distinct source locations from Flights table
- **Parameters:** None
- **Returns:** List<string> of source cities
- **SQL:** EXEC sp_GetSources

TODO Steps:

- Create SqlConnection
- Create SqlCommand with stored procedure
- Open connection
- Execute reader
- Read data and add to list
- Return list

📌 Method 2: GetDestinationsAsync

csharp

public async Task<List<string>> GetDestinationsAsync()

- **Purpose:** Retrieve all distinct destination locations from Flights table
- **Parameters:** None
- **Returns:** List<string> of destination cities
- **SQL:** EXEC sp_GetDestinations

TODO Steps:

- Create SqlConnection
- Create SqlCommand with stored procedure
- Open connection
- Execute reader
- Read data and add to list
- Return list

📌 Method 3: SearchFlightsAsync

csharp

public async Task<List<FlightResult>> SearchFlightsAsync(string source, string destination, int persons
)

- **Purpose:** Search for flights matching criteria
- **Parameters:**

o source: Source city

o destination: Destination city

o persons: Number of travelers

· **Returns:** List<FlightResult> of matching flights

· **SQL:** EXEC sp_SearchFlights @Source, @Destination, @Persons

TODO Steps:

· Create SqlConnection

· Create SqlCommand with stored procedure

· Add parameters

· Open connection

· Execute reader

· Map data to FlightResult objects

· Add to list and return

📌 Method 4: SearchFlightsWithHotelsAsync

csharp

public async Task<List<FlightHotelResult>> SearchFlightsWithHotelsAsync(string source, string destination, int persons)

· **Purpose:** Search for flight+hotel packages

· **Parameters:**

o source: Source city

o destination: Destination city

o persons: Number of travelers

· **Returns:** List<FlightHotelResult> of matching packages

· **SQL:** EXEC sp_SearchFlightsWithHotels @Source, @Destination, @Persons

TODO Steps:

· Create SqlConnection

· Create SqlCommand with stored procedure

· Add parameters

· Open connection

· Execute reader

· Map data to FlightHotelResult objects

· Add to list and return

🔷 TASK 4: Controller Class (C#)

📌 Class: FlightController (Controllers/FlightController.cs)

Constructor:

csharp

public FlightController(IConfiguration configuration)

- **Purpose:** Initialize DatabaseHelper
- **Parameters:** IConfiguration
- **Returns:** None

📌 Action Method 1: Index (GET)

csharp

public async Task<IActionResult> Index()

- **Purpose:** Display search form with populated dropdowns
- **Parameters:** None
- **Returns:** View with SearchViewModel

TODO Steps:

- Create new SearchViewModel
- Call GetSourcesAsync() and GetDestinationsAsync()
- Create SelectList from results
- Assign to ViewModel properties
- Return View with ViewModel

📌 Action Method 2: SearchFlights (POST)

csharp

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> SearchFlights(SearchViewModel model)

- **Purpose:** Process flight-only search and display results
- **Parameters:** SearchViewModel with user input
- **Returns:** View with FlightResult list

TODO Steps:

- Check ModelState.IsValid
- If invalid, reload dropdowns and return to Index
- Call SearchFlightsAsync with model data
- Store results in ViewBag for display
- Return Results view with data

📌 Action Method 3: SearchFlightsWithHotels (POST)

csharp

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> SearchFlightsWithHotels(SearchViewModel model)

- **Purpose:** Process flight+hotel search and display results
- **Parameters:** SearchViewModel with user input
- **Returns:** View with FlightHotelResult list

TODO Steps:

- Check ModelState.IsValid
- If invalid, reload dropdowns and return to Index
- Call SearchFlightsWithHotelsAsync with model data
- Store results in ViewBag for display
- Return Results view with data

🔷 TASK 5: Views (Razor)

📌 View 1: Index.cshtml

Purpose: Search form with dropdowns and buttons

UI Elements to Create:

- Form with method="post"
- Dropdown for Source (asp-items)
- Dropdown for Destination (asp-items)
- Textbox for NumberOfPersons (type="number")
- Validation summary
- "Search Flights Only" button (green)

- "Search Flight + Hotel" button (yellow)
- Anti-forgery token

JavaScript to Add:

- Prevent selecting same source and destination
- Add loading spinner on button click
- Client-side validation

📌 View 2: Results.cshtml

Purpose: Display search results in table format

UI Elements to Create:

- Check if Model is null or empty → Show "No results" message
- If Model is FlightResult type → Display flight-only table
- If Model is FlightHotelResult type → Display package table
- Back to Search button
- Breadcrumb navigation

Table Columns for Flight Only:
| FlightId | FlightName | FlightType | Source | Destination | TotalCost |

Table Columns for Flight+Hotel:
| FlightId | FlightName | Source | Destination | HotelName | TotalCost |

🔷 TASK 6: Configuration

📌 File: appsettings.json

json

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Your SQL Server connection string"
  }
}
```

TODO:

- Add your SQL Server connection string
- Test the connection

✅ COMPLETION CHECKLIST

Database (20 mins)

- Create database
- Create Flights table
- Create Hotels table
- Insert sample data
- Create sp_GetSources
- Create sp_GetDestinations
- Create sp_SearchFlights
- Create sp_SearchFlightsWithHotels

Models (15 mins)

- SearchViewModel with properties and validation
- FlightResult class
- FlightHotelResult class

Data Layer (20 mins)

- DatabaseHelper constructor
- GetSourcesAsync method
- GetDestinationsAsync method
- SearchFlightsAsync method
- SearchFlightsWithHotelsAsync method

Controller (15 mins)

- FlightController constructor
- Index (GET) action
- SearchFlights (POST) action
- SearchFlightsWithHotels (POST) action

Views (20 mins)

- Index.cshtml with form
- Client-side validation
- Results.cshtml with tables
- No results handling

Testing (10 mins)

- Dropdowns populated correctly
- Validation working
- Flight search returns results
- Flight+Hotel search returns results
- Error handling works

🎓 STUDENT TIPS

Time Management (2 Hours)

Time	Task
0:00 - 0:20	Database setup
0:20 - 0:35	Create Models
0:35 - 0:55	Create DatabaseHelper
0:55 - 1:10	Create Controller
1:10 - 1:30	Create Views
1:30 - 1:45	Testing & Debugging
1:45 - 2:00	Final touches

Common Mistakes to Avoid

- ❌ Forgetting to add [ValidateAntiForgeryToken]
- ❌ Not checking ModelState.IsValid
- ❌ Missing using statements
- ❌ Incorrect connection string
- ❌ Not handling null results in view

Quick Debugging Tips

- Use try-catch to see errors

- Check SQL Server is running
- Verify stored procedure names
- Use breakpoints to check data

📤 DELIVERABLES

Submit the following:

1. **SQL Script** - Complete database creation script
2. **Source Code** - Entire ASP.NET Core MVC project
3. **Screenshots** -

- Search page with populated dropdowns
- Flight-only search results
- Flight+hotel search results
- Validation messages