Below is a **complete, clear, scalable project structure** you can follow for building this Employee Registration + Listing + Detailed View system using:

✔ **ASP.NET Web API (Backend)**
✔ **ASP.NET MVC (Frontend)**
✔ **SQL Server (SSMS)**
✔ **External API for Country → State → City cascading dropdown**
✔ **Employee with multiple qualifications + photo upload + marksheet upload**

# ✅ **High-Level Architecture**

```
Solution
│
├── EmployeeManagement.API          (Web API Project)
│
└── EmployeeManagement.Web          (MVC UI Project)
```

DB: SQL Server
External API for Location: e.g.
https://countriesnow.space/api,
https://restcountries.com, or **GeoDB Cities API**

# ✅ **1. DATABASE DESIGN**

### 📌 Tables

### **Employees**

Column	Type
EmployeeId (PK)	int
FirstName	varchar(100)
LastName	varchar(100)
Email	varchar(200)
Phone	varchar(50)
Address	varchar(max)
Country	varchar(100)

State	varchar(100)
City	varchar(100)
PhotoPath	varchar(max)
CreatedDate	datetime

### **Qualifications**

Column	Type
QualificationId (PK)	int
EmployeeId (FK)	int
Degree	varchar(100)
PassingYear	int
Percentage	decimal(5,2)
CollegeName	varchar(200)
MarksheetPath	varchar(max)

# ✅ **2. PROJECT STRUCTURE – WEB API**

### Folder layout:

```
EmployeeManagement.API
│
├── Controllers
│     ├── EmployeesController.cs
│     ├── QualificationController.cs
│     └── LocationController.cs   (calls external API)
│
├── Models
│     ├── Employee.cs
│     ├── Qualification.cs
│
├── DTOs
│     ├── EmployeeCreateDTO.cs
│     ├── EmployeeDetailDTO.cs
│     ├── QualificationDTO.cs
│
├── Services
│     ├── EmployeeService.cs
│     ├── QualificationService.cs
│     └── LocationService.cs  (external API integration)
│
```

```
│
├── Repositories
│   ├── EmployeeRepository.cs
│   ├── QualificationRepository.cs
│
├── Data
│   └── ApplicationDbContext.cs
│
└── Mappings
    └── AutoMapperProfile.cs
```

## 📌 API Responsibilities

### **EmployeesController**

* POST `/api/employees` → Create employee
* GET `/api/employees` → List employees
* GET `/api/employees/{id}` → Employee detail
* PUT `/api/employees/{id}` → Update
* DELETE

### **QualificationController**

* POST `/api/qualification/add`
* PUT `/api/qualification/update`
* DELETE

### **LocationController**

* GET `/country`
* GET `/state/{country}`
* GET `/city/{state}`
  These call **external APIs** and return controlled clean output.

# ⚙️ **How External Country-State-City API Works**

### **LocationService.cs**

```csharp
public async Task<List<string>> GetCountries() {
    var url = "https://countriesnow.space/api/v0.1/countries/positions";
    var response = await _httpClient.GetAsync(url);
    return await response.Content.ReadFromJsonAsync<List<string>>();
}
```

Similar methods for states and cities.

API returns simple lists to MVC for dropdowns.

# ✅ **3. FRONTEND – ASP.NET MVC PROJECT**

```
EmployeeManagement.Web
│
├── Controllers
│     ├── EmployeeController.cs
│     └── LocationController.cs   (calls API)
│
├── Views
│     ├── Employee
│     │      ├── Create.cshtml
│     │      ├── Index.cshtml
│     │      ├── Details.cshtml
│     │      └── _QualificationRow.cshtml (partial)
│
├── Models
│     ├── EmployeeVM.cs
│     ├── QualificationVM.cs
│
└── Helpers
      └── ApiHttpClient.cs
```

# 📝 **Frontend Flow**

## **1. Employee Create Page**

### Contains:

✔ Employee basic form
✔ Photo upload
✔ Country → State → City (AJAX cascading dropdown using API)

✔ Qualification table (dynamic add/remove rows)
✔ Each row uploads marksheet image

Page UI matches exactly the images you uploaded.

### 🔄 Cascading Dropdown Flow

```
MVC → API → External API
```

Example in jQuery:

```javascript
$("#Country").change(function () {
    $.get("/Location/GetStates?country=" + $(this).val(), function (data) {
         $("#State").empty();
         $.each(data, function (i, state) {
             $("#State").append(`<option
value="${state}">${state}</option>`);
         });
    });
});
```

# 📌 **4. FILE UPLOAD HANDLING**

### Employee Photo

Uploaded to API → stored in:

```
/wwwroot/uploads/photos/
```

### Marksheet Image

Uploaded to:

```
/wwwroot/uploads/marksheets/
```

API returns stored file path → saved in DB.

# 🧩 **5. MVC UI Pages**

### **Index.cshtml (Employee List)**

Columns:

| Id | Name | Email | Phone | Qualifications | View Button |

View button → `/Employee/Details/{id}`

### **Details.cshtml**

✔ Profile Picture
✔ All Employee Fields
✔ Qualification Table (with marksheet view button)

# 📦 **6. Dependency Injection (API)**

```
builder.Services.AddScoped<IEmployeeService, EmployeeService>();
builder.Services.AddScoped<ILocationService, LocationService>();
builder.Services.AddScoped<IQualificationService, QualificationService>();
builder.Services.AddHttpClient();
```

# 🚀 **7. Suggested Complete Workflow**

### Step 1 → Create database

### Step 2 → Build API models, DTOs, repositories, services

### Step 3 → Build endpoints

### Step 4 → Add external API integration

### Step 5 → Build MVC UI

### Step 6 → Add AJAX for Country/State/City

### Step 7 → Connect MVC to API

### Step 8 → Add file upload support

### Step 9 → Final Testing

# 🎁 Want me to generate all the CODE files for this?

I can generate:

✔ Models
✔ API Controllers
✔ DTOs
✔ Entity Framework DbContext
✔ Services + Interfaces
✔ MVC Create / Index / Details views
✔ AJAX script for cascading dropdown
✔ Qualification add/remove dynamic script
✔ File upload full code

Just tell me:

👉 "Generate full code for API"
or
👉 "Generate full code for MVC"
or
👉 "Generate entire solution"

And I will generate everything ready to run.