**Design Document for the Contacts API**

**Introduction:**

As asked by the recruiter, I designed an API for doing CRUD operations on Contacts. As part of this task I completed the following tasks:

1. Model (DB + Entity Context) design and development of dbo.Contacts Table.
2. Controller design and development with Asp.Net Core 2.2 Web API.
3. Implementation of following functionality

* Create a contact record
* Retrieve a contact record
* Update a contact record
* Delete a contact record
* Search for a record by email or phone number
* Retrieve all records from the same state or city

1. Manual Testing using Postman.
2. Unit testing using the internal memory database.
3. Deployment to Azure

**High Level Overview ->**

**REST/HTTP Data**

DBContext

ContactsController

Contacts DB

Client sends a REST/HTTP request and then the ASP.Net MVC engine routes the request to appropriate URI with the associated HTTP command and accordingly the action in the contacts controller acts upon the request.

**Low Level Design ->**

ContactsController ->(ContactsController.cs)

Functions:

1. ContactsController(ContactsDBContext context) -> Initialize the controller with a dummy contact info.
2. async Task<ActionResult<IEnumerable<ContactItem>>> GetContactItems() -> Get all Of the ContactItems in the DB at any given time. Works with Http Get Request.
3. async Task<ActionResult<ContactItem>> GetContactItem(long id) -> Get a contact for a specific ID. . Works with Http Get Request.
4. async Task<ActionResult<IEnumerable<ContactItem>>> SearchContactItems(string email, string phone) -> Search for a Contact by its email or phone. Works with Http Get Request.
5. async Task<ActionResult<IEnumerable<ContactItem>>> SearchContactItemsForState(string state) -> Retrieve a contact by its State. Works with Http Get Request
6. async Task<ActionResult<IEnumerable<ContactItem>>> SearchContactItemsForCity(string city) -> Retrieve a contact by its city. Works with Http Get Request.
7. async Task<ActionResult<ContactItem>> PostContactItem([FromForm]ContactItem item,IFormFile profileImage) -> Createsa new contact information from the Form Data. Saves the image to the file system and keep its reference in DB. Works with Post request.
8. async Task<IActionResult> PutContactItem( [FromForm]ContactItem item, long id, IFormFile profileImage) -> Updates an existing record. Works with Http-Put request.
9. async Task<IActionResult> DeleteContactItem(long id) -> Deletes an item by ID. Works with delete request

ContactsDBContext ->

Instantiates the DBContext with EFCore 6 DBSet with set of ContactItems. The primary key Has been Set as ID.

ContactItem ->

Model of each record in DB and in the EfCore Context.

Some of the properties were annotated to make their appearance special.

public long Id { get; set; }

[Required]

public string Name { get; set; }

public string Company { get; set; }

public string ProfileImage { get; set; }

public DateTime BirthDate { get; set; }

[Phone]

public string HomePhoneNumber { get; set; }

[Phone]

public string WorkPhoneNumber { get; set; }

public string Address { get; set; }

public string City { get; set; }

public string State { get; set; }

[EmailAddress]

public string Email { get; set; }

**Assumptions ->**

1. Used SQL Server as a database as nothing was mentioned for Data layer. But if it is a Read-heavy data a No-Sql DB would be a better option.
2. Used the latest ASP.Net core with async functionality to develop the API because of the ability to handle multiple request simultaneously.
3. Used Efcore context directly without using a separate DB layer.
4. No Authentication or authorization in the current version of the Application
5. For phone number and Email did not put any unique constraints in the DB as no explicit mention of it in the requirements.
6. Stored the filename in the DB and the File in the Server for the profileimage. This is done because storing the file could result in corruption of the file.

UnitTesting ->

The Test project was created to unit test the functionality in the controller. Tries with MOQ framework to mock unit tests but found issues with Efcore and Moq. Hence used the InMemoryDatabase of the Entity framework to unit test GetCOntactItems and GetContactItem.