**CandidateInformationAPI Solution Documentation**

**Introduction**

This documentation provides an overview of the **CandidateInformationAPI** solution, which consists of the **CandidateInformationAPI** project and the associated MSTests project for unit testing.

**Project Structure**

**CandidateInformationAPI**

This project contains the main application code for the Candidate Information API.

* **Controllers**: Contains the API controllers responsible for handling HTTP requests.
* **Data**: Contains the database context and migrations for Entity Framework Core.
* **DTOs**: Data Transfer Objects used for communication between the client and the server.
* **Models**: Contains the domain models used by the application.
* **Repositories**: Contains interfaces and implementations for data access.
* **Services**: Contains business logic services used by the controllers.
* **Mapping**: Contains AutoMapper configuration profiles for mapping between DTOs and domain models.
* **appsettings.json**: Configuration file for database connection strings, JWT settings, and other application settings.
* **Program.cs**: Configures services and middleware for the ASP.NET Core application.

**CandidateInformationAPI.MSTests**

This project contains unit tests for the **CandidateInformationAPI** project using MSTest framework.

* **MockData**: Contains mock data and helper methods for setting up test scenarios.
* **TestClasses**: Contains test classes for various components of the **CandidateInformationAPI** project.

**Running the Application**

To run the application locally:

1. Ensure that you have .NET Core SDK installed on your machine.
2. Set up the database connection string in the **appsettings.json** file.
3. Run the database migrations to create the database schema: **dotnet ef database update**.
4. Build and run the application using Visual Studio or the .NET CLI.

**Running Tests**

To run the unit tests:

1. Navigate to the **CandidateInformationAPI.MSTests** project directory.
2. Run the following command: **dotnet test**.

**List of Ways for Improvement**

1. **Enhanced Error Handling**: Implement more robust error handling mechanisms to provide better feedback to clients.
2. **Logging**: Integrate a logging framework such as Serilog to capture application logs for monitoring and debugging purposes.
3. **Performance Optimization**: Identify and optimize performance bottlenecks in the application code to improve response times.
4. **Integration Tests**: Expand test coverage by adding integration tests to validate the interaction between different components of the application.
5. **Swagger Documentation**: Enhance Swagger documentation to provide more comprehensive API documentation for consumers.
6. **CRUD Operations**: Consider adding CRUD (Create, Read, Update, Delete) endpoints for managing candidate information in a more professional and user-friendly way