Instructions

The idea of this test is to gain insight into how you, as a developer, think about problem solving while demonstrating:

- Application of SOLID principles.
- A good understanding of separation of concerns without over-engineering.
- Overall approach to unit testing.

This test and it's content is not a direct reflection of the work that will be undertaken at Bauer Media. It is a technical screen to see how candidates solve problems.

To run this project, you will need Visual Studio 2017 or Visual Studio 2017 Community Edition and the .NET Framework 4.7 Developer Pack installed.

Otherwise, there are no enforced libraries or tools that *have* to be used. You are free to use whatever feels most comfortable.

The Brief

> *You have inherited a legacy restaurant guide application that requires some upgrades to bring it up to date with modern techniques and practices. Please take into account separation of concerns, unit testability and the ease of post deployment database tuning and optimisation. Any UI considerations are a value add, but will also be seen in a positive light.*

1. Taking the legacy code in `HomeController`, refactor the application into a layered architecture using the `Restaurant` domain model object as a base and the provided `AppData` database as a data store. Use any preferred data access/ORM framework. Scope for demonstrating the following:

    - Creation of a data access layer taking into account web security best practices. Feel free to use any appropriate design patterns you feel comfortable implementing.
    - The potential definition of a unit testable business logic layer to sit between your controller and your data layer.
    - The use of dependency injection for better unit testing.
    - Creating a strongly-typed view model.

2. Implement the `Save` method at `/Restaurant/Edit`.

    - As with task 1, show a layered architecture. Refactor as you see fit.
    - In addition to supporting editing of the currently shown fields (i.e. `Name`, `Phone Number`), please add a `Chef` field.
    - The `Name`, `Phone Number` and `Chef` fields must be validated and stored correctly according to rules listed below. Any client side validation should not interfere with compliance to these rules. Please unit test this logic.
    - The database update **must** be implemented using a stored procedure, but can be abstracted in anyway you see fit.

3. Optional bonus points for show-offs: extend the `/Restaurant/Edit`

form to include any other fields in a domain specific way. E.g.

  - Cuisine (a dropdown list).
  - Rating field.
  - Address fields.

Restaurant Name rules

1. The name of the restaurant is mandatory.
2. The name cannot exceed 250 characters.

Chef rules

1. The name of the chef is mandatory.
2. The name of the chef can contain Unicode characters.
3. The name of the chef cannot exceed 500 Unicode characters.

Phone Number rules

1. Phone numbers are assumed to be valid within Australia - they will never be international.
2. The only non-digit characters accepted by the `Phone Number` field are:
    - The whitespace character.
    - The opening and closing parentheses, but only when they enclose the two digit area code.
3. The non-digit characters stated above will be accepted by the UI but ignored/dropped during validation. Phone numbers with other

non-digit characters will be rejected.

4. If less than 9 **significant** digits are entered, the phone number will be rejected.
5. Any digits beyond the 9th most-significant are ignored/dropped during validation.
6. The resulting 9-digit number will be stored in the `PhoneNumber` database column.
7. When displaying the number from the `PhoneNumber` database column, it must be formatted as `(00) 0000 0000`.

Submission

When you have completed the task

- Zip the entire solution.
- Ensure your submission passes the “works on my machine” test.
    - Submissions requiring nuget restore are fine (and preferred).
    - Submissions that fail to run due to missing/non relative database paths will be at a disadvantage.
- Return it to the contact who provided you this test.
- You may not have had time to modify the solution exactly as you would like. If not, include some comments about any technical debt or the architecture you would like to improve upon.