Instructions manual for the project

This project has both the frontend and backend implementation.

The file is zipped archive of the folder which should be placed in a IIS folder or deployed via or via Web Deploy Publishing.

“Web.config” file should be changed to reflect the setting and configuration one might want to have.

Following keys should be configured:

* DbUser
* DbPassword
* DbEndpointUrl
* DatabaseToUse
* and LogFileForSms

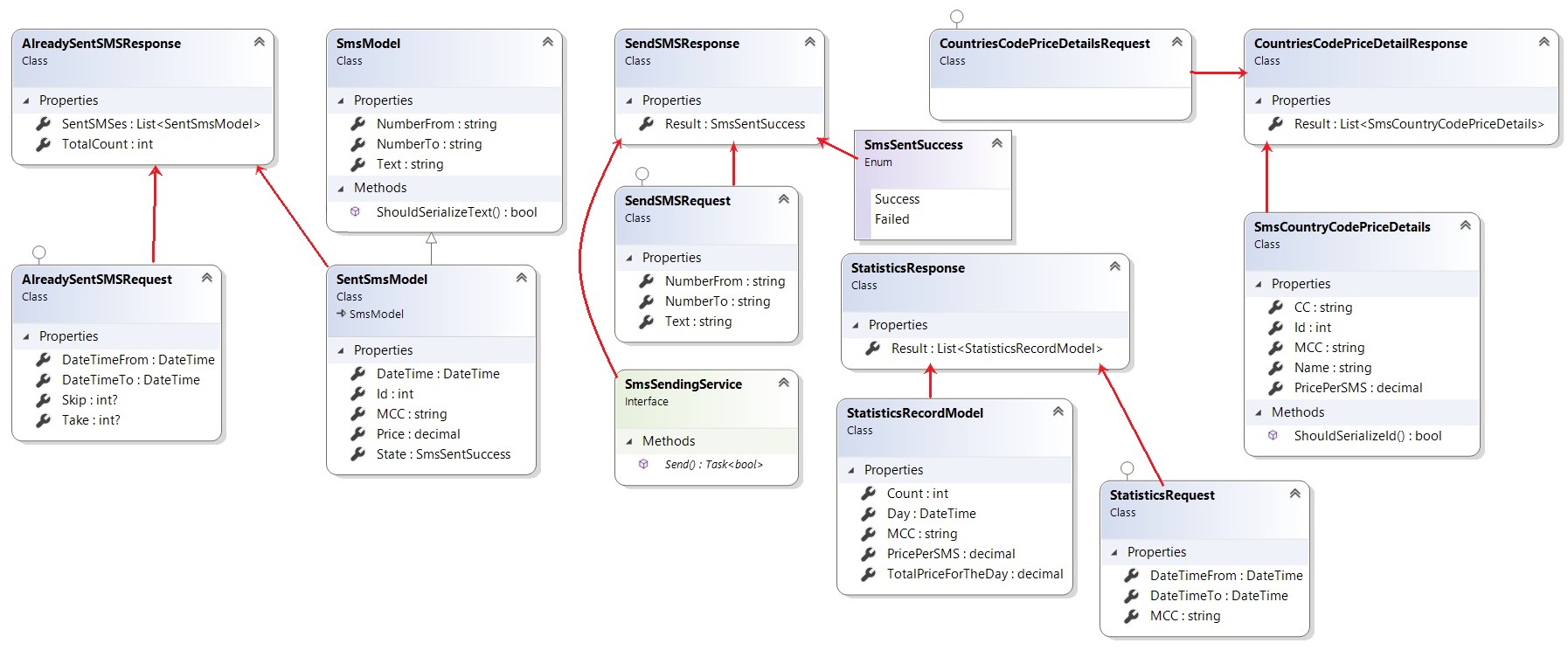
An example of testDb was used for development purposes and shows how should it look like.

You should edit this key to suit your own MySQL instance.

LogFileForSms holds the path to the file where SMSes should be dumped (alongside being recorded to the database).

This should be sufficient for running the project.

Bellow we can see the class diagram. Note that all classes which have xyzResponse name represent the DTO of a class which is returned by the service (i.e. front-end client consumes such a response), where as the abcRequest represents a DTO of a class which is sent to the service (i.e. front-end client sends such a request).



Main difficulties I had were with the async services as I couldn’t get them(services) to be asynchronous, after some time I figured out that asynchornicity lies in the framework it self, ServiceStack v4 (current) was the first one to support it, and and commercial development based on it require a paid licence.

Thankfully this is not a commercial product, just a showcase and as such I was freely able to use the awesomeness of the framework.

Possible optimizations that come to mind are:

* Improve the exception handling. I was mainly either propagating exceptions (re-throwing them) or just swallowing them silently. This should be handled much more effectively and gracefully.  
  ELMAH library comes to mind and is very configurable and easy for use. (Has external tool for searching errors by time, query string, etc.)
* Logging is another eyesore and it is completely missing from action. No statistics nor any kind of logging was used.
* Caching, first thing springing to mind is using Redis server for caching. Another great thing is ServiceStack framework has out of the box support for Redis cache and it is easily configurable.
* Another thing that came to my mind was to rework the model and to introduce the distributed workers which would try to send the message X times before fully stopping. In case 1st or 2nd tries are unsuccessful, the 3rd might be the charm, and after that set the flag to success. Otherwise after 3 or some other configurable value, the workers would stop trying to send the SMS.
* This application doesn’t use authorization rules and as such is very insecure. Depending on the software requirements we could introduce additional layers for security (authorization and authentication – for different roles, 1 for SMS sending, another for statistics, etc.