**Bundles system**

1. **Running application**

Open solution with Visual studio 2015. Press F5 or Debug -> Start Debugging

1. **API**

Application has main three methods:

1. Post (**Customer** customer) – called by “Get bundle” button in GUI. It’s purpose is to return bundle with the highest value by using customer’s answers to the questions and business rules.
2. Post (**MatchViewModel** matchViewModel) – called by “Match” button in GUI. It’s purpose is to check if there is a match (the bundle choice is valid) between given bundle (chosen from select list in GUI), customer’s answers to the questions and business rules.
3. Post (**AddProductViewModel** productViewModel) – there is no GUI for this method. It’s purpose is to add a requested product to the customer’s product list.
4. **Explanation of choices**

The main requirement was to build unit tested Web API. The application is small and there were no requirements for user interface, so it uses jquery-1.10.2 library and plain HTML for user interface. Also bootstrap libraries can be used interchangeably. If there would be a need for complex functionality, such as data binding or heavily reacting to the user’s actions, other libraries, like Angular or React could be used.

There are several places where business rules can be enforced: product rules, bundle rules. Because bundle rules are fully covered by product rules, there was a decision to enforce only product rules. But if the requirements change in the future, application is made in such a way, that business rules of the bundles could easily be added (because of use of inheritance, polymorphism could easily be employed).

Application is made in object oriented manner, using inheritance, polymorphism (for enforcing product rules), design patterns (factories, object injection for more easy writing of unit tests) and unit tests.

Used external libraries:

1. Log4Net for Logging
2. MSTest for Unit testing
3. Jquery for requests to Web API
4. Dapper (fastest micro-ORM) for manipulating data in DB

There are some things that could be done if there were more time dedicated for this:

1. Dependency injection, for example Ninject
2. User authentication
3. Utilization of different languages using resources