**General notes:**

* The main purposes of these tasks are:
* Strong knowledge of typical architecture of enterprise application;
* Deep knowledge of IoC principles;
* Knowledge of native ASP.NET Core container API specifically.

All this knowledge is necessary for passing task interview;

* Task interview will be scheduled after all tasks will be completed and reviewed;
* “Need to know/understand something” phrase means that need to be ready to discuss this theme deeply with mentor;
* This version of the document should be copied to the root of your repository branch which will be used for these tasks. If during tasks implementation some mistakes will be found (or you have some recommendations or tasks improvements) then it should be reflected in the mentioned document’s copy. For this action need to enable change tracking in review tab of MS Word;
* In case if some external and useful tasks related resources will be found then it should be reflected in the mentioned document’s copy.

**Documentation notes:**

* IoC books can be found [here](file:///\\docs.oxagile.com\DEPARTMENTS\HR\TA\Library\dotNet-internal-tasks\IoC\Books\) . “Seemann Mark - Dependency Injection in .NET” is highly recommended for reading;
* IoC videos can be found [here](file:///\\docs.oxagile.com\DEPARTMENTS\HR\TA\Library\dotNet-internal-tasks\IoC\Video);
* Dependency injection documentation in ASP.NET Core can be found [here](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-3.0).

**Theory:**

* Need to know what is the difference between transient, singleton, per web request lifestyles;
* Need to know what is Service Locator and why it should be avoided;
* Besides Service Locator need to know at least 2 IoC anti patterns;
* Need to know what is Martin Fowler’s repository pattern;
* Need to know typical architecture of enterprise application and be ready to discuss strong and weak sides of offered architecture.

**Implementation notes:**

* All external libraries should be referenced as nugget packages;
* MS SQL database should be used;
* ORM should be used for DB access. ORM should be defined by mentor;
* All tasks should be implemented in accordance with [SOLID](http://en.wikipedia.org/wiki/SOLID_%28object-oriented_design%29) principles;
* Hardcode and “magic number” anti-patterns are not allowed;
* Dead code, empty folders, unnecessary commented code are not allowed also;
* The code should be styled in the same manner everywhere.

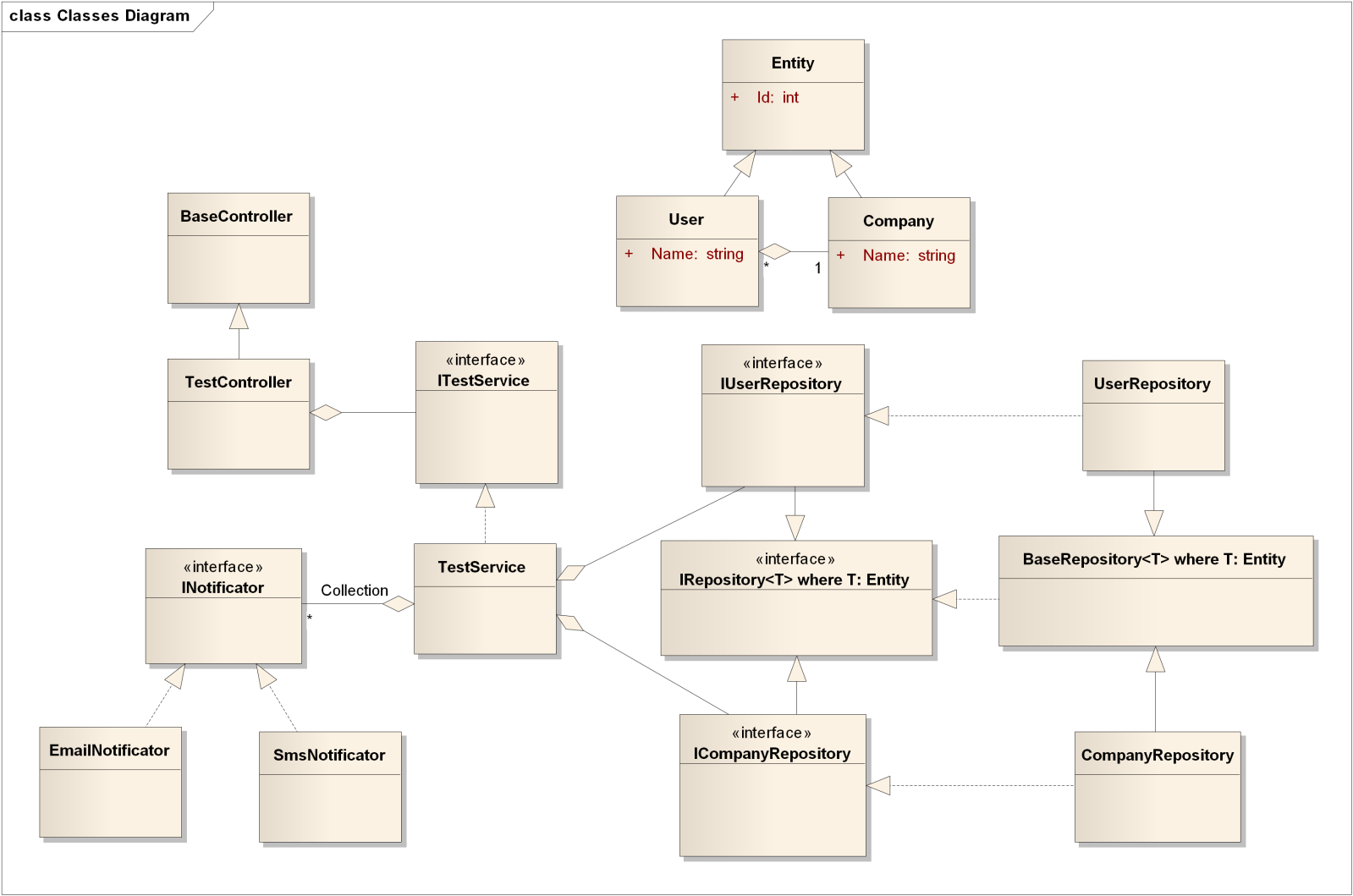


Figure 1.0 Skeleton of typical enterprise application architecture and little bit more

**Tasks:**

* Create simple ASP.NET Core application;
* Create and Register elements from figure 1.0 using native ASP.NET Core container. Will be good if interfaces (contracts) will be places in appropriate and separate assemblies;
* Dependencies should be resolved via constructor injection approach;
* Add support “per request lifetime scope” feature.
* Integrate container with [Serilog](https://serilog.net/) logger;
* Extend the repositories to supporting SQL database via ORM (EF, Dapper and etc.);