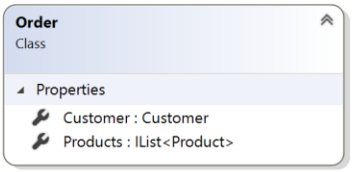
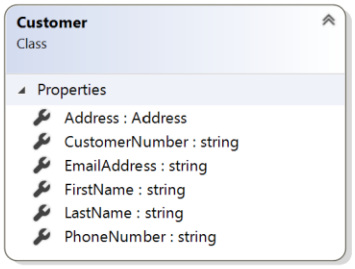
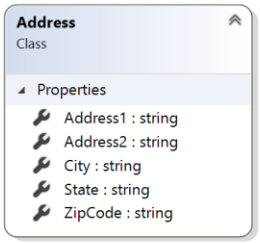
Test Driven Development (TDD) – Class Project

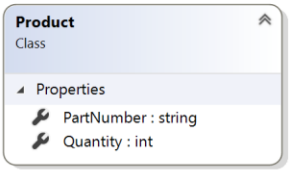
Build an ASP.NET Web API service for processing an order request. For the purposes of this project, there will effectively be only one API endpoint that receives an order request, processes according to business rules defined below and persists data according to defined requirements.

There will be a set of request classes used for input to the API controller action. The order request will consist of the following detail:

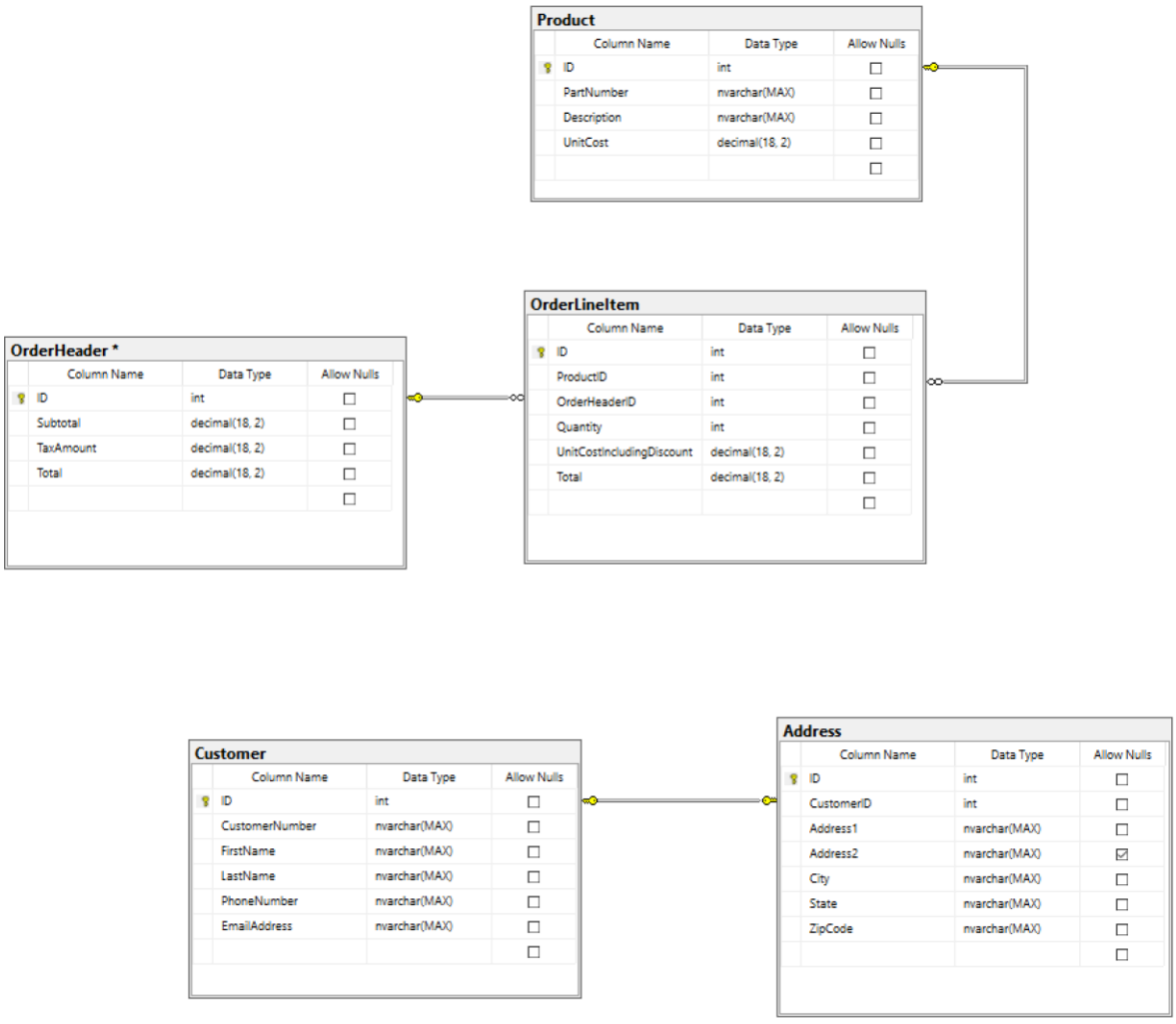








There will be a separate set of classes used to define the domain/data access models leveraged in downstream processing and storage. Reference the following database diagram as a representation of the target models:



The incoming order will need to be processed through the following workflow steps (in sequence) before being persisted to the back-end data store:

1. Translate request objects to domain models for business and data services processing
2. Customer lookup/updates
    a. If customer number is included (order submitted for existing customer)
        i. Update the existing customer record in storage with info provided on the request
        ii. Update the existing address record in storage for the customer with info provided on the request; assume that if a customer is found that an address exists for the customer
    b. If customer number is not included (order submitted for new customer)
        i. Create a new customer record in storage
        ii. Create a new address record in storage associated to the customer
3. Product lookup and order line item creation
    a. Lookup each product by part number and use the located record to build out the order line item
    b. Leverage an internal business rule that determines discounted rate (discount is calculated separately per line item)
        i. If quantity ordered is 500 or greater, apply a 20% discount
        ii. If quantity ordered is less than 500, no discount
        iii. EXTRA (if time permits): Regardless of quantity ordered, if customer has 5 previously existing orders in the system, apply a 10% discount; there is currently no requirement to account for status of those orders – the rule is based on the presence of 5 previously existing orders in any status
    c. Calculate line item cost as quantity * unit cost * (1.0 – discount)
4. Populate and store order header
    a. Calculate subtotal as total of all order line item costs
    b. Calculate tax using 7.5% flat rate
    c. Calculate order total as subtotal + tax
5. Populate and store the collection of order line items (associated to the order header)

For the data sources, you can choose to use either in-memory collections, Entity FW/SQL Server or the “in memory” EF provider – the choice of data storage mechanism is not as important at this juncture. Whichever data source you choose to go with, try to encapsulate functionality in such a way that you can swap in a different data source in the future with minimal impact.

Key focus for this lab will be the creation of a fully tested, fully SOLID-compliant system that meets the defined requirements. Included in the lab is the requirement to ensure that mocking is used to confirm that all key branches of the logic are executed and can be validated in isolation. Use “Red, Green, Refactor” as you go and exercise “test early/test often” as you proceed through your development.