The project implements majority of the requirements. The use cases not implemented include ordering multiple boxes, retrieving revenue based on time periods, search, and wine rank. Both command line interface and REST service have been implemented successfully.

The project took approximately 50 hours to code and about 10 hours preparing the use cases, UML diagrams, and other deliverables.

The biggest challenge has been structuring the code so that it meets all the requirements. The project was about 70% complete before the API was made available but when attempting to make the code work with the API the code became convoluted and unmanageable. Thus the project was started from the beginning while treating the API as the requirements that needed to be met. Most of the original code was reused but the organization has changed.

Other challenges included determining how to deal with delivery dates and times. In the end that was simplified to only days of the week and morning or evening options. Implementing auto generated shipments was also challenging because of uncertainty as to when these shipments should be generated.

The following are statistics about the code.
Lines of code: 1184
Lines of unit tests code: 476
Lines of web interface code: 140
Junit coverage:

Element	Class, %	Method, %	Line, %
action	100% (3/3)	100% (37/37)	87% (280/319)
cli	0% (0/1)	0% (0/2)	0% (0/151)
data	100% (7/7)	70% (19/27)	65% (43/66)
model	100% (12/12)	89% (116/129)	87% (228/262)
request	100% (6/6)	50% (20/40)	50% (40/80)
response	100% (20/20)	32% (49/152)	40% (125/306)
rest	0% (0/3)	0% (0/32)	0% (0/140)

The cli and rest packages have zero coverage because they are wrappers for the action package. The low coverage in data, request, and response packages is due to getters and setters that are only used by Jackson mapper. The Jackson mapper uses the getters and setters to map objects to JSON but these methods are not used anywhere else in the code and thus are not tested.

Cyclomatic complexity:

Class	Average Cyclocomplexity
edu.iit.cs445.vin.action.AdminAction	2.22
edu.iit.cs445.vin.action.AdminActionTest	1
edu.iit.cs445.vin.action.DeliveryAction	3
edu.iit.cs445.vin.action.DeliveryActionTest	1
edu.iit.cs445.vin.action.SubscriberAction	3.29
edu.iit.cs445.vin.action.SubscriberActionTest	1
edu.iit.cs445.vin.cli.CLIMain	17
edu.iit.cs445.vin.data.AdminData	1
edu.iit.cs445.vin.data.Data	1
edu.iit.cs445.vin.data.DataLoader	1
edu.iit.cs445.vin.data.DataLoaderTest	1
edu.iit.cs445.vin.data.IdGenerator	1
edu.iit.cs445.vin.data.MonthlySelectionData	1
edu.iit.cs445.vin.data.ReceiptData	1
edu.iit.cs445.vin.data.SubscriberData	1
edu.iit.cs445.vin.rest.AdminService	1
edu.iit.cs445.vin.rest.DeliveryService	1
edu.iit.cs445.vin.rest.SubscriberService	1

Overall cyclomatic complexity is low because each method is very simple in terms of its function, only exception is the CLIMain class which contains one method that is a long if-else to parse the command line.