Status of Project:

This project is complete. It has been fully-written and complies with the requirements stated in the posted API. It has been thoroughly tested, both through unit tests as well as with human interaction, and after much revision, the product is compliant with the API requirements.

Additional work could be put toward further unit testing. The various reports generated with respect to requirements in the API were challenging to test, so their tests are not as robust as the rest of the back-end and nor as robust as the unit tests for the API code itself.

Code Repository: https://github.com/mseryn/CS445_final_project

Statistics:

Unit test coverage:

Using py.test and pytest –cov

Name	Statements Covered	Statements Missed	Coverage
Delectable/customer.py	51	0	100
Delectable/item.py	67	0	100
Delectable/menu.py	54	1	98
Delectable/order.py	119	6	95
DelectableREST/delectablerest.py	228	26	89
DelectableReport/daily_today.py	12	0	100
DelectableReport/daily_tomorrow.py	12	0	100
DelectableReport/deliveries.py	19	2	89
DelectableReport/report.py	16	8	50
DelectableReport/revenue.py	19	9	53
DelectableReport/undelivered.py	14	3	79
TOTAL	**611**	**55**	**91%**

Cyclomatic complexity:

Using radon on code written in Python 3.5

Average overall complexity: 2.57
Highest complexity: 11

Lines of code: 736

Lines of unit test code: 709

Hours to get code working: 10

Hours spent preparing submission: 4

Challenges:

- *Challenge:* Writing unit tests to test a web framework
    - The techniques needed to unit test our JSON-formatted REST interface were very new to me. For the framework, Flask, I used, the methods for interacting with the web client strictly in code were note clear.
    - *Solution:* First, I read a lot about Flask. I then spent a lot of time in trial-and-error tests to determine what information I actually had access to and how it was formatted. Eventually, I built fairly robust and extensive tests to try every API method from several angles. It both illuminated a lot of problems in my code and it gave me a much better understanding of and appreciation for web development using Python.
- *Challenge:* Git's built-in commands went haywire and caused a scare
    - At some point, Git lost track of a bunch of my files, and it ended up making copies that were pushed (at some point) to the remote. I didn't notice this until much further into development.
    - *Solution:* At first, I only noticed the old files. After panicking and thinking I lost a lot of work, I found the point where the mistake happened and just deleted the old files. I learned an important lesson, though – it's useful to not only commit often, but to also push and re-pull now and then to ensure nothing has happened over time.
- *Challenge:* Keeping dependency trees fairly linear and avoiding loops
    - With a project of this scale, I found it challenging to keep track of all the classes and their capabilities at once. This led to several points where I started to implement a functionality that ruined a tidy dependency tree and led to loops or overly-tied dependencies between classes.
    - *Solution:* I wrote down informal class diagrams. A lot. I used white boards, scrap paper, txt files – anything to keep track of the growth of the classes. Even as I write this memo, there are scribbled but accurate class diagrams and use cases piled on the desk.