

University of Alaska Anchorage

CSCE A470  
Capstone Project

# Android Textbook Marketplace Application Server Design

Author:

**Gabriel Esposito**

Supervisor:

**Dr. Kenrick Mock**

Anchorage AK, April 2016



Computer Science &  
Engineering Department  
UNIVERSITY *of* ALASKA ANCHORAGE

© Copyright 2016  
by  
Gabriel Esposito

Version 1.2

[espogabe@gmail.com](mailto:espogabe@gmail.com)

## **Abstract**

The target of this project is to create a mobile application that will help students save money on textbooks. With the price of textbooks increasing, many students will spend more than expected on materials for schooling. This project hopes to provide an alternative for those students by creating a marketplace centered on locally buying and selling at their university. It employs a client-server architecture running on the Android operating system. My work on the project focuses on the server development, design, testing and architecture in Python. The server should be easy to implement for the client application and hassle-free for the user.

## **Acknowledgements**

I would like to thank Dr. Kenrick Mock for his advice and assistance with this project, including providing a test server for us. Thanks to Aaron Zhao for working with me on this project and Dr. Adriano Cavalcanti for the opportunity to create it.

## Table of Contents

<b>Chapter 1: Introduction</b> .....	<b>8</b>
1.1 Introduction .....	8
1.2 Application .....	9
1.3 Motivation .....	10
1.4 Recent Developments.....	12
<b>Chapter 2: System Integration and Modeling</b> .....	<b>13</b>
2.1 Technologies and Libraries .....	13
2.2 RESTful API Design .....	14
2.3 Client Authentication .....	15
2.4 MySQL Database Wrapper .....	16
2.5 Timeline and Methodology .....	16
<b>Chapter 3: Testing and User Interface</b> .....	<b>18</b>
3.1 User Interface .....	18
3.2 Testing .....	18
3.2a Client Testing .....	19
3.2b Server Testing.....	20
3.3 Agile Methodology .....	22
<b>Chapter 4: Results and Discussion</b> .....	<b>23</b>
4.1 Summary .....	23
4.2 Final Database Schema .....	24
4.3 Software Configuration .....	25
4.4 Results .....	26
<b>Chapter 5: Summary and Conclusion</b> .....	<b>27</b>
5.1 Summary .....	27
5.2 Implications .....	27
5.3 Recommendations .....	28
5.4 Conclusions .....	28

**References ..... 29**  
**Appendix A – Initial UML Diagram ..... 30**  
**Appendix B – Initial Source Code ..... 31**

## Figures

<b>1.1 College textbook price increases</b> .....	<b>9</b>
<b>1.2 Languages and technologies in use</b> .....	<b>10</b>
<b>1.3 Increasing bookstore prices</b> .....	<b>11</b>
<b>1.4 Rental from Amazon</b> .....	<b>11</b>
<b>1.5 Boundless online book rental</b> .....	<b>12</b>
<b>2.1 RESTful API architecture</b> .....	<b>14</b>
<b>2.2 Client authentication with OAuth</b> .....	<b>15</b>
<b>2.3 JSON structure</b> .....	<b>16</b>
<b>2.4 SQL injection avoidance</b> .....	<b>16</b>
<b>2.5 Proposed Gantt chart</b> .....	<b>17</b>
<b>3.1 Network test app</b> .....	<b>19</b>
<b>3.2 Test output</b> .....	<b>20</b>
<b>3.3 Server logging</b> .....	<b>21</b>
<b>3.4 Test script</b> .....	<b>22</b>
<b>3.5 Agile methodology</b> .....	<b>22</b>
<b>4.1 Database schema</b> .....	<b>24</b>
<b>4.2 PuTTY</b> .....	<b>25</b>
<b>4.3 Configuration creation</b> .....	<b>26</b>
<b>4.4 Unauthorized API call</b> .....	<b>26</b>
<b>5.1 HTTPS snippet</b> .....	<b>28</b>

# **Chapter 1**

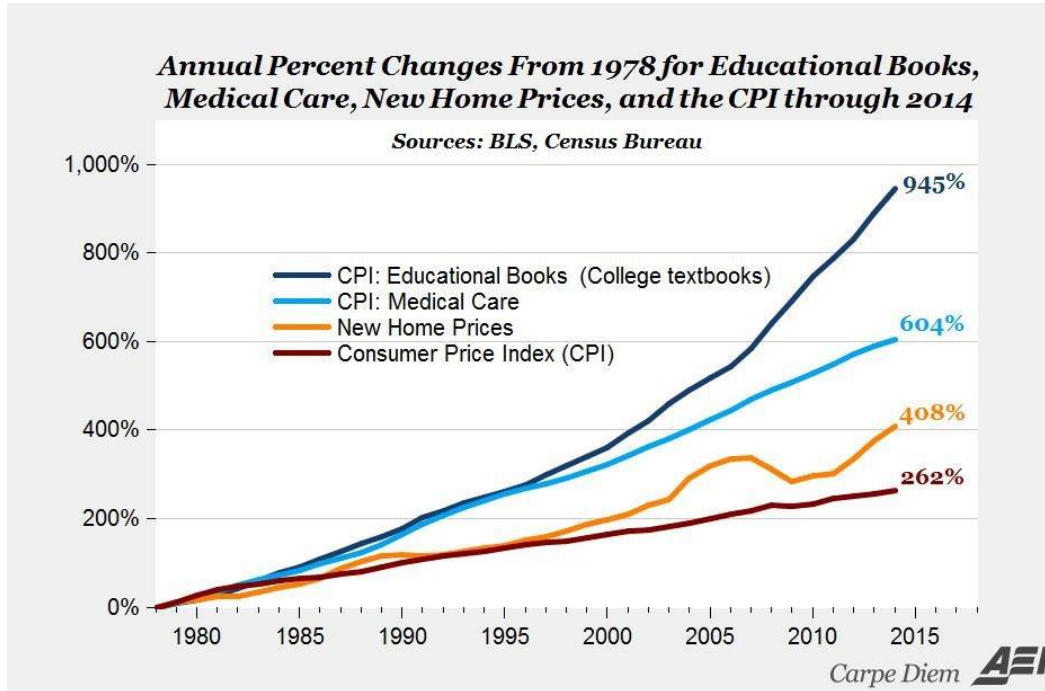
## **Introduction**

---

### **1.1 Introduction**

In recent years, the cost of textbooks for students attending higher education has skyrocketed (Figure 1.1) [1]. Students wishing to buy materials for university classes are often required to spend hundreds of dollars per year on textbooks that may or may not be useful to them after the class is over. In order to provide a solution for the aforementioned problem, we are proposing an Android application to provide a marketplace where students can buy and sell textbooks locally with other students. This application can be of use to college attendees who wish to save money without dealing with shipping or local bookstore prices. It aims to be easy to use and safe for students to sign up for without risking their personal information.

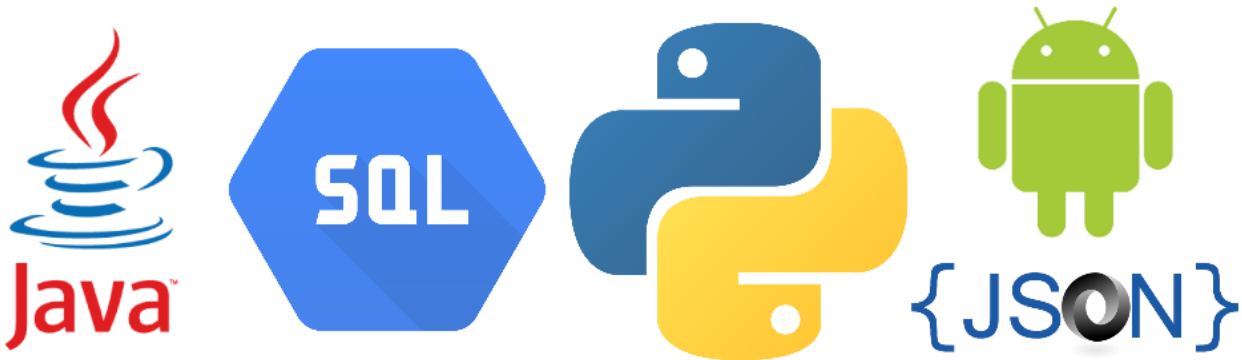




**Figure 1.1:** College textbook price increases from 1978 – 2015 compared to other increases like the price of a new home. [1]

## 1.2 Application

This project has two parts, and this document will be focusing mainly on the backend server implementation of the Textbook marketplace application for technical details. The application will be developed using two main languages, Java and Python. The Android application will be written in Java, while the server will be written in Python, running on the Ubuntu distribution of Linux. In the application, the user will have the ability to both buy and sell textbooks. Listings will be posted on a per-university basis, and users will be able to browse universities near them using location services. The server will have a web server running in Python, able to handle multiple clients and send responses formatted in JSON back (Figure 1.2). Listings will be stored on a database, most likely MySQL or SQLite. User identification will be done by unique device ID and accounts therefore associated with devices to streamline the user experience. A hash of the device ID will be stored server-side to provide interaction with users through sessions if necessary, and to associate users with their listings. In order to facilitate the listing of textbooks, we will be using Google’s Books API [2] to auto-fill information by ISBN. Testing for the Android application will be done on device emulators, and any other physical devices we own (Nexus 6P and Nexus 7). We will tentatively be using the MIT License for our project.



**Figure 1.2:** Programming languages and technologies that will be used to implement this project.

### **1.3 Motivation**

The average textbook can now cost around \$200 to \$300 in local university bookstores [3], depending on the subject, which can be a significant amount to spend for a student who may already be in debt due to student loans. This project aims to combat these statistics by developing an easy to use secondhand textbook marketplace as an Android smartphone application. In doing so, we can take advantage of the characteristics of smartphones and tablets in order to streamline the user experience. Features like per-device password less sign-on and location-based university selection become possible. Essentially, this is a way to improve students' lives, and an outlet for them to discover textbooks at cheaper prices for the classes they need them for.

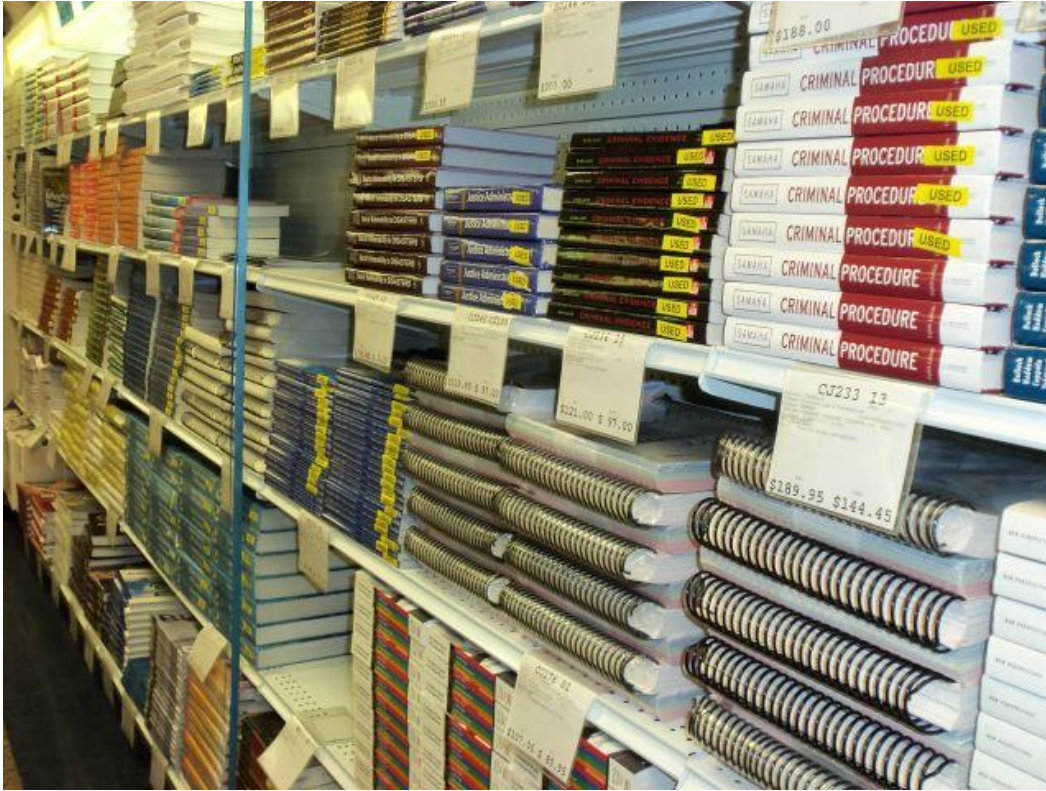


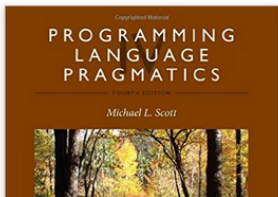
Figure 1.3: Increasing prices in bookstores are leading students to find other methods to acquire books online.

### Programming Language Pragmatics, Fourth Edition 4th Edition

by Michael L. Scott (Author)

Be the first to review this item

Look inside



**Paperback**  
\$75.32 - \$85.45

Other Sellers  
from \$75.20

Buy used

\$75.32

Buy new

**\$85.45**

In Stock.

Ships from and sold by Amazon.com. Gift-wrap available.

List Price: \$89.95 Save: \$4.50 (5%)

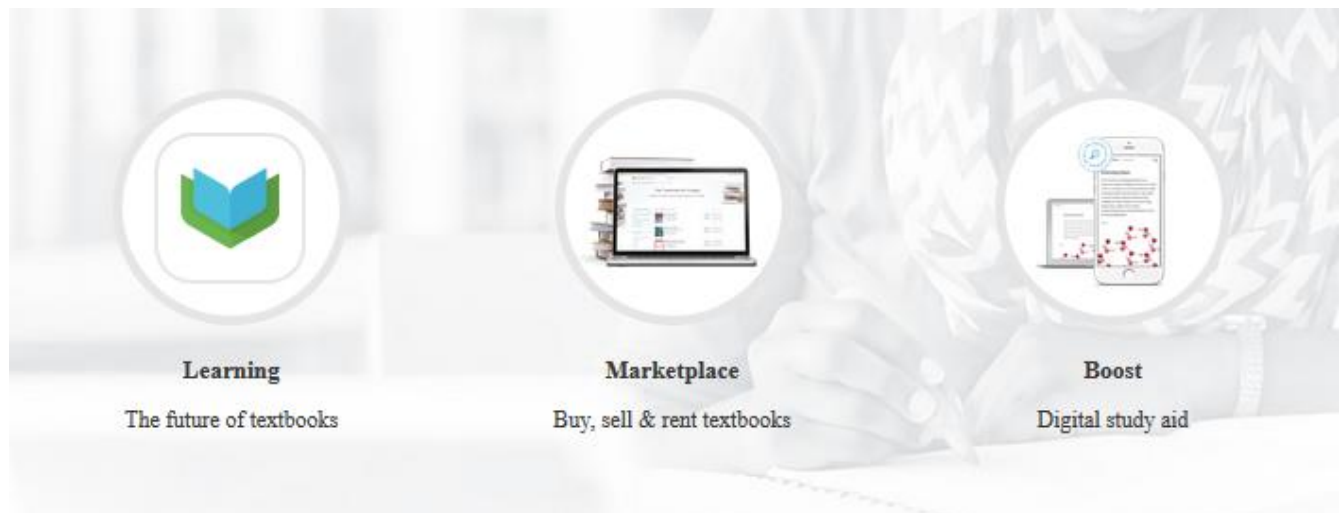
29 New from \$75.20

Figure 1.4: Not all books are available to be rented from Amazon.

## 1.4 Recent Developments

It has already been stated that the price of textbooks has been on the rise for some time now. Many solutions for offering discount solutions to students have already appeared, with varying success. This idea is not a new one, but there are several caveats that existing solutions have. Amazon's textbook rental service [4] is a great solution, but it requires time to ship the textbook, and it is a rental service, which means the student will have to ship it back. Also, the service is only available for some books, and if there is a problem with shipping back the student will be charged the whole price. Other options like Craigslist provide local buying options, but listings can be inconsistent and lacking in information. Still other options include Boundless.com (Figure 1.5) [5], which not only provide a marketplace for buying and selling, but are pushing to open-source educational materials. The problem with this, however, is adoption at universities, which would need to lose profits from textbook sales in order to implement.

Our solution allows for textbooks and potentially other resources (clickers, etc) to be listed and sold locally, with assurance they will work for the classes, if any, which are associated with the listing.



**Figure 1.5:** Boundless.com provides a marketplace for textbooks, but their mission to replace textbooks has yet to appear in many schools.

## **Chapter 2**

# **System Integration and Modeling**

---

### **2.1 Technologies and Libraries**

The different technologies in use must all come together in harmony to produce a working final product. The Android application will be talking to the server via HTTP requests, so a web server framework is needed. For this, I chose Google's webapp2 Python framework [6]. It can run an HTTP server and handle POST and GET requests. We won't be using Google's App Engine cloud platform, but webapp2 is still available for use outside of it.

When we proposed the application to our supervisor, Dr. Mock, he was able to get us access to a server at UAA to run on. On this server, he installed a MySQL instance. In order to interface with it to store data, I used the PyMySQL module [7] and constructed a wrapper for the server to use. The server design implemented is a REST-style (REpresentational State Transfer) HTTP server API with database access via POST and GET requests, and is detailed in the next section.

## 2.2 RESTful API Design

It is important to adhere to standards when designing software, and Application Programming Interfaces or APIs are no different. By creating and documenting an API for the server, it can be easily accessed by the clients and expanded upon in the future. Arguably the most important part of this type of API is its statelessness. Therefore, it will use the REST API guidelines where necessary (Figure 2.1). The critical part is this - in one request, the server or client will have all the information needed to understand the request and return a response if necessary [8]. This way, the implementation is fairly simple, and the groundwork is already there in the form of a webserver. The data transferred from server to client will be in JSON – JavaScript Object Notation – format (Figure 2.3). This is also a standard and is commonly used with RESTful APIs.

### Clients

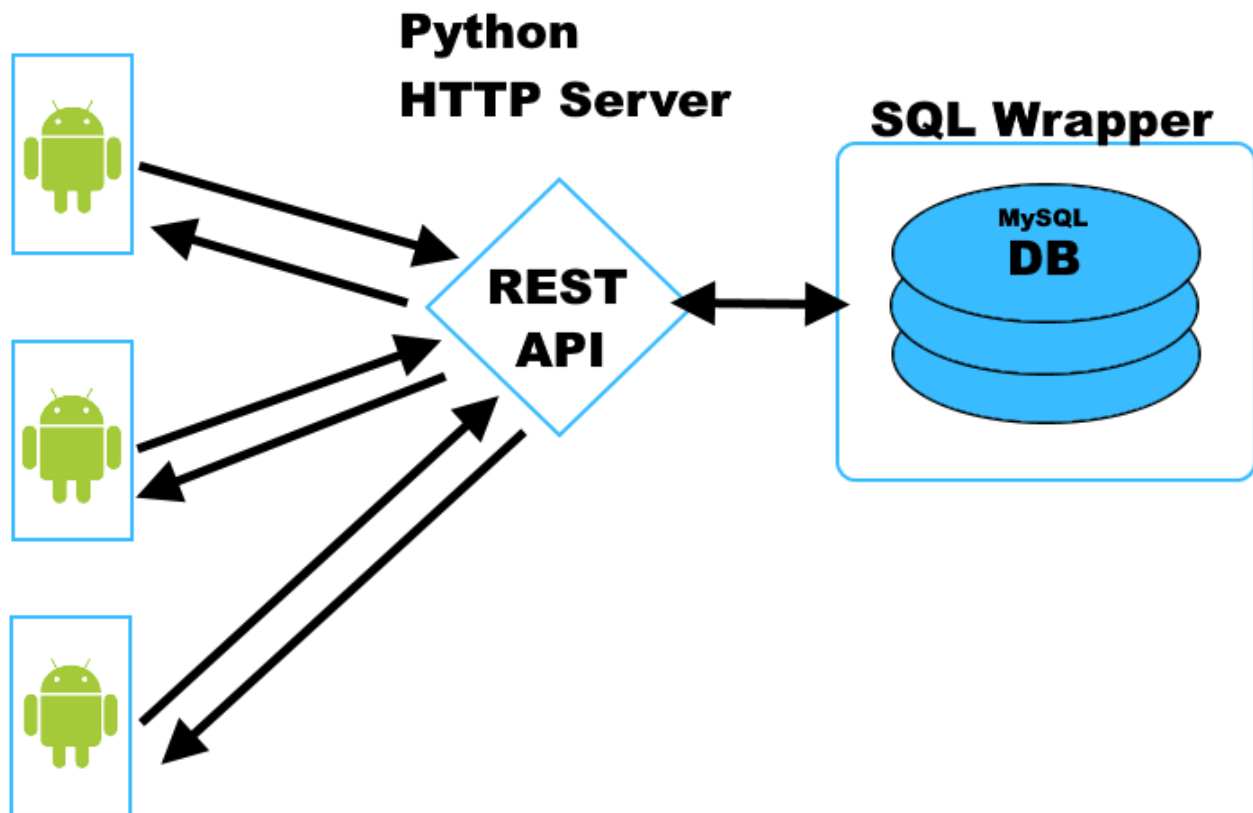
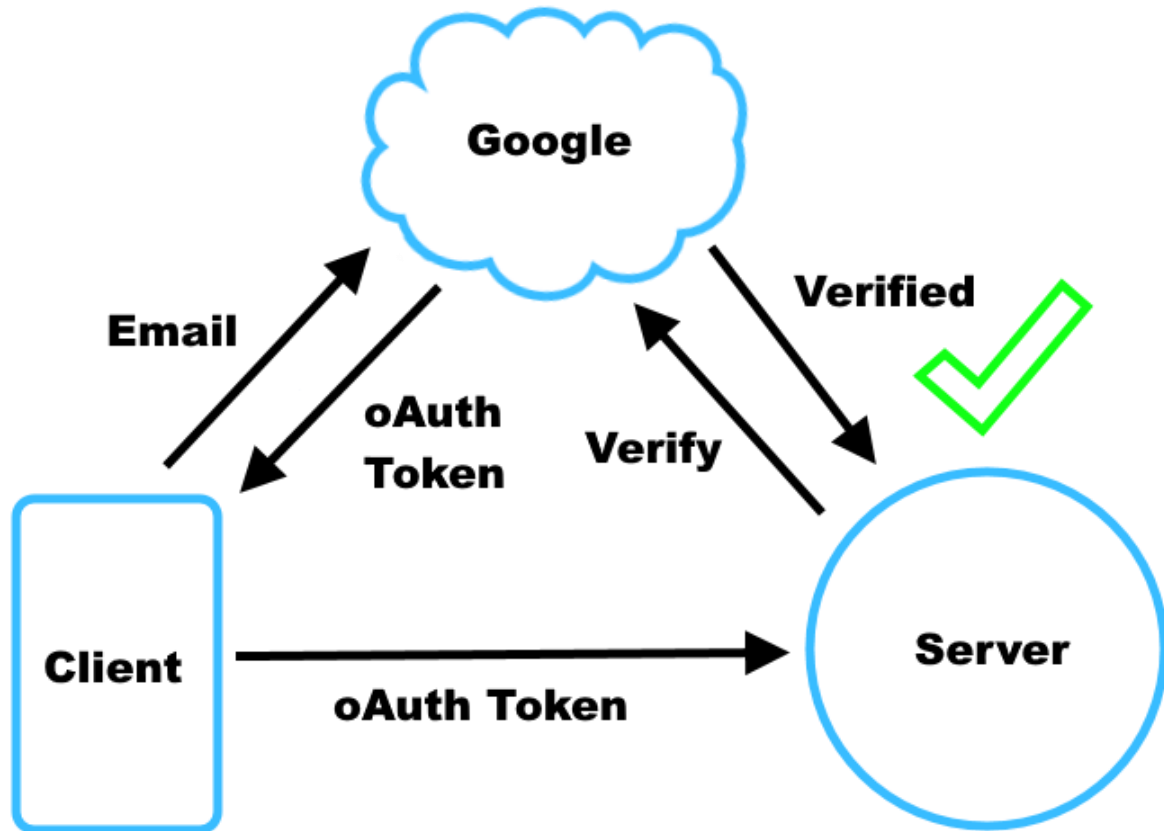


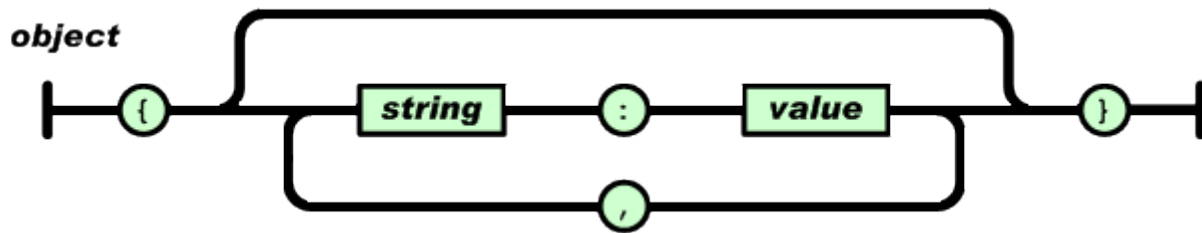
Figure 2.1: Proposed RESTful API implementation.

## 2.3 Client Authentication



**Figure 2.2:** How oAuth helps to authenticate clients.

In order to avoid abuse or attacks on the server, we felt it necessary to implement some form of client authentication. Preferably, all requests to the server would originate from our application. In this case, verification with a username and password would usually be necessary. In our project, however, we didn't want remembering credentials to impact our user experience. So we needed a solution, and one that would work with a RESTful API. Luckily, Google provides an authentication method to do just this with oAuth. Our application will use oAuth to get a token from Google for a registered application. The client app will be compiled with our server's cert inside. The user will, upon opening the app, select an email to link to our application. Then, whenever a call to the server is made, the token will be sent to the server and verified with Google's public key (Figure 2.2) [9]. This provides an easy-to-use sign-in interface for the user and secure authentication for our app at the same time.



**Figure 2.3:** JSON's data structure is easily human-readable as well as simple for machines to parse.  
[10]

## 2.4 MySQL Database Wrapper

The final piece of the server puzzle is how to communicate with the MySQL database. My approach was one that provided a lot of flexibility – I templated some common SQL commands and defined these as the standard for what actions should be allowed over REST. I designed a wrapper class for PyMySQL called RESTWrapper that is easy to use and keeps clutter out of code typically caused by SQL integration. It's also extremely important when accepting strings sent by clients (even if they are authenticated) to sanitize inputs. The first thing I did when creating the wrapper was sanitize the inputs (Figure 2.4). The main server imports and uses the wrapper to get SQL results in just a few lines.

```
def scrub(string):
    """Avoid SQL injection attacks."""
    return ''.join([char for char in string if char.isalnum()])
```

**Figure 2.4:** Securing SQL servers against injection attacks is important whenever inputs can be modified.

## 2.5 Timeline and Methodology

This project is an effort between myself and another CS student, and we hope to seamlessly integrate our projects to work together. The timeline in Figure 2.5 shows that we hope to be testing and debugging as soon as all the features are finished in April. To assist with this, the methodology used for the project will be an agile methodology. As we work, we share progress and show work to each other as we complete it. We will also be meeting with our advisor periodically to showcase our progress and application in this manner.



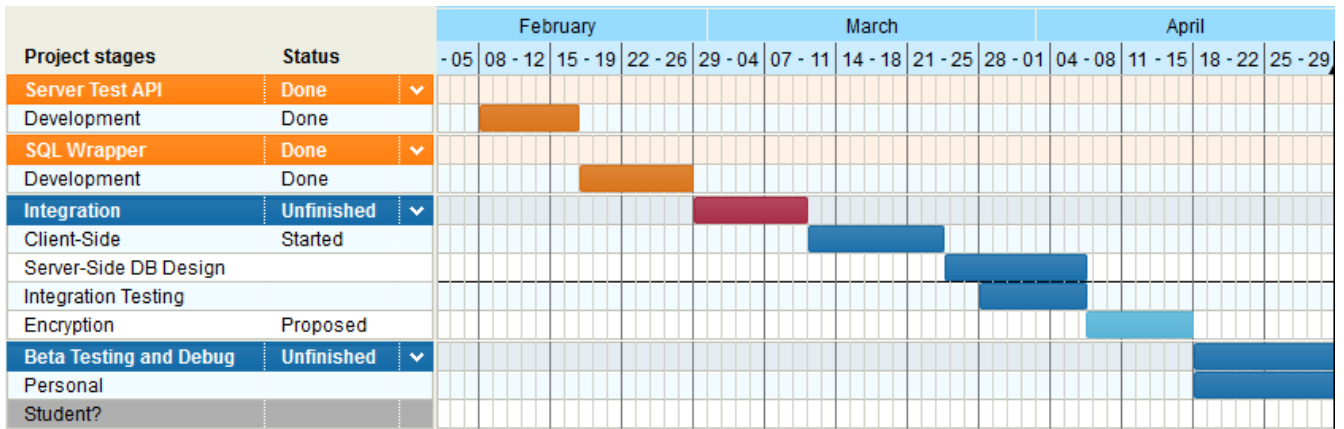


Figure 2.5: Proposed Gantt chart for the project.

## **Chapter 3**

# **Testing and User Interface**

---

### **3.1 User Interface**

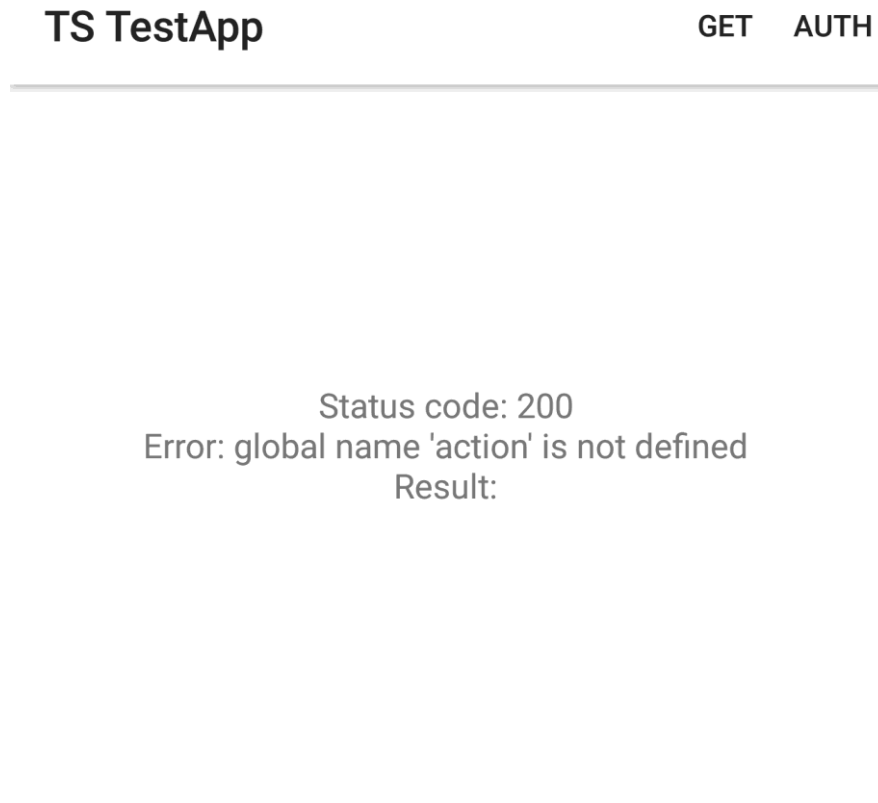
Because this is a server application, there really is not much of a user interface to speak of. It is meant to be configured and then run 24/7. The most that could be classified as an interface would be the API or the configuration file, but even the former isn't seen by the user as it's obfuscated by the client app. So, then, the real in-depth conversation will be on the companion document to this one by my partner Aaron Zhao, as he designed the user interface for the client application.

### **3.2 Testing**

Testing is an extremely important part of any project, and it's very important here. Since there are so many parts that need to line up and work properly, it's essential to test constantly during development and well after. When developing a server, there is somewhat of a catch-22 situation where you need to test, but you don't have the client finished yet. So, I took it upon myself to rapid prototype some Android net code to test the server out [11]. This section will explain how I tested the client and server in lieu of a proper client.

## 3.2a Client Testing

In order to get feedback about development without a client app, I had to come up with an improvised way to establish communication with my server from an Android device. I ended up writing some network code to make a HTTP GET request on our server (Figure 3.1) [12].



**Figure 3.1:** The barebones network test app.

This app turned out to be invaluable in not only testing the server but also debugging possible issues Android network code can run into. For example, when the server returns a 401 Unauthorized, certain implementations of network code will raise a `FileNotFoundException` [13]. This was an issue as the server returns a 401 if the token is incorrect.

## 3.2b Server Testing

The server code implements some tests to ensure the SQL wrapper properly works. It attempts to do an insert with varying numbers of parameters on a test database table. If this succeeds, it will perform select statements that will return the input values. The current version of the script uses assert statements to test if the returns from the wrapper methods are correct (Figure 3.2) [14].

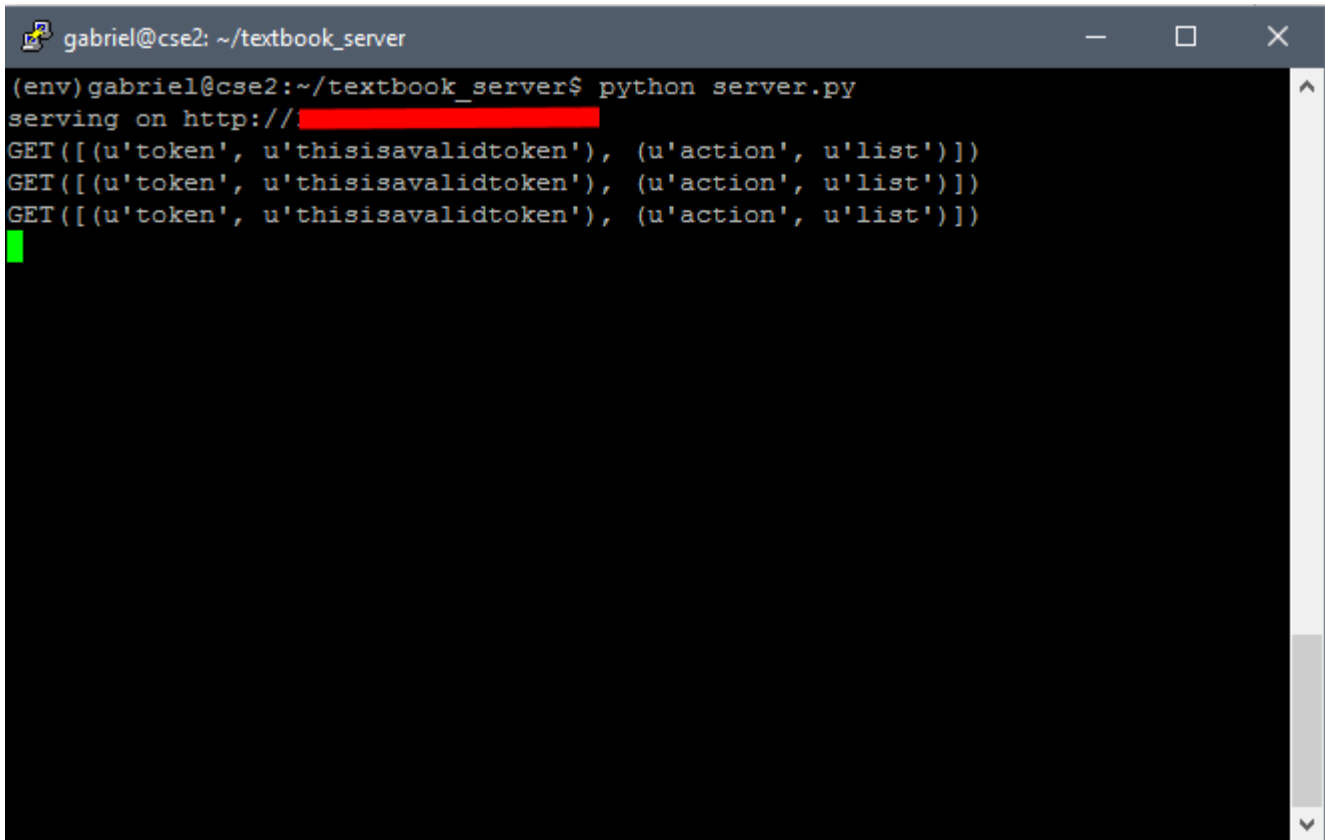
```

gabriel@cse2: ~/textbook_server
(env)gabriel@cse2:~/textbook_server$ python test.py
Testing wrapper insert
Test succeeded
Testing wrapper select
Traceback (most recent call last):
  File "test.py", line 13, in <module>
    assert type(d) is DictType, "d is not a dict: %r" % d
AssertionError: d is not a dict: [{u'this': u'valuepy2', u'testing': u'valuepy12
(env)gabriel@cse2:~/textbook_server$ nano test.py
(env)gabriel@cse2:~/textbook_server$ python test.py
Testing wrapper insert
Traceback (most recent call last):
  File "test.py", line 7, in <module>
    assert True == wr.insert('test', testing='valuepy123', this='valuepy2')
  File "/home/gabriel/textbook_server/sqlwrapper.py", line 31, in insert
    cursor.execute(query)
  File "/home/gabriel/textbook_server/env/local/lib/python2.7/site-packages/pymy
sql/cursors.py", line 146, in execute
    result = self._query(query)
  File "/home/gabriel/textbook_server/env/local/lib/python2.7/site-packages/pymy
sql/cursors.py", line 296, in _query
    conn.query(q)
  File "/home/gabriel/textbook_server/env/local/lib/python2.7/site-packages/pymy
sql/connections.py", line 819, in query

```

**Figure 3.2:** The output of the test script, part of which can be seen in Figure 3.4. The highlighted area shows an AssertionError.

It was important to also add debug statements to the server in order to observe the data being sent by the client application (Figure 3.3). A lot of development was done with quick edits in GNU Nano over SSH due to the nature of the server.

A terminal window titled 'gabriel@cse2: ~/textbook\_server' showing the output of a Python server script. The terminal displays the command '(env) gabriel@cse2:~/textbook\_server\$ python server.py' followed by 'serving on http://[redacted]'. Three consecutive GET requests are shown, each with parameters: 'GET [(u'token', u'thisisavalidtoken'), (u'action', u'list')]'. A green cursor is visible on the line following the third request.

```
(env) gabriel@cse2:~/textbook_server$ python server.py
serving on http://[redacted]
GET [(u'token', u'thisisavalidtoken'), (u'action', u'list')]
GET [(u'token', u'thisisavalidtoken'), (u'action', u'list')]
GET [(u'token', u'thisisavalidtoken'), (u'action', u'list')]
█
```

Figure 3.3: HTTP GET requests coming from the test app.

### 3.3 Agile Methodology

Adopting the agile methodology helped push our project along especially with proper tests established before code was written. Being able to run tests against new code pushed my development along faster than code-first. It also influenced my coding style to be more modular in nature for the whole project. In terms of project management, it's a useful style but in coding it helps to be concise and clearer about what you are writing [15].

```
GNU nano 2.2.6 File: test.py
from types import *
from sqlwrapper import RESTWrapper

print 'Testing wrapper insert'
wr = RESTWrapper('localhost', '████████', '████████', '████████')
assert True == wr.insert('test', testing='valuepy12345', this='valuepy2')
print 'Test succeeded'

print 'Testing wrapper select'
wr = RESTWrapper('localhost', '████████', '████████', '████████')
l = wr.select_all('test', testing='valuepy12345')
assert type(l) is ListType, "l is not a list: %r" % l
print 'Test succeeded'
```

Figure 3.4: Part of the test script on the server application.

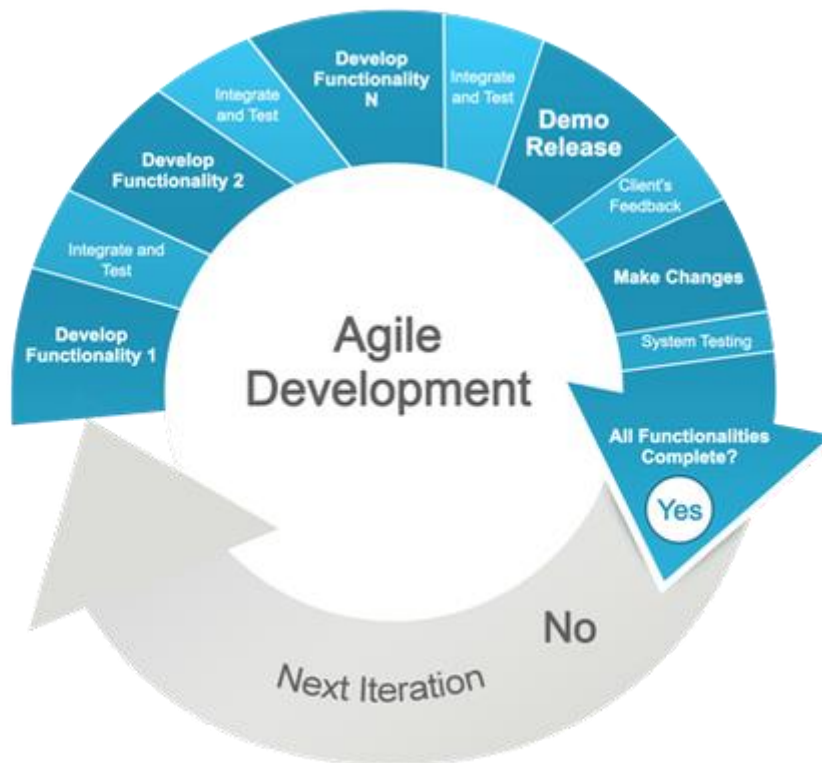


Figure 3.5: Agile development in action.

## **Chapter 4**

# **Results and Discussion**

---

### **4.1 Summary**

The Android textbook marketplace app is a project created to facilitate the buying and selling of textbooks between students. In this project, I was responsible for the development of the server which would store sales and interact with the Android app. I also wrote the basic functions for the app's network code for communicating with the server.

This server software was created to be easy to understand and extendable. Some functions would most likely be useful outside of this software, such as the MySQL Python wrapper. In this chapter, the final database schema will be presented and the configuration of the server will be explained.

### **4.2 Final Database Schema**

```
mysql> describe Sale;
+-----+-----+-----+-----+-----+-----+
| Field          | Type          | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| ID             | int(11)       | NO   | PRI | NULL     | auto_increment |
| SellerID       | varchar(255) | NO   |     | NULL     |                |
| Title          | varchar(255) | YES  |     | NULL     |                |
| ISBN           | varchar(13)   | YES  |     | NULL     |                |
| Authors        | varchar(255) | YES  |     | NULL     |                |
| Edition        | varchar(255) | YES  |     | NULL     |                |
| Price          | varchar(255) | YES  |     | NULL     |                |
| Description     | varchar(255) | YES  |     | NULL     |                |
| University     | varchar(255) | YES  |     | NULL     |                |
| CourseCode     | varchar(255) | YES  |     | NULL     |                |
| ThumbnailLink  | varchar(255) | YES  |     | NULL     |                |
| GoogleInfoLink | varchar(255) | YES  |     | NULL     |                |
| Lat            | decimal(10,8) | YES  |     | NULL     |                |
| Lng            | decimal(11,8) | YES  |     | NULL     |                |
+-----+-----+-----+-----+-----+-----+
14 rows in set (0.00 sec)

mysql> describe WatchList;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| ID    | int(11)       | NO   | PRI | NULL     | auto_increment |
| SaleID | int(11)       | NO   | MUL | NULL     |                |
| UserID | varchar(255) | NO   |     | NULL     |                |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

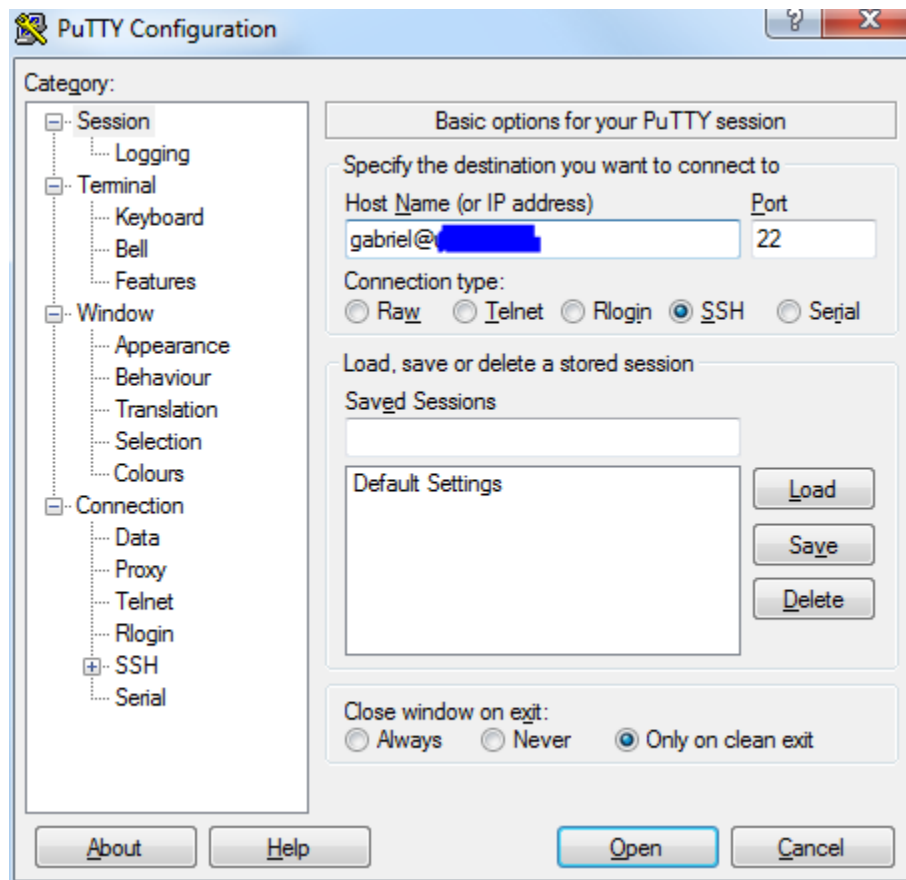
mysql> describe Message;
+-----+-----+-----+-----+-----+-----+
| Field          | Type          | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| ID             | int(11)       | NO   | PRI | NULL     | auto_increment |
| PreviousMsgID  | int(11)       | YES  | MUL | NULL     |                |
| NextMsgID      | int(11)       | YES  | MUL | NULL     |                |
| Receiver       | varchar(255) | NO   |     | NULL     |                |
| Sender         | varchar(255) | NO   |     | NULL     |                |
| Message        | varchar(255) | YES  |     | NULL     |                |
+-----+-----+-----+-----+-----+-----+
6 rows in set (0.00 sec)
```

**Figure 4.1:** The final database schema.



The final schema (Figure 4.1) for the MySQL database consisted of three tables. A table to store sales, a table to store a user's watched sales, and a table to store messages between users. These tables are updated through use of the server's RESTful web API. In order to avoid malicious users inserting bad records into our database, a Google oAuth token is required as a URL parameter. Furthermore, the server will only allow users who created a record to delete it. The full schema is also on the project GitHub page.

### 4.3 Software Configuration



**Figure 4.2:** The PuTTY main window.

This software is meant to be easy to install and use. It is open source and uses the MIT License. In order to administrate the server, all one needs is an SSH client, such as PuTTY for Windows. (Figure 4.2) . The server is run completely from the command line. As it requires some configuration, however, the main server file will provide a way for the user to create the configuration file using Python's *ConfigParser* module. The server will need to have access to a MySQL database with the provided schema, a python installation, and the dependencies listed on the GitHub. Once all these requirements are met, running the server will create the required configuration file (Figure 4.3).

```
(testenv)gabriel@cse2:~/textbook_test_env$ python server.py
Config not present. Creating config.
Hostname: localhost
Port: 800
OAuth Server Client ID: 
```

**Figure 4.3:** Interactively creating a configuration file.

## 4.4 Results

Overall the server works well. It is designed in such a way that it can perform any necessary action with only one API call. This is known as a REST API. It processes queries and then returns them to the client with a result and error field so the client knows what went wrong (Figure 4.4). It is crash-resistant and outputs verbose errors to assist in debugging the client application on both the client and server side. It is easy to see why this style of server is an industry standard for ease of use and development.



**Figure 4.4:** Trying to hit the API from a web browser.

## **Chapter 5**

# **Summary and Conclusion**

---

### **5.1 Summary**

As mentioned many times before, the Android Textbook Marketplace App is by students, for students. It is a collaborative effort between myself and Aaron Zhao to help relieve some of the financial pressure students feel during college. It will allow students to buy and sell textbooks from each other in the same university area, and use information like course codes to assist them.

It has many features that simplify use for the user, so they can feel secure about knowing what information they provide and to whom. It can be used with the server we created, or, with the included documentation, a different environment may be developed to even better serve students.

### **5.2 Implications**

Our main goal with this Android application is to help students save money. There are no figures or numbers, since development has just completed, but if we ever release it, we expect it could help students

save a significant amount of money. Its ease of use and simple UI design with no passwords required, and the ability to scan in a book's ISBN with the camera help to make users feel more comfortable using it over other services like Craigslist.

## 5.3 Recommendations

Unfortunately, we did not have enough time to complete all the features we would have liked in the app. It is functional, but not very secure, in my opinion. Requests to the server are sent in plaintext, unencrypted. In order to release this application on the Google Play Store, we would have to implement HTTPS on the server side. However, this is very simple to do with the current design. Once a signed certificate is acquired, the main code would just have to be modified to run the server with that certificate. If this is done, I am confident this app would be ready to be released. Other features that could be added include a map to display sales instead of just listing what is nearby, or push notifications for messages. These are just some examples for future developers, or things that we may implement later.

```
    httpserver.serve(app, host=ServerHost, port=ServerPort)
    # TODO: HTTPS
    # ssl_pem="cert/self-signed.pem"

if __name__ == '__main__':
    main()
```

**Figure 5.1:** The small code snippet for HTTPS implementation.

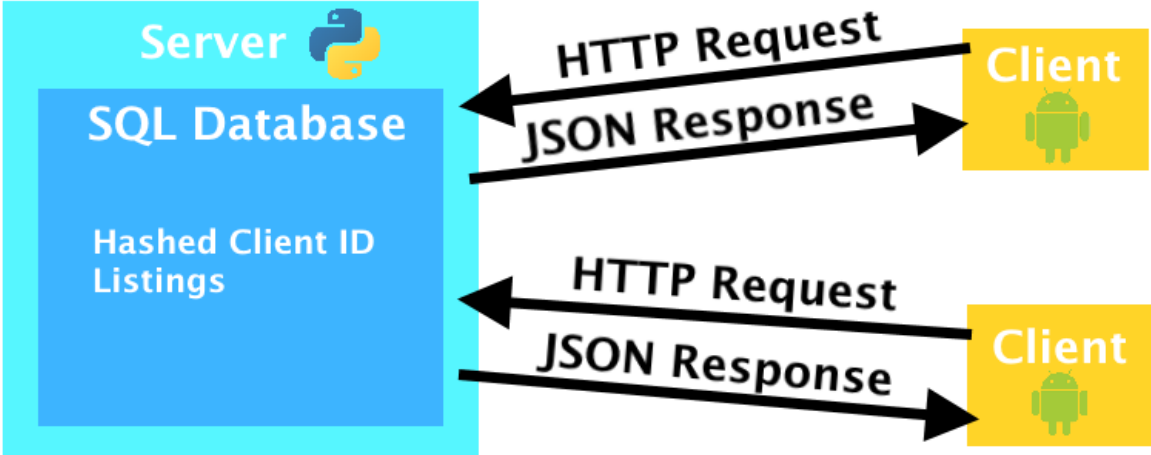
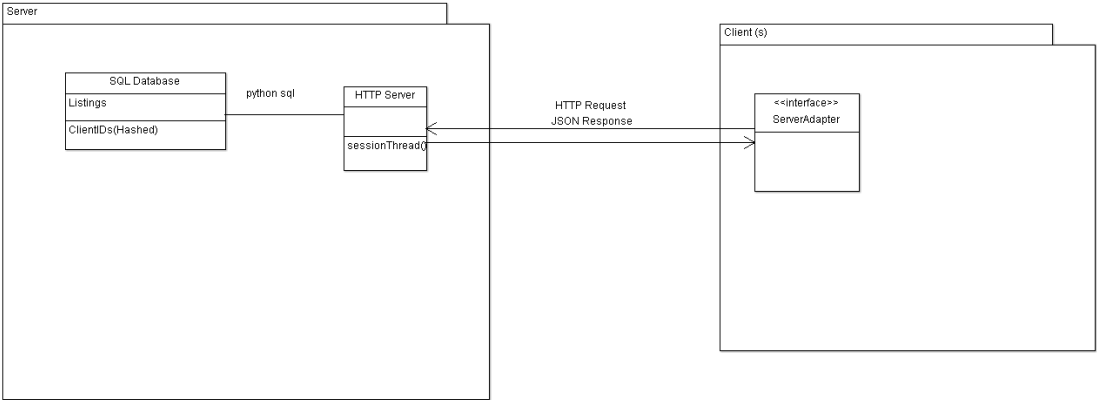
## 5.4 Conclusions

This software is only half of an entire project. Without it, the client app would not have network connectivity or be very useful, and without the client, the server would have no content to store. Not only is this software useful for the app it was designed for, but it is entirely open source, and can be built upon for other projects. We hope that our code will be used to write better server applications that use Python and MySQL, and if we publish our app, then we will put our all into making sure students are aware of it.

## References

- [1] Perry, M. J. (2015, July 26). The new era of the \$400 college textbook, which is part of the unsustainable higher education bubble - AEI. Retrieved from <https://www.aei.org/publication/the-new-era-of-the-400-college-textbook-which-is-part-of-the-unsustainable-higher-education-bubble/>
- [2] Google Books APIs | Google Developers. (n.d.). Retrieved from <https://developers.google.com/books/>
- [3] Kurtzleben, D. (2012, August 8). How your textbook dollars are divvied up. Retrieved from <http://www.usnews.com/news/articles/2012/08/28/how-your-textbook-dollars-are-divvied-up>
- [4] Amazon.com: Textbook Rentals, Rent Textbooks. (n.d.). Retrieved from <http://www.amazon.com/b?ie=UTF8&node=5657188011>
- [5] Boundless for Students. (n.d.). Retrieved from <https://www.boundless.com/students/>
- [6] Explaining the webapp2 Framework. (n.d.). Retrieved from <https://cloud.google.com/appengine/docs/python/gettingstartedpython27/usingwebapp>
- [7] PyMySQL/PyMySQL. (n.d.). Retrieved from <https://github.com/PyMySQL/PyMySQL>
- [8] Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures* University of California, Irvine. Retrieved from [http://www.ics.uci.edu/~fielding/pubs/dissertation/rest\\_arch\\_style.htm](http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm)
- [9] Authorizing with Google for REST APIs. (n.d.). Retrieved from <https://developers.google.com/android/guides/http-auth#obtain>
- [10] Introducing JSON. (n.d.). Retrieved from <http://www.json.org/>
- [11] Android Bootstrap. (n.d.). Retrieved from <http://www.androidbootstrap.com/>
- [12] Sending a Simple Request. (n.d.). Retrieved from <http://developer.android.com/training/volley/simple.html>
- [13] FileNotFoundException. (n.d.). Retrieved from <http://developer.android.com/reference/java/io/FileNotFoundException.html>
- [14] Using Assertions Effectively. (n.d.). Retrieved from <https://wiki.python.org/moin/UsingAssertionsEffectively>
- [15] Rasmusson, J. (2010). *The agile samurai: How agile masters deliver great software*. Raleigh, NC: The Pragmatic Bookshelf.

# Appendix A – Initial UML Diagram



## Appendix B – Source Code

App Demo Video: <https://www.youtube.com/watch?v=DpWqz-uhsrY>

Server GitHub: <https://github.com/espogabe/Textbook-Server>

App GitHub: <https://github.com/Aaron-Zhao/Textbookbns>

```
#!/usr/bin/env python

# Copyright (c) 2016 Aaron Zhao
# Copyright (c) 2016 Gabriel Esposito
# See LICENSE for details.

"""
REST API for Android Textbook Marketplace App.
"""

import os
import json
import decimal
import ConfigParser
import traceback

import webapp2
from oauth2client import crypt, client

from sqlwrapper import RESTWrapper, scrub
from helpers import point_near, dec_default, create_config

config = ConfigParser.ConfigParser()

# Make sure server config exists, if not, set it up
if not os.path.isfile('server.cfg'):
    create_config()
config.read('server.cfg')

SQLuser = config.get('mysql', 'user')
SQLpass = config.get('mysql', 'pass')
SQLhost = config.get('mysql', 'host')
SQLDB = config.get('mysql', 'db')

oAuthServerClientID = config.get('oAuth', 'server_client_id')
oAuthAndroidClientID = config.get('oAuth', 'android_client_id')

ServerHost = config.get('http', 'host')
ServerPort = config.get('http', 'port')

class AppAPI(webapp2.RequestHandler):
    """Request handler API class. Takes care of GET and POST requests."""

    def verify_oauth_token(self, token):
        """Verify the oAuth token with Google"""

        try:
            print "Verifying token"
            token = client.verify_id_token(token, oAuthServerClientID)
            print 'Token: %s' % token
```

```
    if token['aud'] != oAuthServerClientID:
        raise crypt.AppIdentityError("Unrecognized server.")
    if token['azp'] != oAuthAndroidClientID:
        raise crypt.AppIdentityError("Unrecognized client.")
    if token['iss'] not in ['accounts.google.com', 'https://accounts.google.com']:
        raise crypt.AppIdentityError("Wrong Issuer.")

    return True
except Exception, e:
    print e
    print traceback.format_exc()

return False

def verify_permissions(self, req):
    """Essentially allow everyone to read, but only the creator of a record can
write/delete"""

    if req['action'] in ['delete', 'update']:
        if 'SellerID' in req.keys():
            if req['SellerID'] == get_sellerid(req['token']):
                return True
            return False
        else:
            return True

def get(self):
    """Handle GET requests by passing to SQL wrapper. Respond with JSON."""

    self.response.headers['Content-Type'] = 'application/json'
    print self.request.GET
    try:
        if self.verify_oauth_token(self.request.GET['token']) and
self.verify_permissions(self.request.GET):
            action = self.request.GET['action']
            if action in ['select', 'delete', 'insert', 'update']:

                table = self.request.GET['table']

                wr = RESTWrapper(SQLHost, SQLUser, SQLpass, SQLDB)

                sql_args = self.request.GET
                del sql_args['token']
                del sql_args['action']
                del sql_args['table']

                res = wr.query(action, table, **sql_args)

                if action in ['select', 'insert']:
                    self.response.write(json.dumps({"error": "", "message": res},
default=dec_default))
                else:
                    self.response.write(json.dumps({"error": "", "message":
"Successful"}))
            elif action == 'getsalesnearby':

                lat = double(scrub(self.request.GET['lat']))
                lng = double(scrub(self.request.GET['long']))
                radius = int(scrub(self.request.GET['r']))

                wr = RESTWrapper(SQLHost, SQLUser, SQLpass, SQLDB)
```



```
res = wr.query_passthru("""SELECT * FROM Sale WHERE Lat IS NOT NULL AND
Lng IS NOT NULL""")

nearest = []

# Find sales within r miles of user
for row in res:
    if point_near(lat, lng, row['Lat'], row['Lng'], r):
        nearest.append(row)

self.response.write(json.dumps({"error": "", "message": nearest},
default=dec_default))
    else:
        self.response.write(json.dumps({"error": "Invalid action", "message":
""}))
    else:
        self.response.set_status(401)
        self.response.write(json.dumps({"error": "Unauthorized token or bad
permissions (You can only delete and update records you created)", "message": ""}))
    except KeyError, e:
        print e
        print traceback.format_exc()
        self.response.write(json.dumps({"error": "Malformed GET request", "message":
""}))
    except Exception, e:
        print e
        print traceback.format_exc()
        self.response.write(json.dumps({"error": str(e), "message": ""}))

app = webapp2.WSGIApplication([
    ('/api', AppAPI),
], debug=True)

def main():
    from paste import httpserver
    httpserver.serve(app, host=ServerHost, port=ServerPort)
    # TODO: HTTPS
    # ssl_pem="cert/self-signed.pem"

if __name__ == '__main__':
    main()
```