Part1

a. What will your project do?
    i. A marketplace application that offers great convenience to students who want to purchase products or services online from
b. Who or what will be its users? Your project must impose registration, authenticated login and timeout/explicit logout, so your answer should say something about this.
    i. Students will be the users.
    ii. All users must be registered in order to use the website. Users can create their own accounts or login via accounts from either Facebook or Apple
    iii. If a user is not active for more than 20 minutes, this user's session will end.
c. Your project must be demoable, but does not need a GUI if there's a command line console or some other way to demonstrate. (All demos must be entirely online, there will be no in-person demos.) What do you think you'll be able to show in your demo?
    i. A website.
d. Your project must store and retrieve some application data persistently (e.g., using a database or key-value store, not just a file), such that when your application terminates and starts up again some arbitrary time later, the data is still there and used for later processing. What kind of data do you plan to store?
    i. Plan to store user information in SQL. We are going to use SQLAlchemy (https://flask-sqlalchemy.palletsprojects.com/en/2.x/). We will create a few database models and set up relationships between tables and keys. .
e. Your project must leverage some publicly available API beyond those that "come with" the platform; it is acceptable to use an API for external data retrieval instead of to call a library or service. The API does not need to be a REST API. There are many public APIs linked at https://github.com/public-apis/public-apis (Links to an external site.) and https://github.com/n0shake/Public-APIs (Links to an external site.). What API do you plan to use and what will you use it for?
    i. Paypal API for payment processing
    ii. Facebook and Apple API for login

Part 2

Write three to five user stories for your proposed application, constituting a Minimal Viable Product (MVP); registration/login/logout should not be included among these user stories, nor should 'help' or other generic functionality. That is, your application should do at least three application-specific things. Use the format

(1) As a **student buyer**, I **want** to be able to purchase cheap used textbooks for my classes **so that** I do not have to pay extremely high prices at Amazon and wait for 5 days.

My conditions of satisfaction are:

- If there are no textbooks I want to find, I can post a request for that textbook.
- If there are textbooks I want to buy, I can contact the seller and buy that textbook.
- Before purchasing, I want to be able to see the condition of the books via pictures/videos.

(2) As a **student seller,** I **want** to be able to post things I don’t need anymore and sell them **so that** I can get some money back.

My conditions of satisfaction are:

- I can provide text descriptions and photos of the thing I want to sell
- I can determine the price of the merchandise
- I can post my contact information for buyers to contact me

(3) As a **student seller**, I **want** to be able to receive my payments securely and promptly after transaction so that I feel safe and secure.

My conditions of satisfaction are:

- I can receive my payment within a short period of time.
- No chargeback.
- I can approve whether or not I want to sell my product. I can both deny and approve and transaction.

(4) As an **admin**, I want to be able to **get the total amount of transactions** on the website so that I can use it to track my revenue from the website.

My conditions of satisfaction are:

- I can see an aggregate of all the previous transactions on the website.
- I can see the monthly revenue of the website. If the monthly revenue is down from the previous month, it shows the figure in red. If the monthly revenue is up, then it shows in green.

Part 3

Explain how you will conduct acceptance testing on your project. This means that every MVP user story must be associated with a plan for user-level testing. The test plan should address both common cases and special cases. Discuss sample inputs the user or client would enter and the results expected for the corresponding test to pass vs. fail. Note inputs might come from files, network, devices, etc., not necessarily from a GUI or command line, and results might involve changes in application state, files, outgoing network traffics, control of devices, etc., not necessarily outputs via a GUI or command line. You may optionally discuss testing plans for your wishlist additional user stories, if any.

(1) **Student buyer acceptance testing**

(a) Common cases:

(i) User inputs the name of the item that he/she wants to buy in the text field

1) If found, then a list of relevant items should be return and shown on the website. If not found, it should return a message "Nothing is found" with the corresponding error code. If none of these happens, the test fails.
2) Each item in the return list must include at least one photo, and the user should be able to look at the photos/videos of the item. If this cannot be done, the test fails.
3) The user should be able to contact the seller and chat with the seller for that item. If this cannot be done, the test fails.
4) The user should be able to send a secure payment to the seller if the user decides to purchase. If this cannot be done, the test fails.
5) After the transaction, the item should be marked "unavailable" and the post should be archived. If this cannot be done, the test fails.

(ii) User inputs the name of the item that he/she wants to buy but cannot find the item.

1) The user will see a message "Nothing is found." If this cannot be done, the test fails.
2) The user will be able to post a "Want to buy" request via a "Want to buy" button on the website. If this cannot be done, the test fails.
3) Each want-to-buy post must include a title, a product description and a want-to-buy-price associated with the item. If this cannot be done, the test fails.
4) After posting, the posting should be shown under the "Want to buy" category and the general public can see that post. If this cannot be done, the test fails.

(b) Uncommon cases:

(i) User will input the name of the item he/she is interested in, and manages to find the item, yet the item has already been sold and this is not reflected on the website.

1) The seller can mark the item unavailable. If the seller can't mark it or the result of the action can not be reflected on the website, the test fails.

(2) Student seller posting related acceptance testing

The user input for student sellers will be a post of some kind of merchandise.

Common cases:

1. The seller wants to create a new post to sell something.
    a. In the post, the seller can enter the name of the merchandise, a text description, some related photos, contact information and a price.
    b. After they complete the post, all buyers should be able to see it on the website.
2. After a merchandise is sold, i.e. the seller has received payment from the buyer, the post should be automatically closed so that no buyers can see it anymore.

Special cases:

1. The seller decides to stop selling one of the merchandise he posted. He should be able to remove the post so that no buyers can see it anymore.

(3) Student seller transaction related acceptance testing

The input is the update to the *product* database that a particular item for sale has been *in_transaction*. The *transaction* object is either approved by Paypal API or rejected.

Common case:

1. The transaction was successful.
    a. Assert that Paypal API returns *success* to the transaction.
    b. Assert that given product in the *product* database is marked as *sold*.
    c. Assert that the seller is notified that the transaction succeeded.

Uncommon case:

1. The transaction was unsuccessful
    a. Assert that Paypal API returns *failure* to the transaction.
    b. Assert that given product in the *product* database is again marked as *not_sold*.
    c. Assert that the seller is notified that the transaction failed.

(4) **Admin related acceptance testing**

The user input is the period of which they want to see the revenue, which could be monthly or total aggregate. Depending on the input of the user, the system will show the number corresponding to the monthly, yearly, or total revenue of the website.

Common case:

1. User chooses *Monthly* revenue. It returns a view of monthly revenue for the past 12 months from the *Transaction* database.

        a. *Assert* that: there are twelve returned values corresponding to monthly revenue.
        b. *Assert* that months with lower revenue compared to previous month have color variable “Red.”
        c. *Assert* that the months with higher revenue compared to previous month have color variable “Blue.”
        d. *Assert* that screen title shows “Monthly Revenue.”

2. User chooses *Total* revenue. It returns a view of total aggregate revenue from the *transaction* database
    a. *Assert* that there is only one positive value returned.
    b. *Assert* that the screen title shows “Total Revenue.”

Uncommon cases:

1. The admin does not have access to the revenue database. The system returns an error message.
    a. In acceptance testing, we can suppose a case where the user does not have access to the database (it does not exist). Then we can assert that the error message code we received is the same as the error message code for not having access to the database (or database not found).

Part 4:
Identify the specific facilities corresponding to JDK, Eclipse, Maven, CheckStyle, JUnit, Emma, Spotbugs, and SQLite that your team plans to use. That is, state what you plan to use for compiler/runtime (or equivalent), an IDE or code editor, a build tool (or package manager if 'build' not applicable), a style checker, a unit testing tool, a coverage tracking tool, a bug finder (that's not just a style checker), and a persistent data store appropriate for your chosen language(s) and platform(s). If different members of the team plan to use different tools, please explain. It is ok to change your choice of tools later after you start developing your application.

Back-end:
IDE: Pycharm
Compiler: python 3
Package manager: pip
Unit Testing : unittest module
Style Checker: pep8
Coverage: Coverage.py module
DB: MySQL
Bug finder: DeepBugs
https://plugins.jetbrains.com/plugin/12218-deepbugs-for-python

Front-end:
IDE: Visual Studio Code
Package manager: npm
Style checker: ESLint
Bug finder: ESLint
Language: javascript
Framework: Reactjs**Part 1**