# BookStore Report

- 罗皓天 520030910325

## Backend and DB Selection

flask

postgreSQL + sqlalchemy

## DataBase Design

- users: we have the info of every user about their id, pwd, money, time_token and terminal

| user_id | password | balance | token | terminal |
|---|---|---|---|---|

- stores: we have the info of every store's id and its book_id. Also the info of the book and stock_level

| store_id | book_id | book_info | stock_level |
|---|---|---|---|

- user_store: we have the relation of the owner between user and store

| user_id | store_id |
|---|---|

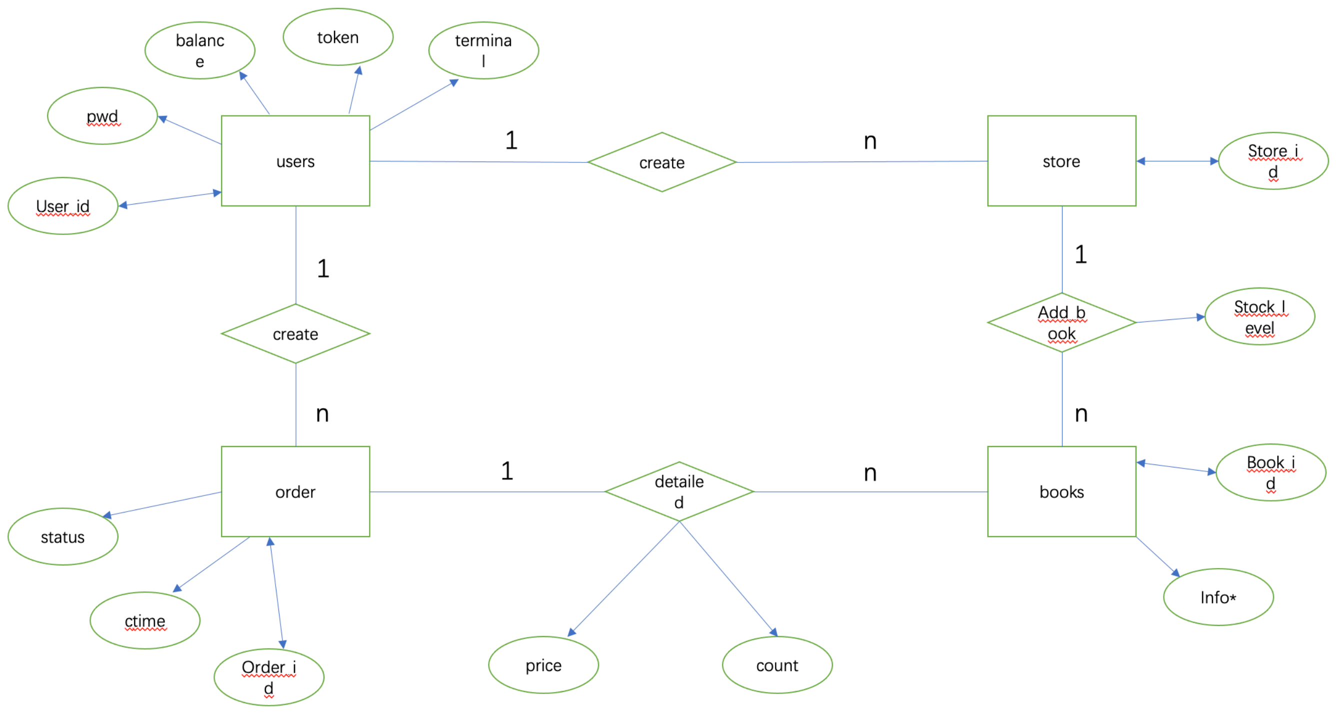
- new_order: we have the info of orders

| order_id | user_id | store_id | status | create_time | total_price |
|---|---|---|---|---|---|

- new_order_detail: some additional info of orders

| order_id | book_id | count | price |
|---|---|---|---|

## ER



## Details

### Base 60%

Use sqlalchemy to implement the backend logic of postgreSQL (Only show the Important Parts)

- user.py
  - register

    Add a new info of user into the database "users" and immediately commit

    ```
    new_user = users(user_id = user_id, password = password, balance = 0,
                     token = token, terminal = terminal)

    self.session.add(new_user)
    self.session.commit()
    ```

  - check_password

    Find the password of the user in the database and check whether it matches

    ```
    result = self.session.query(users).filter(users.user_id == user_id).all()
    if len(result) == 0:
        return error.error_authorization_fail()
    if password != result[0].password:
        return error.error_authorization_fail()
    ```

  - login

  Invoke check_password to check the user's password and update the token in the database to store the timestamp of the time when the user login, commit immediately

  ```
  self.session.query(users).filter(users.user_id == user_id).update({users.token:
  token, users.terminal: terminal})
  self.session.commit()
  ```

  - logout

    check the token of the user first

    then update the token in the database (dummy_token), commit immediately

    If the duration has been more than 3,600 s, the user will logout automatically

    ```
    terminal = "terminal_{}".format(str(time.time()))
    dummy_token = jwt_encode(user_id, terminal)
    self.session.query(users).filter(users.user_id == user_id).update({
      users.token: dummy_token, users.terminal: terminal})
    self.session.commit()
    ```

  - unregister

    check the password first

    then delete the infomation of the user from the database, and commit immediately

    ```
    self.session.query(users).filter(users.user_id == user_id).delete()
    self.session.commit()
    ```

  - change_password

    Fisrt, invoke check_password to check the user's password and then updata the new password in the database, commit immediately

    ```
    self.session.query(users).filter(users.user_id == user_id).update({
      users.password: new_password, users.token: token, users.terminal: terminal})
    self.session.commit()
    ```

- buyer.py
  - new_order

    Fisrt, check the whether the bookstore and the book exist

    Then, check the stock_level of the book

    Update the new stock_level and create a new_order_detail row for every book in the database

    Create a new_order for this order in the database.

    All operation of the DB commit immediately.

```
for books in ...:
  self.session.query(stores).filter(stores.store_id == store_id, stores.book_id ==
book_id, stores.stock_level >= count).update({stores.stock_level:
stores.stock_level - count})
  self.session.commit()
  order_detail = new_order_detail(order_id = uid, book_id = book_id, count = count,
price = price)
  self.session.add(order_detail)
  self.session.commit()
order = new_order(order_id = uid, store_id = store_id, user_id = user_id, status =
0, create_time = datetime.datetime.now(), total_price = total_price)
self.session.add(order)
self.session.commit()
```

  - payment

    First, cheak the user's password and the order id

    Then check the balance of the user

    If the balance is enough, then update the status of order to paid, the new balance of the buyer in the database.

    All operation of the DB commit immediately.

    ```
    session.query(users).filter(users.user_id == buyer_id).update({users.balance:
    users.balance - total_price})
    session.commit()

    session.query(new_order).filter(new_order.order_id ==
    order_id).update({new_order.status: 1})
    session.commit()
    ```

  - add_funds

    First, cheak the password of the user. Then update the balance.

    All operation of the DB commit immediately.

    ```
    self.session.query(users).filter(users.user_id ==
    user_id).update({users.balance: users.balance + add_value})
    self.session.commit()
    ```

- seller.py
  - add_book

    Check the store_id and book_id and update the info of store in the database

    All operation of the DB commit immediately.

```
new_book = stores(store_id = store_id, book_id = book_id, book_info =
book_json_str, stock_level = stock_level)
self.session.add(new_book)
self.session.commit()
```

- add_stock_level

  Check the store_id and book_id and update the stock_level of book in the database

  All operation of the DB commit immediately.

  ```
  self.session.query(stores).filter(stores.store_id == store_id, stores.book_id
  == book_id).update({stores.stock_level: stores.stock_level + add_stock_level})
  self.session.commit()
  ```

- create_store

  Check the store_id and add a new info of user_store in the database.

  ```
  new_store = user_store(store_id = store_id, user_id = user_id)
  self.session.add(new_store)
  self.session.commit()
  ```

- store.py
  - init_database()

    Init the database (create the table)

    ```
    global database_instance
       database_instance.Base.metadata.create_all(database_instance.engine)
    ```

- db_conn.py

  Some utils to check whether the _id exists

  - user_id_exist && book_id_exist && store_id_exist

    ```
    def user_id_exist(self, user_id):
        result = self.session.query(users).filter(users.user_id == user_id).all()
        return len(result) != 0


    def is_my_store(self, user_id, store_id):
        result = self.session.query(user_store).filter(user_store.user_id ==
    user_id, user_store.store_id == store_id).all()
        return len(result) != 0


    def book_id_exist(self, store_id, book_id):
    ```

```
    result = self.session.query(stores).filter(stores.store_id == store_id,
stores.book_id == book_id).all()
    return len(result) != 0


def store_id_exist(self, store_id):
    result = self.session.query(user_store).filter(user_store.store_id ==
store_id).all()
    return len(result) != 0
```

## Addition 40%

### Delivery

- buyer.py
  - send_out_delivery

    First check the *order_id* and *user_id*

    Then update the status of the order (status = 2)

    All operation of the DB commit immediately.

    ```
                self.session.query(new_order).filter(new_order.order_id ==
    order_id).update({new_order.status: 2})
    self.session.commit()
    ```

  - take_delivery

    First check the *order_id* and *user_id*

    Then update the status of the order (status = 3), and add the balance to the seller's account

    ```
    self.session.query(users).filter(users.user_id ==
    seller_id).update({users.balance: users.balance + total_price})
    self.session.commit()
                self.session.query(new_order).filter(new_order.order_id ==
    order_id).update({new_order.status: 3})
    self.session.commit()
    ```

### Search Book

- user.py
  - search_for_book

    ```
    def search_for_book(self, user_id:str, target: str ,store_id: str = '-1') ->
    (int, str, list)
    ```

If the value of *store_id* is -1, we search in all stores.

If the results of search is too large, we will return only 10 results once with `limit()`

## Order Function

- buyer.py
  - cancel_order()

    Cancel the order and update the information accroding to the status of order(paid or not paid)
- order cancel automatically
  - be/server.py

    Check the order every 15 seconds (a short time to test easily), the order in status 4 would be deleted

    ```
    scheduler = APScheduler()
    scheduler.add_job('regular_inspection', regular_inspection, trigger='interval',
    seconds=15)
    scheduler.start()
    ```

  - be/times.py
    A function to check the status of orders. If the order is not paid, then cancel the order automatically

    ```
    def regular_inspection():
        duration_limit = 10

        session = store.get_db_conn()
        result = session.query(new_order).filter(new_order.status == 0).all()
        for row in result:
            order_id = row.order_id
            duration = (datetime.datetime.now() - row.create_time).total_seconds()
            if duration > duration_limit:
                store_id = row.store_id
                booklist =
    session.query(new_order_detail).filter(new_order_detail.order_id ==
    order_id).all()
                for bookrow in booklist:
                    book_id = bookrow.book_id
                    count = bookrow.count
                    session.query(stores).filter(stores.store_id == store_id,
    stores.book_id == book_id).update({stores.stock_level: stores.stock_level +
    count})
                    session.commit()
                session.query(new_order).filter(new_order.order_id ==
    order_id).update({new_order.status: 4})
                session.commit()
    ```

Check the duration: if the time is longer than the pre-defined value and the order is not paid, then cancel the order.

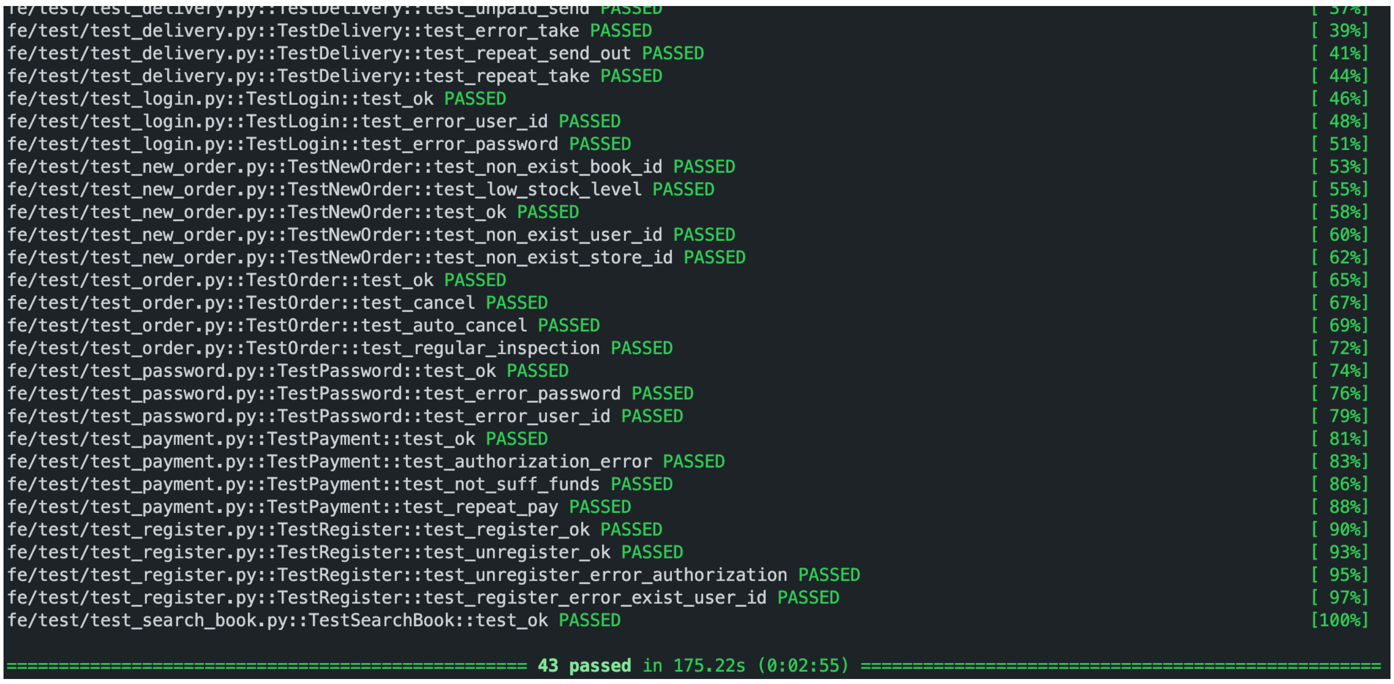
- query_order

  Search all order and show their status.

  ```
  booklist = self.session.query(new_order_detail).filter(new_order_detail.order_id ==
  order_id).all()
  ```

# Evaluations

## Base 60%



```
fe/test/test_delivery.py::TestDelivery::test_unpaid_send PASSED                                     [ 37%]
fe/test/test_delivery.py::TestDelivery::test_error_take PASSED                                      [ 39%]
fe/test/test_delivery.py::TestDelivery::test_repeat_send_out PASSED                                 [ 41%]
fe/test/test_delivery.py::TestDelivery::test_repeat_take PASSED                                     [ 44%]
fe/test/test_login.py::TestLogin::test_ok PASSED                                                    [ 46%]
fe/test/test_login.py::TestLogin::test_error_user_id PASSED                                         [ 48%]
fe/test/test_login.py::TestLogin::test_error_password PASSED                                        [ 51%]
fe/test/test_new_order.py::TestNewOrder::test_non_exist_book_id PASSED                              [ 53%]
fe/test/test_new_order.py::TestNewOrder::test_low_stock_level PASSED                                [ 55%]
fe/test/test_new_order.py::TestNewOrder::test_ok PASSED                                             [ 58%]
fe/test/test_new_order.py::TestNewOrder::test_non_exist_user_id PASSED                              [ 60%]
fe/test/test_new_order.py::TestNewOrder::test_non_exist_store_id PASSED                             [ 62%]
fe/test/test_order.py::TestOrder::test_ok PASSED                                                    [ 65%]
fe/test/test_order.py::TestOrder::test_cancel PASSED                                                [ 67%]
fe/test/test_order.py::TestOrder::test_auto_cancel PASSED                                           [ 69%]
fe/test/test_order.py::TestOrder::test_regular_inspection PASSED                                    [ 72%]
fe/test/test_password.py::TestPassword::test_ok PASSED                                              [ 74%]
fe/test/test_password.py::TestPassword::test_error_password PASSED                                  [ 76%]
fe/test/test_password.py::TestPassword::test_error_user_id PASSED                                   [ 79%]
fe/test/test_payment.py::TestPayment::test_ok PASSED                                                [ 81%]
fe/test/test_payment.py::TestPayment::test_authorization_error PASSED                               [ 83%]
fe/test/test_payment.py::TestPayment::test_not_suff_funds PASSED                                    [ 86%]
fe/test/test_payment.py::TestPayment::test_repeat_pay PASSED                                        [ 88%]
fe/test/test_register.py::TestRegister::test_register_ok PASSED                                     [ 90%]
fe/test/test_register.py::TestRegister::test_unregister_ok PASSED                                   [ 93%]
fe/test/test_register.py::TestRegister::test_unregister_error_authorization PASSED                  [ 95%]
fe/test/test_register.py::TestRegister::test_register_error_exist_user_id PASSED                    [ 97%]
fe/test/test_search_book.py::TestSearchBook::test_ok PASSED                                         [100%]

================================================== 43 passed in 175.22s (0:02:55) ==================================================
```

All 42 pass (contains our own tests) in 175.22s (contains 15s sleep for our own test)

## Additional 40%

For the additional function, we write three more test.py:

- test_delivery.py

  First, create an order and test the delivery and take delivery.

  We test some unallowed behaviors

  - deliver before payment
  - take delivery before deliver
  - repeat deliver
  - repeat take delivery

- test_search_book.py

  First, we create two book stores and add some books(a part of these books are same)

  We test the function search in one certain store and all stores.

- test_order.py

  First, we create some orders.

  We test the delivery and take delivery. (For testing some unallowed behaviors)

  We test some unallowed behaviors

  - deliver before payment
  - auto cancel
  - deliver after cancel

  We test auto cancel by sleep 15s (a short time to test easily)

All our test programs have cover all the error number which we write. We also test some unallowed behaviors to ensure. So we believe our code coverage is high.

We use *coverage* to test

```
Name                                 Stmts   Miss Branch BrPart  Cover
----------------------------------------------------------------------
be/__init__.py                           0      0      0      0   100%
be/app.py                                3      3      2      0     0%
be/model/__init__.py                     0      0      0      0   100%
be/model/buyer.py                      230     59     88     19    71%
be/model/db_conn.py                     17      0      0      0   100%
be/model/error.py                       25      3      0      0    88%
be/model/seller.py                      64     18     26      3    70%
be/model/store.py                       73      0      0      0   100%
be/model/user.py                       161     46     58     10    65%
be/serve.py                             39      2      2      1    93%
be/times.py                             21      0      6      0   100%
be/view/__init__.py                      0      0      0      0   100%
be/view/auth.py                         46      1      0      0    98%
be/view/buyer.py                        58      0      2      0   100%
be/view/seller.py                       28      0      0      0   100%
fe/__init__.py                           0      0      0      0   100%
fe/access/__init__.py                    0      0      0      0   100%
fe/access/auth.py                       36      0      0      0   100%
fe/access/book.py                      106     19     10      1    83%
fe/access/buyer.py                      60      0      2      0   100%
fe/access/new_buyer.py                   8      0      0      0   100%
fe/access/new_seller.py                  8      0      0      0   100%
fe/access/seller.py                     31      0      0      0   100%
fe/bench/__init__.py                     0      0      0      0   100%
fe/bench/run.py                         13      0      6      0   100%
fe/bench/session.py                     47      0     12      1    98%
fe/bench/workload.py                   125      1     22      2    98%
```

```
fe/conf.py                           11      0      0      0   100%
fe/conftest.py                       17      1      0      0    94%
fe/test/gen_book_data.py             48      0     16      0   100%
fe/test/gen_book_many.py             48      4     16      5    86%
fe/test/test_add_book.py             35      0     10      0   100%
fe/test/test_add_funds.py            22      0      0      0   100%
fe/test/test_add_stock_level.py      38      0     10      0   100%
fe/test/test_bench.py                 6      2      0      0    67%
fe/test/test_create_store.py         19      0      0      0   100%
fe/test/test_delivery.py             68      1      4      1    97%
fe/test/test_login.py                27      0      0      0   100%
fe/test/test_new_order.py            39      0      0      0   100%
fe/test/test_order.py                94      0      2      0   100%
fe/test/test_password.py             32      0      0      0   100%
fe/test/test_payment.py              59      1      4      1    97%
fe/test/test_register.py             30      0      0      0   100%
fe/test/test_search_book.py          51      0      4      0   100%
-------------------------------------------------------------------
TOTAL                              1843    161    302     44    88%
```

Our total coverage is 88%

## Improvement

### 事务处理

Every operation except query will commit immediately in my code, so I believe it can handle very large number of sessions.

### Index

All primary_key in the tables has index to help speed up the query.

### Access Control

We find that there do not exist some access control between the owner of the store and the buyers.

Owing to the lack of access control, any user who knows the *store_id* can modify the info of books, so we add access control in the backend.

```python
def is_my_store(self, user_id, store_id):
    result = self.db['user_store'].find_one({'user_id': user_id, 'store_id': store_id})
    if result is None:
        return False
    else:
        return True
```

And we will invoke this function to check access.

## Git

https://github.com/AegeanYan/DB_PJ1

Git and GitHub is used to manage the code version