Testing

Unit tests

VM.py

```
def add_to_cart(self, product_idx):
    if product_idx not in [0, 1, 2, 3, 4]:
        return False
    product = self.products[product_idx]
    if product.stock > 0 and (product.stock > self.selected_products[product_idx]):
        self.selected_products[product_idx] += 1
        self.last_operation_time = time.time()
        return True
    return False
```

TC1.1.1: add_to_cart_failed -> matched by T1.1.2 test_add_to_cart_fail TC1.1.2: add_to_cart_succeeded -> matched by T1.1.1 test_add_to_cart TC1.1.3: invalid_product_index -> matched by T1.1.3 test_add_to_cart_invalid TC1.1.4: valid_product_index -> matched by T1.1.1 test_add_to_cart

```
def remove_from_cart(self, product_idx):
    if product_idx not in [0, 1, 2, 3, 4]:
        return False
    if product_idx in self.selected_products:
        self.selected_products[product_idx] -= 1
        if self.selected_products[product_idx] <= 0:
            del self.selected_products[product_idx]
        self.last_operation_time = time.time()
        return True
    return False
```

TC1.2.1: remove_from_cart_failed -> matched by T1.2.2 test_remove_from_cart_fail TC1.2.2: remove_from_cart_succeeded -> matched by T1.2.1 test_remove_from_cart TC1.2.3: selected_products_sub_to_0 -> matched by T1.2.1 test_remove_from_cart TC1.2.4: selected_products_sub -> matched by T1.2.3 test_remove_from_cart_remain TC1.2.5: remove_valid_product -> matched by T1.2.1 test_remove_from_cart TC1.2.6: remove_invalid_product -> matched by T1.2.4 test_remove_from_cart_invalid

```
def process_payment(self):
    total = self.calculate_total()
    self.last_operation_time = time.time()
    if total == 0:
        return False, {}, "请先选购商品"
    if self.current_amount < total:
        return False, {}, f"金额不足，还需{total - self.current_amount:.1f}元"

    change = self.current_amount - total
    if change > 0 and not self.can_give_change(change):
        return False, {}, "零钱不足，请先退币"

    self.current_amount -= total
    for idx, qty in self.selected_products.items():
        self.products[idx].stock -= qty

    success_msg = f"支付成功！当前余额：{self.current_amount:.1f}元"
    raw_products = dict(self.selected_products)
    self.selected_products.clear()
    products = {idx + 1: qty for idx, qty in raw_products.items() if qty > 0}
    return True, products, success_msg
```

TC1.3.1: product_not_selected -> matched by T1.3.4 test_process_payment_no_product TC1.3.2: product_selected -> matched by T1.3.1 test_process_payment_success TC1.3.3: insert_amount_insufficient -> matched by T1.3.2 test_process_payment_insufficient TC1.3.4: insert_amount_sufficient -> matched by T1.3.1 test_process_payment_success

TC1.3.5: change_insufficient -> matched by T1.3.3 test_process_payment_not_refund TC1.3.6: purchase_succeeded -> matched by T1.3.1 test_process_payment_success

```
def refund_all(self):
    self.last_operation_time = time.time()
    if self.current_amount == 0:
        return {}, "当前没有可退金额"

    actual_refund, change_result = self.give_change(self.current_amount)
    self.current_amount = 0
    return change_result, f"退币成功！退还：{actual_refund:.1f}元"
```

TC1.4.1: no_refund_amount -> matched by T1.4.2 test_refund_all_no_money TC1.4.2: refund_succeeded -> matched by T1.4.1 test_refund_all

```
def can_give_change(self, amount):
    temp_stock = self.coin_stock.copy()
    for coin in self.valid_coins:
        temp_stock[coin] += self.insert_stock[coin]
    remaining = amount
    for coin in self.valid_coins:
        if remaining <= 0.05:
            return True
        count = min(remaining // coin, temp_stock[coin])
        remaining -= count * coin
    return remaining <= 0.05
```

TC1.5.1: can_give_change -> matched by T1.5.1 test_can_give_change TC1.5.2: cannot_give_change -> matched by T1.5.1 test_can_give_change

```
def give_change(self, amount):
    for coin in self.valid_coins:
        self.coin_stock[coin] += self.insert_stock[coin]
        self.insert_stock[coin] = 0
    remaining = amount
    change_given = 0
    change_result = {coin: 0 for coin in self.valid_coins}
    for coin in self.valid_coins:
        if remaining <= 0:
            break
        count = min(remaining // coin, self.coin_stock[coin])
        if count > 0:
            self.coin_stock[coin] -= count
            remaining -= count * coin
            change_given += count * coin
            change_result[coin] = int(count)
    return change_given, change_result
```

TC1.6.1: give_changes -> matched by T1.4.1 test_refund_all TC1.6.2: searching_changes -> matched by T1.4.1 test_refund_all TC1.6.3: have_required_coins -> matched by T1.4.1 test_refund_all TC1.6.4: no_required_coins -> will not be covered. Never happened since the setting of insert_stock and function of can_give_change

```
def check_timeout(self):
    need_clear = (time.time() - self.last_operation_time) > self.timeout
    if need_clear:
        if self.selected_products:
            self.selected_products.clear()
        if self.current_amount > 0:
            for coin in self.valid_coins:
                self.coin_stock[coin] += self.insert_stock[coin]
                self.insert_stock[coin] = 0
            self.current_amount = 0
    return need_clear
```

TC1.7.1: time_in -> matched by T1.3.1 test_process_payment_success TC1.7.2: time_out -> matched by T1.6.1 test_check_timeout TC1.7.3: products_remained -> matched by T1.6.2 test_check_timeout_product_remain TC1.7.4: no_products_remained -> matched by T1.6.1 test_check_timeout TC1.7.5: money_remained -> matched by T1.6.3 test_check_timeout_money_remain TC1.7.6: no_money_remained -> matched by T1.6.1 test_check_timeout

```
def verify_admin(self, input_pwd):
    if input_pwd == self.admin_password:
        return True
    return False
```

TC1.8.1: password_correct -> matched by T1.7.1 test_verify_admin TC1.8.2: password_wrong -> matched by T1.7.1 test_verify_admin

```
def refill_coin(self, coin):
    if coin in [50, 20, 10, 5, 1, 0.5]:
        if self.coin_stock[coin] < 20:
            self.coin_stock[coin] = 20
    self.check_alerts()
    return True
```

TC1.9.1: refill_100 -> matched by T1.8.1 test_refill_coins TC1.9.2: refill_non_100 -> matched by T1.8.1 test_refill_coins TC1.9.3: refill_coin_stock_sufficient -> matched by T1.8.2 test_refill_coins_sufficient TC1.9.4: refill_coin_stock_insufficient -> matched by T1.8.1 test_refill_coins

```
def withdraw_coin(self, coin):
    if coin == 100:
        self.coin_stock[coin] = 0
    elif coin in self.valid_coins:
        if self.coin_stock[coin] > 20:
            self.coin_stock[coin] = 20
    else:
        return False
    self.check_alerts()
    return True
```

TC1.10.1: withdraw_100 -> matched by T1.9.1 test_withdraw_coins TC1.10.2: withdraw_non_100 -> matched by T1.9.1 test_withdraw_coins TC1.10.3: withdraw_coin_stock_sufficient -> matched by T1.9.2 test_withdraw_coins_sufficient TC1.10.4: withdraw_coin_stock_insufficient -> matched by T1.9.1 test_withdraw_coins TC1.10.5: withdraw_valid_coins -> matched by T1.9.1 test_withdraw_coins TC1.10.6: withdraw_invalid_coins -> matched by T1.9.3 test_withdraw_coins_invalid

```
def insert_coin(self, coin):
    if coin in self.valid_coins:
        self.current_amount += coin
        self.insert_stock[coin] += 1
        self.last_operation_time = time.time()
```

TC1.11.1: insert_valid_coins -> matched by T1.10.1 test_insert_coin TC1.11.2: insert_invalid_coins -> matched by T1.10.2 test_insert_invalid_coin

```
def restock_product(self, product_idx):
    if product_idx not in [0, 1, 2, 3, 4]:
        return False
    product = self.products[product_idx]
    product.stock = 20
    self.check_alerts()
    return True
```

TC1.12.1: restock_valid_products -> matched by T1.11.1 test_restock_products
TC1.12.2: restock_invalid_products -> matched by T1.11.2 test_restock_invalid_products

UI.py

```
def setup_user_interface(self):
    ...
    control_layout = QVBoxLayout()
    self.cart_list = QListWidget()
    control_layout.addWidget(self.cart_list)

    coin_row = QHBoxLayout()
    for coin in self.vm.valid_coins:
        btn = QPushButton(f"{coin}元")
        if coin in [1, 0.5]:
            btn.clicked.connect(lambda _, c=coin:
self.send_request(f"insert_coin@{c}"))
        else:
            btn.clicked.connect(lambda _, c=coin:
self.send_request(f"insert_bill@{c}"))
        coin_row.addWidget(btn)
    control_layout.addLayout(coin_row)
    ...
```

TC2.1.1: insert_coin_connect -> matched by T2.1.1 test_insert_coin_binding
TC2.1.2: insert_bill_connect -> matched by T2.1.1 test_insert_coin_binding

```
def setup_admin_interface(self):
    ...
    self.coin_labels = {}
    for i, coin in enumerate(sorted(self.vm.valid_coins, reverse=True)):
        label = QLabel(f"{coin}元:")
        count_label = QLabel(str(self.vm.coin_stock[coin]))
        refill_btn = QPushButton("补充")
        refill_btn.setObjectName(f"refill_{coin}")
        if coin in [1, 0.5]:
            refill_btn.clicked.connect(lambda _, c=coin:
self.send_request(f"add_coin@{c}"))
        else:
            refill_btn.clicked.connect(lambda _, c=coin:
self.send_request(f"add_bill@{c}"))
        withdraw_btn = QPushButton("取钱")
        withdraw_btn.setObjectName(f"withdraw_{coin}")
        if coin in [1, 0.5]:
            withdraw_btn.clicked.connect(lambda _, c=coin:
self.send_request(f"remove_coin@{c}"))
        else:
            withdraw_btn.clicked.connect(lambda _, c=coin:
self.send_request(f"remove_bill@{c}"))
```

```
            stock_group.addWidget(label, i, 3)
            stock_group.addWidget(count_label, i, 4)
            stock_group.addWidget(refill_btn, i, 5)
            stock_group.addWidget(withdraw_btn, i, 6)
            self.coin_labels[coin] = count_label
        ...
```

TC2.2.1: add_coin_connect -> matched by T2.2.1 test_refill_coin_binding TC2.2.2: add_bill_connect -> matched by T2.2.1 test_refill_coin_binding TC2.2.3: remove_coin_connect -> matched by T2.3.1 test_withdraw_binding TC2.2.4: remove_bill_connect -> matched by T2.3.1 test_withdraw_binding

```
def check_timeout(self):
    if self.vm.check_timeout():
        self.logout_succeeded()
        self.update_display()
    remaining = self.vm.timeout - (time.time() - self.vm.last_operation_time)
    self.user_timer_label.setText(f"操作剩余时间: {max(0, int(remaining))}秒")
```

TC2.3.1：time_in_display -> matched by T2.4.1 test_check_timeout_diaplay
TC2.3.2: time_out_display -> matched by T2.4.2 test_check_timein_diaplay

```
def update_display(self):
    ...
    alerts = []
    self.vm.check_alerts()
    if self.vm.alerts["low_stock"]:
        alerts.append(f"需补货: {', '.join(self.vm.alerts['low_stock'])}")
    if self.vm.alerts["low_coins"]:
        alerts.append(f"需补硬币: {', '.join(map(str, self.vm.alerts['low_coins']))}
元")
    if self.vm.alerts["low_bills"]:
        alerts.append(f"需补纸币: {', '.join(map(str, self.vm.alerts['low_bills']))}
元")
    if self.vm.alerts["full_cash"]:
        alerts.append(f"需取钱: {', '.join(map(str, self.vm.alerts['full_cash']))}
元")
    self.alert_label.setText("\n".join(alerts))

    if self.admin_panel.isVisible():
        self.admin_panel.repaint()
```

TC2.4.1: low_stock_alert -> matched by T2.5.1 test_update_display TC2.4.2: low_coin_alert -> matched by T2.5.1 test_update_display TC2.4.3: low_bill_alert -> matched by T2.5.1 test_update_display TC2.4.4: full_cash_alert -> matched by T2.5.1 test_update_display

Remaining unit tests are functional tests: vm functional tests: test_check_alerts ui functional tests: starts with test_login_succeed and ends with test_user_failed

Overall we have 58 branches in my vending machine coding, and we match 57 of them, branch coverage is 98.3%. The only one we didn't not match is TC1.6.4: no_required_coins, and we prevent it from happenning through functIon can_give_change(self) in vm, called by process_payment(self) in vm and data structure insert_stock in vm. The former function helps vm determine whether it can make changes, if it can't it will refund the inserted coins. The latter structure stores insert coins and it will not be added into coin_stock until purchase is successful or timeout.

Integration tests

Input Partition

vaild inputs:

- TCOND1：insert_coin@i for i in [1, 0.5]
- TCOND2：insert_bill@i for i in [100, 50, 20, 10, 5]
- TCOND3：select_product@i for i in [1, 2, 3, 4, 5] and vm.prodcuts[i].stock > 0
- TCOND4：deselect_product@i for i in [1, 2, 3, 4, 5] and vm.selected_products[i] > 0
- TCOND5：purchase for sufficient inserted money and can give change
- TCOND6: refund_all for remaining money in vm
- TCOND7：login@123456
- TCOND8：logout in admin mode
- TCOND9：add_product@i for i in [1, 2, 3, 4, 5] in admin mode
- TCOND10：add_coin@i for i in [1, 0.5] in admin mode
- TCOND11：add_bill@i for i in [100, 50, 20, 10, 5] in admin mode
- TCOND12：remove_coin@i for i in [1, 0.5] in admin mode
- TCOND13：remove_bill@i for i in [100, 50, 20, 10, 5] in admin mode

invaild inputs:

- TCOND14: insert_coin@i for i not in [1, 0.5]
- TCOND15: insert_bill@i for i not in [100, 50, 20, 10, 5]
- TCOND16：insert_coin in admin mode
- TCOND17：insert_bill in admin mode

- TCOND18: select_product@i for i not in [1, 2, 3, 4, 5]
- TCOND19: select_product@i for i in [1, 2, 3, 4, 5] and vm.prodcuts[i].stock <= 0
- TCOND20: select product in admin mode
- TCOND21: deselect_product@i for i not in [1, 2, 3, 4, 5]
- TCOND22：deselect_product@i for i in [1, 2, 3, 4, 5] and vm.selected_products[i] <= 0
- TCOND23: deselect product in admin mode
- TCOND24: refund_all for no remaining money in vm
- TCOND25: refund_all in admin mode
- TCOND26: purchase for sufficient inserted money but can't give change
- TCOND27：purchase for insufficient inserted money
- TCOND28：purchase for no selected products
- TCOND29：purchase in admin mode
- TCOND30: login@wrong password
- TCOND31: login in admin mode
- TCOND32：logout in user mode
- TCOND33：add_product@i for i not in [1, 2, 3, 4, 5]
- TCOND34：add_product@i in user mode
- TCOND35：add_coin@i for i not in [1, 0.5]
- TCOND36：add_coin@i in user mode
- TCOND37：add_bill@i for i not in [100, 50, 20, 10, 5]
- TCOND38：add_bill@i in user mode
- TCOND39：remove_coin@i for i not in [1, 0.5]
- TCOND40：remove_coin@i in user mode
- TCOND41：remove_bill@i for i not in [100, 50, 20, 10, 5]
- TCOND42：remove_bill@i in user mode

Output Partition

- TCOND43: 投币成功 induced by TCOND1, TCOND2
- TCOND44: 请先退出管理员模式 induced by TCOND16, TCOND17, TCOND20, TCOND23, TCOND25, TCOND29
- TCOND45: 商品已添加 induced by TCOND3
- TCOND46: 商品选择失败 induced by TCOND19
- TCOND47: 商品已移除 induced by TCOND4
- TCOND48: 商品移除失败 induced by TCOND22
- TCOND49: 当前没有可退金额 induced by TCOND24
- TCOND50: 退币成功！退还：money元 induced by TCOND6

- TCOND51: 支付成功！当前余额：money元 induced by TCOND5
- TCOND52: 请先选购商品 induced by TCOND28
- TCOND53: 金额不足，还需money元 induced by TCOND27
- TCOND54: 零钱不足，请先退币 induced by TCOND26
- TCOND55: 登陆成功in user panel 管理员模式已激活in admin panel induced by TCOND7
- TCOND56: 登陆失败，请重试 induced by TCOND30
- TCOND57: 请选择商品 induced by TCOND8
- TCOND58: 商品补货成功 induced by TCOND9
- TCOND59: 钱币补充成功 induced by TCOND10
- TCOND60: 钞票补充成功 induced by TCOND11
- TCOND61: 已取钱 induced by TCOND12, TCOND13
- TCOND62: 请先进入管理员模式 induced by TCOND32, TCOND34, TCOND36, TCOND38, TCOND40, TCOND42

```
def test_login_admin_success(self)
```

covered: TCOND7, TCOND55, TCOND2, TCOND6

```
def test_login_admin_failed(self)
```

covered: TCOND29, TCOND56

```
def test_logout_admin(self)
```

covered: TCOND8, TCOND57

```
def test_admin_restock_product_succeeded(self)
```

covered: TCOND9, TCOND58

```
def test_admin_restock_product_falied(self)
```

covered: TCOND34, TCOND62

```
def test_admin_refill_coin_succeeded(self)
```

covered: TCOND10, TCOND11, TCOND59, TCOND60

```
def test_admin_refill_coin_falied(self)
```

covered: TCOND36, TCOND38, TCOND62

```
def test_admin_withdraw_succeeded(self)
```

covered: TCOND12, TCOND13, TCOND61

```
def test_admin_withdraw_falied(self)
```

covered: TCOND40, TCOND42, TCOND62

```
def test_insert_coin_button_succeeded(self)
```

covered: TCOND1, TCOND2, TCOND43

```
def test_insert_coin_button_failed(self)
```

covered: TCOND16, TCOND17, TCOND44

```
def test_return_money_succeeded(self)
```

covered: TCOND6, TCOND50

```
def test_return_money_failed_no_refund(self)
```

covered: TCON24, TCOND49

```
def test_return_money_failed_admin(self)
```

covered: TCOND25, TCOND44

```
def test_select_product_success(self)
```

covered: TCOND3, TCOND45

```
def test_select_product_failed_no_product(self)
```

covered: TCOND19, TCOND46

```
def test_select_product_failed_in_admin(self)
```

covered: TCOND20, TCOND44

```
def test_deselect_product_success(self)
```

covered: TCOND4, TCOND47

```
def test_deselect_product_fail_no_product(self)
```

covered: TCOND22, TCOND48

```
def test_deselect_product_fail_in_admin(self)
```

covered: TCOND23, TCOND44

```
def test_purchase_success(self)
```

covered: TCOND5, TCOND51

```
def test_purchase_fail_insufficient_funds(self)
```

covered: TCOND27, TCOND53

```
def test_purchase_fail_no_change(self)
```

covered: TCOND26, TCOND54

```
def test_purchase_fail_in_admin_mode(self)
```

covered: TCOND29, TCOND44

```
def test_purchase_no_product(self)
```

covered: TCOND28, TCOND52

All testcases come from integration_test.py. We cover 51 TCONDs in all 62 TCONDs. Those uncovered TCONDs are 14, 15, 18, 21, 31, 32, 33, 35, 37, 39, 41. All of them are out of range inputs like invalid insert money, invalid product index. These TCONDs can't be tested through UItest since we actually don't give such button. In API test, these API instructions will return false.

System tests

Common workflows

For customers:

- Operation 1: Select product that he/she wants and available by clicking the "+" button on the right side of the product. The selected product will be displayed in cart list.
- Operation 2: Insert enough money. The amount of money will be displayed in the left down part of the panel.
- Operation 3: Click purchase button

- Operation 4: Click refund button Operation 1 and Operation 2 are in no particular order. After Operation 3, customers can go back to Operation 1 and buy more things. Also, customers can deselect product by clicking the "-" button on the right side of "+" button. The time between each operation should be less than 300 seconds. Once the time is larger than 300 seconds, the cart list will be cleaned and the remaining money will not be returned.

For managers:

- Operation 1: Log in and enter the admin mode.
- Operation 2: Refill coins or bills.
- Operation 3: Restore products.
- Operation 4: Withdraw coins or bills.
- Operation 5: Log out. Operation 2, Operation 3 and Operation 4 are in no particular order.

Rare workflows

Step:

- Insert 100
- Add product 0
- Log in
- withdraw 100
- purchase Links to the admin withdraw. For our implement, every withdraw operation is same to change the number in coin_stock. Thus there may be such circumstance: customer inserts some 100, then admin log in and withdraw 100, then customer will lose all 100 and he cannot refund. Our first mitigation is to refund first if login succeed. Then customer cannot purchase after admin log out. They must insert coin again; thus we have our second mitigation: add insert_stock to restore the inserted money, and only after purchase and time out it will be added into coin_stock. Also, it tests that in commom case customer can get right change and right product. testcase(UItest.py):

```
def test_rare1(self)
```

Step

- Empty coin_stock(set the base environment)
- Insert 100
- Add product 0
- purchase Links to the coins/bills are not restocked. testcase(UItest.py):

```
def test_rare2(self)
```

Step

- Insert 100
- Add product 4
- purchase Links to the products are not restocked. testcase(UItest.py):

```
def test_rare3(self)
```

Step

- Log in
- Insert 100
- Wait for 300s Links to admin forgot to log out. testcase(UItest.py):

```
def test_rare4(self)
```

Model checking

Risk management

For vending machines, risks are these:

Customer cannot get right change

- The machine may not have enough coins/bills to provide the exact change for a purchase.

- The risk will happen after purchase. If coins/bills are not restocked in appropriate time, or admin withdraws too many coins/bills, it may happen.
- It may happen occasionally if restocking properly(We assume that circumstance).
- Severity is high since it causes bad user experience, even may leads to complaints and finanicial loss.
- The overall refund process is that first check whether vm can give change(by calling function can_give_change()) based on current cart list and the amount invested during purchase. If and only if change can be made, the purchase will succeed and end this operation. Otherwise, the machine will reject the purchase and ask the customer to refund first. The above process can be repeated without affecting its functionally.

Customer cannot get right product

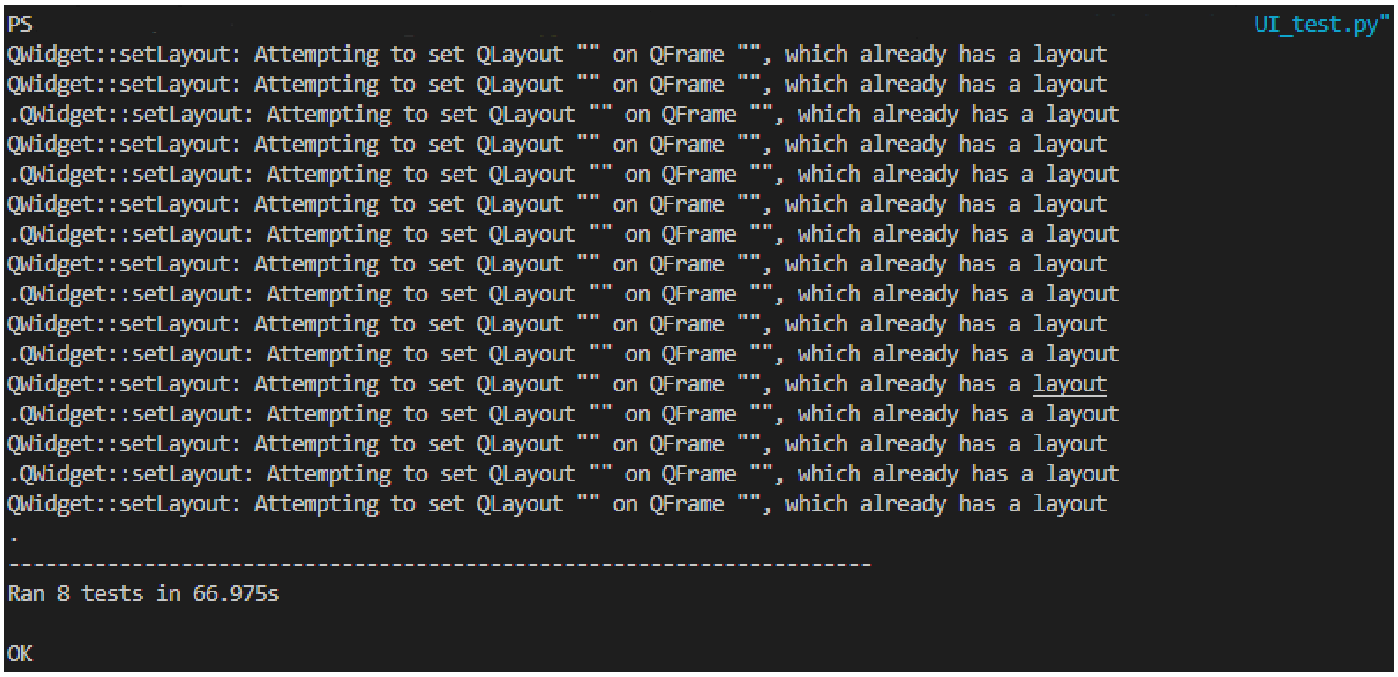
- The machine may not give the correct product after purchase or may accept selection of products which has sold out.
- The risk will happen during selection and purchase. If products are not restocked in appropriate time, or the software runs wrongly, it may happen.
- It seldom happens if restocking properly(We assume that circumstance).
- Severity is high since it causes bad user experience, even may leads to complaints and finanicial loss.
- Selection will succeed only when the stock of selected product is larger than 0. Deselection will succeed only when the amount of selected products is larger than 0. Purchase will decrease the stock of products correctly and it will be displayed on the UI board.

Admin features are not avaliable to customers

- Users (non-admin) may be able to access admin features by mistake.
- The risk will happen if admin forgot to log out or UI control wrongly exposes admin-only buttons.
- It seldom happens since it depends on code bug and the wrong operation of admin.
- Severity is very high since it causes large amount of finanicial loss.

- First only when user login successfully the admin panel will display on UI panel. All admin operations will first check whether its state is admin state. Also, we design a time check. If admin doesn't operate for larger than 300 seconds, machine will automatically logout.

Result



```
PS                                                                                        UI_test.py"
QWidget::setLayout: Attempting to set QLayout "" on QFrame "", which already has a layout
QWidget::setLayout: Attempting to set QLayout "" on QFrame "", which already has a layout
.QWidget::setLayout: Attempting to set QLayout "" on QFrame "", which already has a layout
QWidget::setLayout: Attempting to set QLayout "" on QFrame "", which already has a layout
.QWidget::setLayout: Attempting to set QLayout "" on QFrame "", which already has a layout
QWidget::setLayout: Attempting to set QLayout "" on QFrame "", which already has a layout
.QWidget::setLayout: Attempting to set QLayout "" on QFrame "", which already has a layout
QWidget::setLayout: Attempting to set QLayout "" on QFrame "", which already has a layout
.QWidget::setLayout: Attempting to set QLayout "" on QFrame "", which already has a layout
QWidget::setLayout: Attempting to set QLayout "" on QFrame "", which already has a layout
.QWidget::setLayout: Attempting to set QLayout "" on QFrame "", which already has a layout
QWidget::setLayout: Attempting to set QLayout "" on QFrame "", which already has a layout
.QWidget::setLayout: Attempting to set QLayout "" on QFrame "", which already has a layout
QWidget::setLayout: Attempting to set QLayout "" on QFrame "", which already has a layout
.QWidget::setLayout: Attempting to set QLayout "" on QFrame "", which already has a layout
QWidget::setLayout: Attempting to set QLayout "" on QFrame "", which already has a layout
.
----------------------------------------------------------------------
Ran 8 tests in 66.975s

OK
```