Group 7

Author: Wenlin Zhu

Software Validatiion

Banking System

Table of Contents

[System Architecture 4](#_Toc1911900784)

[T1: Unit Test 4](#_Toc518666627)

[T1.1: Processor Unit Test 4](#_Toc593876816)

[T1.1.1: Test create\_account( ) 4](#_Toc1931138903)

[T1.1.2: Test close\_account ( ) 5](#_Toc1731185610)

[T1.1.3: Test deposit\_cash ( ) 6](#_Toc1928097714)

[T1.1.4: Test withdraw\_cash ( ) 6](#_Toc10045968)

[T1.1.5: Test open\_app ( ) 7](#_Toc1338859710)

[T1.1.6: Test close\_app ( ) 8](#_Toc881492704)

[T1.1.7: Test log\_out ( ) 8](#_Toc1905679122)

[T1.1.8: Test transfer\_money ( ) 9](#_Toc1177892096)

[T1.1.9: Test change\_passport ( ) 10](#_Toc1328199426)

[T1.1.10: Test query ( ) 11](#_Toc2102725864)

[T1.1.11: Test return\_card ( ) 12](#_Toc1522701216)

[T1.1.12: Test insert\_card ( ) 13](#_Toc476716013)

[T1.1.13: Test check\_password ( ) 13](#_Toc2052027181)

[T1.2: DataBase Unit Test 14](#_Toc1980943894)

[T1.2.1: Test create\_account ( ) 14](#_Toc1285047017)

[T1.2.2: Test close\_account ( ) 15](#_Toc542176840)

[T1.2.3: Test check\_password ( ) 15](#_Toc593035659)

[T1.2.4: Test update\_balance ( ) 16](#_Toc678715086)

[T1.2.5: Test check\_is\_valid ( ) 16](#_Toc1878801185)

[T1.2.6: Test change\_password ( ) 17](#_Toc411970807)

[T1.2.7: Test check\_enough\_balance ( ) 18](#_Toc506075321)

[T1.2.8: Test reset ( ) 18](#_Toc1572677927)

[T1.2.9: Test query ( ) 19](#_Toc769191813)

[T1.3: APPUI Unit Test 20](#_Toc2102729798)

[T1.3.1: Test handle\_login ( ) 20](#_Toc1588819954)

[T1.3.2: Test handle\_transfer ( ) 21](#_Toc1485300080)

[T1.3.3: Test handle\_query ( ) 22](#_Toc1088531832)

[T1.3.4: Test handle\_changePin ( ) 23](#_Toc541311631)

[T1.3.5: Test handle\_logout ( ) 24](#_Toc1083853525)

[T1.3.6: Test back\_to\_select ( ) 24](#_Toc1369900821)

[T1.3.7: Test compusary\_log\_out ( ) 25](#_Toc750919060)

[T1.3.8: Test set\_app\_id ( ) 25](#_Toc2082731648)

[T1.3.9: Test handle\_close\_app ( ) 26](#_Toc487361836)

[T1.3.10: Test closeEvent ( ) 26](#_Toc587747994)

[T1.4: ATMUI Unit Test 27](#_Toc2003242605)

[T1.4.1: Test checkPIN ( ) 27](#_Toc249844569)

[T1.4.2: Test handle\_insert\_card ( ) 27](#_Toc807141298)

[T1.4.3: Test insert\_card ( ) 28](#_Toc2117081034)

[T1.4.4: Test handle\_return\_card ( ) 29](#_Toc124391295)

[T1.4.5: Test handle\_change\_pin ( ) 29](#_Toc1142906534)

[T1.4.6: Test handle\_query ( ) 30](#_Toc1736378170)

[T1.4.7: Test handle\_transfer ( ) 31](#_Toc1152624107)

[T1.4.8: Test handle\_deposit\_cash ( ) 33](#_Toc1850870409)

[T1.4.9: Test handle\_withdraw\_cash ( ) 34](#_Toc1278337268)

[T1.4.10: Test withdraw\_cash ( ) 35](#_Toc1588058688)

[T1.4.11: Test handle\_create\_account ( ) 36](#_Toc1575604300)

[T1.4.12: Test create\_account ( ) 37](#_Toc560618943)

[T1.4.13: Test go\_to\_selectPage ( ) 38](#_Toc1311815612)

[T1.4.14: Test handle\_close\_account ( ) 38](#_Toc1617870782)

[T1.4.15: Test collectCash ( ) 39](#_Toc116294760)

[T1.4.16: Test ATM\_cashupdate ( ) 40](#_Toc355912550)

[T1.5: ATM Unit Test 40](#_Toc1080270955)

[T1.5.1: Test deposit ( ) 40](#_Toc1287188947)

[T1.5.2: Test withdraw ( ) 41](#_Toc34372351)

[T1.5.3: Test check\_balance ( ) 41](#_Toc23002214)

[T2: Integration Test 42](#_Toc51154238)

[T2.1: Processor + Database + APPUI Integration 42](#_Toc755819266)

[T2.1.1: Test Change Pin 42](#_Toc688631657)

[T2.1.2: Test Transfer 42](#_Toc1042885516)

[T2.2: Processor + Database + ATMUI + ATM Integration 43](#_Toc15340598)

[T2.2.1: Test Account Operation 43](#_Toc131392946)

[T2.2.2: Test Money Operation 43](#_Toc708054306)

[T2.3: Processor + Database + APPUI + ATMUI + ATM Integration 44](#_Toc1061832915)

[T2.3.1: Test Money Operation 44](#_Toc636695835)

[T3: Functional Test 45](#_Toc35885844)

[T3.1: Use Case “Use ATM Machine” 45](#_Toc1837958948)

[T3.1.1: Test “ATM Deposit Cash” 45](#_Toc1171260588)

[T3.1.2: Test “ATM Withdraw Cash” 45](#_Toc1541594114)

[T3.1.3: Test “ATM Transfer” 45](#_Toc182072943)

[T3.1.4: Test “ATM Query Account” 46](#_Toc2083239673)

[T3.2: Use Case “Use Banking APP” 46](#_Toc435803423)

[T3.2.1: Test “APP Reset Information” 46](#_Toc1628894091)

[T3.2.2: Test “APP Transfer” 47](#_Toc701455481)

[T3.2.3: Test “APP Query Account” 47](#_Toc1824530784)

[T4: Model Checking 47](#_Toc969891175)

[T4.1: Introduction 47](#_Toc1560097495)

[T4.2: Assumptions 48](#_Toc1930752242)

[T4.3: Banking System Model 48](#_Toc1675025124)

[T4.3.1: ATM 48](#_Toc784130545)

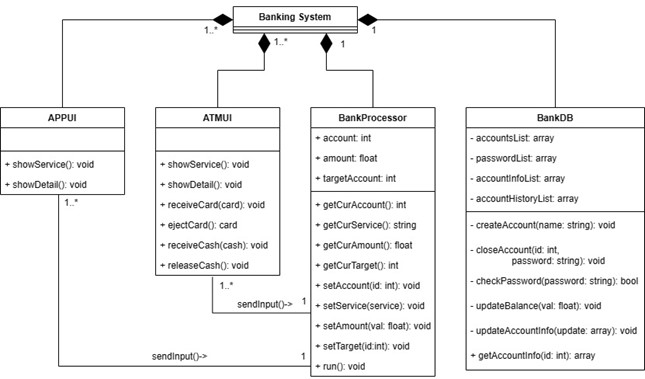
[T4.3.2: APP 49](#_Toc1922411823)

[T4.3.3: Processor 50](#_Toc1084040046)

[T4.3.4: Check Properties 51](#_Toc209791974)

## System Architecture

The system architecture is shown below:



## T1: Unit Test

### T1.1: Processor Unit Test

This section provides information of unit tests we made for every function with statement coverage, branch coverage and condition coverage criteria. Testing cases with runnable test functions are provided in every test, you can find in corresponding files.

#### T1.1.1: Test create\_account( )

def create\_account(self, password):

if len(password)!= 6:

return "failed@create\_acount"

account\_id = ''.join([str(random.randint(0, 9)) for \_ in range(10)])

while check\_account\_exist(account\_id):

account\_id = ''.join([str(random.randint(0, 9)) for \_ in range(10)])

create\_account(account\_id, password)

self.current\_account = account\_id

self.account\_available = True

return "account\_created@" + account\_id

* Coverage Criteria: Branch coverage
* Test case

|  |  |  |
| --- | --- | --- |
|  | Test Case T1.1.1.1 | Test Case T1.1.1.2 |
| Coverage Item | Tcover1.1.1.1 | Tcover1.1.1.2 |
| Input | password = 1234 | password = 123456 |
| State | procrssor = Processor()  db = user\_database  db.accounts = {} | procrssor = Processor()  db = user\_database  db.accounts = {} |
| Expected Output | failed@created\_account  db.accounts = {} | accounted\_created  the new account information was added to the database |

* Test coverage: 2/2=100%
* Test result: 2 passed

#### T1.1.2: Test close\_account ( )

def close\_account(self):

if self.account\_available == False:

return "failed@close\_account"

result = query(self.current\_account)

if result['balance'] != 0:

return "failed@close\_account"

close\_account(self.current\_account)

self.account\_available = False

return "account\_closed@" + self.current\_account

* Coverage Criteria: Branch coverage
* Test case

|  |  |  |  |
| --- | --- | --- | --- |
|  | Test Case T1.1.2.1 | Test Case T1.1.2.2 | Test Case T1.1.2.3 |
| Coverage Item | Tcover1.1.2.1 | Tcover1.1.2.2 | Tcover1.1.2.3 |
| Input | ------ | ------ | ------ |
| State | procrssor = Processor()  processor.account\_available=False  db = user\_database  db.accounts = {(2023123456,111111,0, AccountState.Valid, creation\_time)} | procrssor = Processor()  processor.account\_available  =True  db = user\_database  db.accounts = {(2023123456,111111,500, AccountState.Valid, creation\_time)} | procrssor = Processor()  processor.account\_available  =True  db = user\_database  db.accounts = {(2023123456,111111,0, AccountState.Valid, creation\_time)} |
| Expected Output | failed@close\_account  db.accounts = {(2023123456,111111,0, AccountState.Valid, creation\_time)} | failed@close\_account  db.accounts = {(2023123456,111111,500, AccountState.Valid, creation\_time)} | account\_closed@2023123456  db.accounts = {(2023123456,111111,0, AccountState.Closed, creation\_time)} |

* Test coverage: 3/3=100%
* Test result: 3 passed

#### T1.1.3: Test deposit\_cash ( )

def deposit\_cash(self, deposit\_amount):

if self.account\_available and deposit\_amount.isdigit():

update\_balance(self.current\_account, int(deposit\_amount))

return "cash\_deposited@" + deposit\_amount

return "failed@deposit\_cash"

* Coverage Criteria: Branch coverage
* Test case

|  |  |  |
| --- | --- | --- |
|  | Test Case T1.1.3.1 | Test Case T1.1.3.2 |
| Coverage Item | Tcover1.1.3.1 | Tcover1.1.3.2 |
| Input | deposit\_amount = abc | deposit\_amount = 100 |
| State | procrssor = Processor()  db = user\_database  db.accounts = {(2023123456,111111,500, AccountState.Valid, creation\_time)} | procrssor = Processor()  db = user\_database  db.accounts = {(2023123456,111111,500, AccountState.Valid, creation\_time)} |
| Expected Output | failed@deposit\_cash  db.accounts = {(2023123456,111111,500, AccountState.Valid, creation\_time)} | cash\_deposited@600  db.accounts = {(2023123456,111111,600, AccountState.Valid, creation\_time)} |

* Test coverage: 2/2=100%
* Test result: 2 passed

#### T1.1.4: Test withdraw\_cash ( )

def withdraw\_cash(self, withdraw\_amount, password):

if self.account\_available and withdraw\_amount.isdigit() and check\_password(self.current\_account, password) and check\_enough\_balance(self.current\_account, int(withdraw\_amount)):

update\_balance(self.current\_account, -int(withdraw\_amount))

return "cash\_withdrawn@" + withdraw\_amount

return "failed@withdraw\_cash"

* Coverage Criteria: Branch coverage
* Test case

|  |  |  |
| --- | --- | --- |
|  | Test Case T1.1.4.1 | Test Case T1.1.4.2 |
| Coverage Item | Tcover1.1.4.1 | Tcover1.1.4.2 |
| Input | withdraw\_amount=abc  password=111111 | withdraw\_amount=1000  password=111111 |
| State | procrssor = Processor()  db = user\_database  db.accounts = {(2023123456,111111,500, AccountState.Valid, creation\_time)} | procrssor = Processor()  db = user\_database  db.accounts = {(2023123456,111111,500, AccountState.Valid, creation\_time)} |
| Expected Output | failed@withdraw\_cash  db.accounts = {(2023123456,111111,500, AccountState.Valid, creation\_time)} | cash\_withdraw@1000  db.accounts = {(2023123456,111111,1500, AccountState.Valid, creation\_time)} |

* Test coverage: 2/2=100%
* Test result: 2 passed

#### T1.1.5: Test open\_app ( )

def open\_app(self):

self.app\_count += 1

app\_id = str(self.app\_count)

self.apps\_account[app\_id] = None

self.apps\_state[app\_id] = "open"

return "app\_opened#" + app\_id

Coverage Criteria: Statement coverage

* Test case

|  |  |
| --- | --- |
|  | Test Case T1.1.5.1 |
| Coverage Item | Tcover1.1.5.1 |
| Input | ------ |
| State | procrssor = Processor( )  processor.app\_count = 0; |
| Expected Output | app\_opened#1  processor.app\_count = 1  processor.apps\_account[“1”] = None  processor.apps\_state[“1”] = “open” |

* Test coverage: 1/1=100%
* Test result: 1 passed

#### T1.1.6: Test close\_app ( )

def close\_app(self, app\_id):

if app\_id in self.apps\_account and self.apps\_state.get(app\_id) != "close":

self.apps\_state[app\_id] = "close"

self.close\_app\_signal.emit(app\_id)

return "app\_closed#" + app\_id

return "failed@close\_app#" + app\_id

Coverage Criteria: Branch coverage

* Test case

|  |  |  |
| --- | --- | --- |
|  | Test Case T1.1.6.1 | Test Case T1.1.6.2 |
| Coverage Item | Tcover1.1.6.1 | Tcover1.1.6.2 |
| Input | app\_id = “1” | app\_id = “2” |
| State | procrssor = Processor( )  processor.app\_count = 1  processor.apps\_state[“1”] = “open” | procrssor = Processor( )  processor.app\_count = 1  processor.apps\_state[“1”] = “open” |
| Expected Output | app\_closed#1  processor.app\_count = 1  processor.apps\_state[“1”] = “close” | failed@close\_app#2  processor.app\_count = 1  processor.apps\_state[“1”] = “open” |

* Test coverage: 2/2=100%
* Test result: 2 passed

#### T1.1.7: Test log\_out ( )

def log\_out(self, app\_id):

account\_id = self.apps\_account.get(app\_id)

if not app\_id.isdigit() or not account\_id or self.apps\_state.get(app\_id) != "log\_in":

return "failed@log\_out#" + app\_id

self.apps\_state[app\_id] = "open"

return "logged\_out@" + account\_id + "#" + app\_id

Coverage Criteria: Branch coverage

* Test case

|  |  |  |
| --- | --- | --- |
|  | Test Case T1.1.8.1 | Test Case T1.1.8.2 |
| Coverage Item | Tcover1.1.8.1 | Tcover1.1.8.2 |
| Input | app\_id = “a” | app\_id = “1” |
| State | procrssor = Processor( )  processor.apps\_state[“1”] = “log\_in” | procrssor = Processor( )  processor.apps\_state[“1”] = “log\_in” |
| Expected Output | failed@log\_out#a  processor.apps\_state[“1”] = “log\_in” | log\_out@account\_id#1  processor.apps\_state[“1”] = “open” |

* Test coverage: 2/2=100%
* Test result: 2 passed

#### T1.1.8: Test transfer\_money ( )

def transfer\_money(self, receiver\_id, transfer\_amount, app\_id=None):

account\_id = self.current\_account if app\_id is None else self.apps\_account.get(app\_id)

if not check\_is\_valid(receiver\_id) or not check\_enough\_balance(account\_id, int(transfer\_amount)) or account\_id == receiver\_id:

return "failed@transfer\_money" + (f”#{app\_id}” if app\_id else "")

update\_balance(account\_id, -int(transfer\_amount))

update\_balance(receiver\_id, int(transfer\_amount))

return "money\_transfered@" + transfer\_amount + (f"#{app\_id}" if app\_id else "")

Coverage Criteria: Branch coverage

* Test case

|  |  |  |
| --- | --- | --- |
|  | Test Case T1.1.9.1 | Test Case T1.1.49.2 |
| Coverage Item | Tcover1.1.9.1 | Tcover1.1.9.2 |
| Input | receiver\_id = 2023123457  transfer\_amount=1000 | receiver\_id = 2023123457  transfer\_amount=500 |
| State | procrssor = Processor( )  db = user\_database  db.accounts = {(2023123456,111111,500, AccountState.Valid, creation\_time),(2023123457,123456,500, AccountState.Valid, creation\_time)} | procrssor = Processor( )  db = user\_database  db.accounts = {(2023123456,111111,500, AccountState.Valid, creation\_time),(2023123457,123456,500, AccountState.Valid, creation\_time)} |
| Expected Output | failed@transfer\_money  db = user\_database  db.accounts = {(2023123456,111111,500, AccountState.Valid, creation\_time),(2023123457,123456,500, AccountState.Valid, creation\_time)} | money\_transfered@500  db = user\_database  db.accounts =  {(2023123456,111111,0, AccountState.Valid, creation\_time),(2023123457,123456,1000, AccountState.Valid, creation\_time)} |

* Test coverage: 2/2=100%
* Test result: 2 passed

#### T1.1.9: Test change\_passport ( )

def change\_password(self, new\_password, app\_id=None):

account\_id = self.current\_account if app\_id is None else self.apps\_account.get(app\_id)

if app\_id and (not app\_id.isdigit() or not account\_id or self.apps\_state.get(app\_id) != "log\_in"):

return "failed@change\_password#" + app\_id

if not self.account\_available and not app\_id:

return "failed@change\_password"

if (not new\_password.isdigit()) or new\_password == "" or len(new\_password)!=6:

return "failed@change\_password"

change\_password(account\_id, new\_password)

return "password\_changed@" + (app\_id if app\_id else "")

Coverage Criteria: Branch coverage

* Test case

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Test Case T1.1.10.1 | Test Case T1.1.10.2 | Test Case T1.1.10.3 | Test Case T1.1.10.4 |
| Coverage Item | Tcover1.1.10.1 | Tcover1.1.10.2 | Tcover1.1.10.3 | Tcover1.1.10.4 |
| Input | new\_password = 654321  app\_id=a | new\_password = 654321 | new\_password = 653ab  app\_id=1 | new\_password = 654321  app\_id=1 |
| State | procrssor = Processor( )  processor.app\_count = 1  processor.apps\_state[“1”] = “open”  processor.account\_available=True  db =user\_database  db.accounts = {(2023123456,111111,500, AccountState.Valid, creation\_time)} | procrssor = Processor( )  processor.app\_count = 1  processor.apps\_state[“1”] = “log\_in”  processor.account\_available=False  db = user\_database  db.accounts = {(2023123456,111111,500, AccountState.Valid, creation\_time)} | procrssor = Processor( )  processor.app\_count = 1  processor.apps\_state[“1”] = “log\_in”  processor.account\_available=True  db =user\_database  db.accounts = {(2023123456,111111,500, AccountState.Valid, creation\_time)} | procrssor = Processor( )  processor.app\_count = 1  processor.apps\_state[“1”] = “log\_in”  processor.account\_available=True  db =user\_database  db.accounts = {(2023123456,111111,500, AccountState.Valid, creation\_time)} |
| Expected Output | failed@change\_password#a  processor.app\_count = 1  processor.apps\_state[“1”] = “open”  processor.account\_available=True  db =user\_database  db.accounts = {(2023123456,111111,500, AccountState.Valid, creation\_time)} | failed@change\_password  processor.app\_count = 1  processor.apps\_state[“1”] = “open”  processor.account\_available=False  db = user\_database  db.accounts = {(2023123456,111111,500, AccountState.Valid, creation\_time)} | failed@change\_password  processor.app\_count = 1  processor.apps\_state[“1”] = “log\_in”  processor.account\_available=True  db =user\_database  db.accounts = {(2023123456,111111,500, AccountState.Valid, creation\_time)} | password\_changed@1  processor.app\_count = 1  processor.apps\_state[“1”] = “log\_in”  processor.account\_available=True  db =user\_database  db.accounts = {(2023123456,654321,500, AccountState.Valid, creation\_time)} |

* Test coverage: 4/4=100%
* Test result: 4 passed

#### T1.1.10: Test query ( )

def query(self,app\_id = None):

account\_id = self.current\_account if app\_id is None else self.apps\_account.get(app\_id)

result = query(account\_id)

results = [result,app\_id]

self.query\_result\_signal.emit(results)

if(result== None):

if app\_id == None:

return "failed@query"

else:

return "failed@query#" + app\_id

if app\_id == None:

return "query\_showed"

else:

return ("query\_showed" + app\_id)

Coverage Criteria: Branch coverage

* Test case

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Test Case T1.1.11.1 | Test Case T1.1.11.2 | Test Case T1.1.11.3 | Test Case T1.1.11.4 |
| Coverage Item | Tcover1.1.11.1 | Tcover1.1.11.2 | Tcover1.1.11.3 | Tcover1.1.11.4 |
| Input | app\_id = 1 | ------ | app\_id = 1 | ------ |
| State | procrssor = Processor( )  processor.app\_count = 1  processor.apps\_state[“1”] = “open”  processor.account\_available=True  db =user\_database  db.accounts = {(2023123457,111111,500, AccountState.Valid, creation\_time)} | procrssor = Processor( )  processor.app\_count = 1  processor.apps\_state[“1”] = “open”  processor.account\_available=True  db =user\_database  db.accounts = {(2023123457,111111,500, AccountState.Valid, creation\_time)} | procrssor = Processor( )  processor.app\_count = 1  processor.apps\_state[“1”] = “open”  processor.account\_available=True  db =user\_database  db.accounts = {(2023123456,111111,500, AccountState.Valid, creation\_time)} | procrssor = Processor( )  processor.app\_count = 1  processor.apps\_state[“1”] = “open”  processor.account\_available=True  db =user\_database  db.accounts = {(2023123456,111111,500, AccountState.Valid, creation\_time)} |
| Expected Output | failed@query#1 | failed@query | query\_showed1 | query\_showed |

* Test coverage: 4/4=100%
* Test result: 4 passed

#### T1.1.11: Test return\_card ( )

def return\_card(self):

rmessage = "card\_returned@" + self.current\_account

self.current\_account = None

return rmessage

Coverage Criteria: Branch coverage

* Test case

|  |  |
| --- | --- |
|  | Test Case T1.1.12.1 |
| Coverage Item | Tcover1.1.12.1 |
| Input | ------ |
| State | procrssor = Processor( )  processor.current\_account = “2023123456”  db =user\_database  db.accounts = {(2023123456,111111,500, AccountState.Valid, creation\_time)} |
| Expected Output | card\_returned@2023123456  processor.current\_account = None  db.accounts = {(2023123456,111111,500, AccountState.Valid, creation\_time)} |

* Test coverage: 1/1=100%
* Test result: 1 passed

#### T1.1.12: Test insert\_card ( )

def insert\_card(self,account\_id):

if not check\_is\_valid(account\_id):

return "failed@insert\_card"

self.current\_account = account\_id

self.account\_available = True

return "card\_inserted@" + account\_id

Coverage Criteria: Branch coverage

* Test case

|  |  |  |
| --- | --- | --- |
|  | Test Case T1.1.13.1 | Test Case T1.1.13.2 |
| Coverage Item | Tcover1.1.13.1 | Tcover1.1.13.2 |
| Input | account\_id = 2023123456 | account\_id = 1234567890 |
| State | procrssor = Processor( )  processor.current\_account  = None  processor.account\_available  = False  db =user\_database  db.accounts = {(2023123456,111111,500, AccountState.Valid, creation\_time)} | procrssor = Processor( )  processor.current\_account  = None  processor.account\_available  = False  db =user\_database  db.accounts = {(2023123456,111111,500, AccountState.Valid, creation\_time)} |
| Expected Output | card\_inserted@2023123456  processor.current\_account  = 2023123456  processor.account\_available  = True  db.accounts = {(2023123456,111111,500, AccountState.Valid, creation\_time)} | failed@inserted\_card  processor.current\_account  = None  processor.account\_available  = False  db.accounts = {(2023123456,111111,500, AccountState.Valid, creation\_time)} |

* Test coverage: 2/2=100%
* Test result: 2 passed

#### T1.1.13: Test check\_password ( )

def check\_password(self,account\_id,password):

return check\_password(account\_id, password)

Coverage Criteria: Branch coverage

* Test case

|  |  |  |
| --- | --- | --- |
|  | Test Case T1.1.14.1 | Test Case T1.1.14.2 |
| Coverage Item | Tcover1.1.14.1 | Tcover1.1.14.2 |
| Input | account\_id = 1234567890  password = 123456 | account\_id = 2023123456  password = 111111 |
| State | procrssor = Processor( )  db =user\_database  db.accounts = {(2023123456,111111,500, AccountState.Valid, creation\_time)} | procrssor = Processor( )  db =user\_database  db.accounts = {(2023123456,111111,500, AccountState.Valid, creation\_time)} |
| Expected Output | False | True |

* Test coverage: 2/2=100%
* Test result: 2 passed

### T1.2: DataBase Unit Test

#### T1.2.1: Test create\_account ( )

def create\_account(account\_id, password, balance=0):

creation\_time = datetime.now().strftime('%Y-%m-%d %H:%M:%S')

cursor.execute('''

INSERT INTO accounts (account\_id, password, balance, state, creation\_time)

VALUES (?, ?, ?, ?, ?)

''', (account\_id, password, balance, AccountState.Valid, creation\_time))

conn.commit()

* Coverage Criteria: Statement coverage
* Test case

|  |  |
| --- | --- |
|  | Test Case T1.2.1.1 |
| Coverage Item | Tcover1.2.1.1 |
| Input | account\_id=2023123457  password=123456  balance=1000 |
| State | db = user\_database  db.accounts = {(2023123456,111111,500, AccountState.Valid, creation\_time)} |
| Expected Output | account\_created  db.accounts = {(2023123456,111111,500, AccountState.Valid, creation\_time), (2023123457,123456,1000, AccountState.Valid, now)} |

* Test coverage: 1/1=100%
* Test result: 1 passed

#### T1.2.2: Test close\_account ( )

def close\_account(account\_id):

cursor.execute('''

UPDATE accounts

SET state = ?

WHERE account\_id = ?

''', (AccountState.Deleted, account\_id))

conn.commit()

* Coverage Criteria: Statement coverage
* Test case

|  |  |
| --- | --- |
|  | Test Case T1.2.2.1 |
| Coverage Item | Tcover1.2.2.1 |
| Input | account\_id=2023123456 |
| State | db = user\_database  db.accounts = {(2023123456,111111,500, AccountState.Valid, creation\_time)} |
| Expected Output | db.accounts = {(2023123456,111111,500, AccountState.Closed, creation\_time)} |

* Test coverage: 1/1=100%
* Test result: 1 passed

#### T1.2.3: Test check\_password ( )

def check\_password(account\_id, password):

cursor.execute('''

SELECT \* FROM accounts

WHERE account\_id = ? AND password = ?

''', (account\_id, password))

return cursor.fetchone() is not None

* Coverage Criteria: Statement coverage
* Test case

|  |  |  |
| --- | --- | --- |
|  | Test Case T1.2.3.1 | Test Case T1.2.3.2 |
| Coverage Item | Tcover1.2.3.1 | Tcover1.2.3.2 |
| Input | account\_id=2023123456  password=111111 | account\_id=2023123456  password=111112 |
| State | db = user\_database  db.accounts = {(2023123456,111111,500, AccountState.Valid, creation\_time)} | db = user\_database  db.accounts = {(2023123456,111111,500, AccountState.Valid, creation\_time)} |
| Expected Output | True | False |

* Test coverage: 2/2=100%
* Test result: 2 passed

#### T1.2.4: Test update\_balance ( )

def update\_balance(account\_id, value):

cursor.execute('''

UPDATE accounts

SET balance = balance + ?

WHERE account\_id = ?

''', (value, account\_id))

conn.commit()

* Coverage Criteria: Statement coverage
* Test case

|  |  |
| --- | --- |
|  | Test Case T1.2.4.1 |
| Coverage Item | Tcover1.2.4.1 |
| Input | account\_id=2023123456  value=200 |
| State | db = user\_database  db.accounts = {(2023123456,111111,500, AccountState.Valid, creation\_time)} |
| Expected Output | db.accounts = {(2023123456,111111,700, AccountState.Valid, creation\_time)} |

* Test coverage: 1/1=100%
* Test result: 1 passed

#### T1.2.5: Test check\_is\_valid ( )

def check\_is\_valid(account\_id):

cursor.execute('''

SELECT state FROM accounts

WHERE account\_id = ?

''', (account\_id,))

result = cursor.fetchone()

if result:

return result[0] == AccountState.Valid

else:

return False

* Coverage Criteria: Branch coverage
* Test case

|  |  |  |  |
| --- | --- | --- | --- |
|  | Test Case T1.2.5.1 | Test Case T1.2.5.2 | Test Case T1.2.5.1 |
| Coverage Item | Tcover1.2.5.1 | Tcover1.2.5.2 | Tcover1.2.5.1 |
| Input | account\_id=2023123457 | account\_id=2023123456 | account\_id=2023123456 |
| State | db = user\_database  db.accounts = {(2023123456,111111,500, AccountState.Valid, creation\_time)} | db = user\_database  db.accounts = {(2023123456,111111,500, AccountState.Valid, creation\_time)} | db = user\_database  db.accounts = {(2023123456,111111,500, AccountState.Closed, creation\_time)} |
| Expected Output | False | True | False |

* Test coverage: 3/3=100%
* Test result: 3 passed

#### T1.2.6: Test change\_password ( )

def change\_password(account\_id,new\_password):

cursor.execute('''

UPDATE accounts

SET password = ?

WHERE account\_id = ?

''', (new\_password, account\_id))

conn.commit()

* Coverage Criteria: Statement coverage
* Test case

|  |  |
| --- | --- |
|  | Test Case T1.2.7.1 |
| Coverage Item | Tcover1.2.7.1 |
| Input | account\_id=2023123456  new\_ password =123456 |
| State | db = user\_database  db.accounts = {(2023123456,111111,50  0,AccountState.Valid, creation\_time)} |
| Expected Output | db.accounts = {(2023123456,123456,50  0,AccountState.Valid, creation\_time)} |

* Test coverage: 1/1=100%
* Test result: 1 passed

#### T1.2.7: Test check\_enough\_balance ( )

def check\_enough\_balance(account\_id,required\_balance):

cursor.execute('''

SELECT balance FROM accounts

WHERE account\_id = ?

''', (account\_id,))

result = cursor.fetchone()

if result:

balance = result[0]

return balance >= required\_balance

else:

return False

* Coverage Criteria: Branch coverage
* Test case

|  |  |  |
| --- | --- | --- |
|  | Test Case T1.2.15.1 | Test Case T1.2.15.2 |
| Coverage Item | Tcover1.2.15.1 | Tcover1.2.15.2 |
| Input | account\_id=2023123456  required\_balance = 600 | account\_id=2023123456  required\_balance=200 |
| State | db = user\_database  db.accounts = {(2023123456,111111,50  0,AccountState.Valid, creation\_time)} | db = user\_database  db.accounts = {(2023123456,111111,50  0,AccountState.Valid, creation\_time)} |
| Expected Output | False | True |

* Test coverage: 2/2=100%
* Test result: 2 passed

#### T1.2.8: Test reset ( )

def reset():

cursor.execute('DROP TABLE IF EXISTS accounts')

create\_table()

create\_account('2023123456',111111,500)

Coverage Criteria: Statement coverage

* Test case

|  |  |
| --- | --- |
|  | Test Case T1.2.9.1 |
| Coverage Item | Tcover1.2.9.1 |
| Input | ------ |
| State | db = user\_database  db.accounts = {(2023123456,111111,70  0,AccountState.Valid, creation\_time)} |
| Expected Output | db.accounts = {(2023123456,111111,50  0,AccountState.Valid, now)} |

* Test coverage: 1/1=100%
* Test result: 1 passed

#### T1.2.9: Test query ( )

def query(account\_id):

cursor.execute('''

SELECT account\_id, balance, creation\_time FROM accounts

WHERE account\_id = ?

''', (account\_id,))

result = cursor.fetchone()

if result:

return {

'account\_id': result[0],

'balance': result[1],

'creation\_time': result[2]

}

else:

return None

* Coverage Criteria: Branch coverage
* Test case

|  |  |  |
| --- | --- | --- |
|  | Test Case T1.2.10.1 | Test Case T1.2.10.2 |
| Coverage Item | Tcover1.2.10.1 | Tcover1.2.10.2 |
| Input | account\_id=2023123456 | account\_id=2023123456 |
| State | db = user\_database  db.accounts = {(2023123457,111111,50  0,AccountState.Valid, creation\_time)} | db = user\_database  db.accounts = {(2023123456,111111,50  0,AccountState.Valid, creation\_time)} |
| Expected Output | None | account\_id: 2023123456,  balance: 500,  creation\_time:creation\_time |

* Test coverage: 2/2=100%
* Test result: 2 passed

### T1.3: APPUI Unit Test

#### T1.3.1: Test handle\_login ( )

def handle\_login(self):

card\_id = self.ui\_loginPage.lineEdit.text()

password = self.ui\_loginPage.lineEdit\_2.text()

receive\_message = self.processor.process("log\_in@"+card\_id+"@"+password+"#"+str(self.app\_id))

self.ui\_loginPage.lineEdit.clear()

self.ui\_loginPage.lineEdit\_2.clear()

self.ui\_loginPage.lineEdit.setFocus()

if receive\_message.startswith("fail"):

QtWidgets.QMessageBox.warning(None, "Login Failed", "Invalid ID or Password")

else:

self.stackedWidget.setCurrentWidget(self.selectPage)

* Coverage Criteria: Branch coverage
* Test case

|  |  |  |
| --- | --- | --- |
|  | Test Case T1.3.1.1 | Test Case T1.3.1.2 |
| Coverage Item | Tcover1.3.1.1 | Tcover1.3.1.2 |
| Input |  |  |
| State | UI=APPUI  UI.stackedWidget=Login Page  processor = Processor()  db=user\_database  card\_id=”2023123456”  password=”111111” | UI=APPUI  UI.stackedWidget=Login Page  processor = Processor()  db=user\_database  card\_id=”2023123456”  password=”111112” |
| Expected Output | login succeed  UI.stackedWidget=selectPage | login failed  UI.stackedWidget= Login Page |

* Test coverage: 2/2=100%
* Test result: 1 passed

#### T1.3.2: Test handle\_transfer ( )

def handle\_transfer(self):

receiver\_id = self.ui\_transfer.lineEdit.text()

amount = self.ui\_transfer.lineEdit\_2.text()

receive\_message = self.processor.process("transfer\_money@"+receiver\_id+"@"+amount+"#"+str(self.app\_id))

if receive\_message.startswith("fail"):

QtWidgets.QMessageBox.warning(None, "transfer Failed", "Invalid receiver or lacking balane")

self.ui\_transfer.lineEdit.clear()

self.ui\_transfer.lineEdit\_2.clear()

self.ui\_transfer.lineEdit.setFocus()

self.stackedWidget.setCurrentWidget(self.selectPage)

else:

QtWidgets.QMessageBox.information(None,"transfer succeeded!","SUCCEED")

self.ui\_transfer.lineEdit.clear()

self.ui\_transfer.lineEdit\_2.clear()

self.ui\_transfer.lineEdit.setFocus()

self.stackedWidget.setCurrentWidget(self.selectPage)

* Coverage Criteria: Branch coverage
* Test case

|  |  |  |  |
| --- | --- | --- | --- |
|  | Test Case T1.3.2.1 | Test Case T1.3.2.2 | Test Case T1.3.2.3 |
| Coverage Item | Tcover1.4.1.1 | Tcover1.4.1.2 | Tcover1.4.1.3 |
| Input | receiver\_id ="2023123456"  amount = "1000" | receiver\_id ="0000000000"  amount = "100" | receiver\_id ="2023123456"  amount = "0" |
| State | UI=APPUI  UI.stackedWidget=Transfer Page  processor = Processor()  db=user\_database | UI=APPUI  UI.stackedWidget=Transfer Page  processor = Processor()  db=user\_database | UI=APPUI  UI.stackedWidget=Transfer Page  processor = Processor()  db=user\_database |
| Expected Output | tranfer failed  UI.stackedWidget=selectPage | tranfer failed  UI.stackedWidget=selectPage | tranfer success  UI.stackedWidget=selectPage |

* Test coverage: 3/3=100%
* Test result: 3 passed

#### T1.3.3: Test handle\_query ( )

def handle\_query(self,results):

result = results[0]

if results[1] == None or results[1] != str(self.app\_id):

return

if result:

result\_text = (

f"Account ID: {result['account\_id']}\n"

f"Balance: {result['balance']} YUAN\n"

f"Creation Time: {result['creation\_time']}"

)

else:

result\_text = "Account not found"

self.ui\_query.label\_3.setText(result\_text)

self.stackedWidget.setCurrentWidget(self.queryPage)

Coverage Criteria: Branch coverage

* Test case

|  |  |  |  |
| --- | --- | --- | --- |
|  | Test Case T1.3.3.1 | Test Case T1.3.3.2 | Test Case T1.3.3.3 |
| Coverage Item | Tcover1.3.3.1 | Tcover1.3.3.2 | Tcover1.3.3.3 |
| Input | results = [{'account\_id': '2023123456', 'balance': 1000, 'creation\_time': '2024-01-01 00:00:00'}, '1'] | results = [None,'1'] | results = [{'account\_id': '2023123456', 'balance': None, 'creation\_time': '2024-01-01 00:00:00'}, None] |
| State | UI=APPUI  UI.stackedWidget=SelectPage  processor = Processor()  db=user\_database | UI=APPUI  UI.stackedWidget=SelectPage  processor = Processor()  db=user\_database | UI=APPUI  UI.stackedWidget=SelectPage  processor = Processor()  db=user\_database |
| Expected Output | query success  UI.stackedWidget=queryPage | UI.stackedWidget=SelectPage | query failed  UI.stackedWidget=SelectPage |

* Test coverage: 3/3=100%
* Test result: 3 passed

#### T1.3.4: Test handle\_changePin ( )

def handle\_changePin(self):

new\_pin = self.ui\_changePin.lineEdit.text()

new\_pin2 = self.ui\_changePin.lineEdit\_2.text()

if new\_pin != new\_pin2:

QtWidgets.QMessageBox.information(None,"inconsistent new pin","you should enter consistent new pins")

return

receive\_message = self.processor.process("change\_password@"+new\_pin+"#"+str(self.app\_id))

if receive\_message.startswith("fail"):

QtWidgets.QMessageBox.warning(None, "ChangePin Failed", "Invalid NEW PIN")

self.ui\_changePin.lineEdit.clear()

self.ui\_changePin.lineEdit\_2.clear()

self.ui\_changePin.lineEdit.setFocus()

else:

QtWidgets.QMessageBox.information(None,"ChangePin succeeded!","SUCCEED")

self.ui\_changePin.lineEdit.clear()

self.ui\_changePin.lineEdit\_2.clear()

self.ui\_changePin.lineEdit.setFocus()

self.stackedWidget.setCurrentWidget(self.selectPage)

* Coverage Criteria: Branch coverage
* Test case

|  |  |  |  |
| --- | --- | --- | --- |
|  | Test Case T1.3.4.1 | Test Case T1.3.4.2 | Test Case T1.3.4.3 |
| Coverage Item | Tcover1.3.4.1 | Tcover1.3.4.2 | Tcover1.3.4.3 |
| Input | new\_pin=”123456”  new\_pin2=”123456” | new\_pin=”123456”  new\_pin2=”111111” | new\_pin=”1234”  new\_pin2=”1234” |
| State | UI=APPUI  UI.stackedWidget=changePinPage  processor = Processor()  db=user\_database | UI=APPUI  UI.stackedWidget=changePinPage  processor = Processor()  db=user\_database | UI=APPUI  UI.stackedWidget=changePinPage  processor = Processor()  db=user\_database |
| Expected Output | ChangePin success  UI.stackedWidget=selectPage | UI.stackedWidget=changePinPage | ChangePin Failed  UI.stackedWidget=changePinPage |

* Test coverage: 3/3=100%
* Test result: 3 passed

#### T1.3.5: Test handle\_logout ( )

def handle\_logout(self):

receive\_message = self.processor.process("log\_out#"+str(self.app\_id))

if receive\_message.startswith("fail"):

QtWidgets.QMessageBox.warning(None, "Logout out", "app without login")

else:

self.stackedWidget.setCurrentWidget(self.loginPage)

* Coverage Criteria: Branch coverage
* Test case

|  |  |  |
| --- | --- | --- |
|  | Test Case T1.3.5.1 | Test Case T1.3.5.2 |
| Coverage Item | Tcover1.3.5.1 | Tcover1.3.5.2 |
| Input |  |  |
| State | UI=APPUI  UI.stackedWidget=selectPage  processor = Processor()  db=user\_database | UI=APPUI  UI.stackedWidget=selectPage  processor = Processor()  db=user\_database |
| Expected Output | Logout failed  UI.stackedWidget=selectPage | Logout success  UI.stackedWidget=loginPage |

* Test coverage: 2/2=100%
* Test result: 2 passed

#### T1.3.6: Test back\_to\_select ( )

def back\_to\_select(self):

current\_widget = self.stackedWidget.currentWidget()

for line in current\_widget.findChildren(QtWidgets.QLineEdit):

line.clear()

self.stackedWidget.setCurrentWidget(self.selectPage)

* Coverage Criteria: Statement coverage
* Test case

|  |  |
| --- | --- |
|  | Test Case T1.3.6 |
| Coverage Item | Tcover1.3.6 |
| Input |  |
| State | UI=APPUI  UI.stackedWidget=transferPage  processor = Processor()  db=user\_database |
| Expected Output | success  UI.stackedWidget=selectPage |

* Test coverage: 1/1=100%
* Test result: 1 passed

#### T1.3.7: Test compusary\_log\_out ( )

def compusary\_log\_out(self,aid):

if aid == str(self.app\_id):

self.handle\_logout()

* Coverage Criteria: Branch coverage
* Test case

|  |  |  |
| --- | --- | --- |
|  | Test Case T1.3.7.1 | Test Case T1.3.7.2 |
| Coverage Item | Tcover1.3.7.1 | Tcover1.3.7.2 |
| Input | aid = “1” | aid = “2” |
| State | UI=APPUI  processor = Processor()  db=user\_database  app\_id = 1 | UI=APPUI  processor = Processor()  db=user\_database  app\_id = 1 |
| Expected Output | Logout success  UI.stackedWidget=loginPage | Logout failed |

* Test coverage: 2/2=100%
* Test result: 2 passed

#### T1.3.8: Test set\_app\_id ( )

def set\_app\_id(self, app\_id):

self.app\_id = app\_id

* Coverage Criteria: Statement coverage
* Test case

|  |  |
| --- | --- |
|  | Test Case T1.3.8 |
| Coverage Item | Tcover1.3.8 |
| Input | aid=2 |
| State | UI=APPUI  processor = Processor()  db=user\_database  app\_id = 1 |
| Expected Output | app\_id = 2 |

* Test coverage: 1/1=100%
* Test result: 1 passed

#### T1.3.9: Test handle\_close\_app ( )

def handle\_close\_app(self):

self.processor.process("close\_app#" + str(self.app\_id))

* Coverage Criteria: Statement coverage
* Test case

|  |  |
| --- | --- |
|  | Test Case T1.3.9 |
| Coverage Item | Tcover1.3.9 |
| Input |  |
| State | UI=APPUI  processor = Processor()  db=user\_database  app\_id = 1 |
| Expected Output | close\_app#1 |

* Test coverage: 1/1=100%
* Test result: 1 passed

#### T1.3.10: Test closeEvent ( )

def closeEvent(self, event):

self.processor.process("close\_app#" + str(self.app\_id))

event.accept()

* Coverage Criteria: Statement coverage
* Test case

|  |  |
| --- | --- |
|  | Test Case T1.3.10 |
| Coverage Item | Tcover1.3.10 |
| Input |  |
| State | UI=APPUI  processor = Processor()  db=user\_database  app\_id = 1 |
| Expected Output | close\_app#1 |

* Test coverage: 1/1=100%
* Test result: 1 passed

### T1.4: ATMUI Unit Test

#### T1.4.1: Test checkPIN ( )

def checkPIN(self,card\_id = None):

'''Call out the checkPIN page to ensure security.'''

if card\_id == None:

card\_id = self.account\_id

self.check\_page = CheckPINPage(self.processor,card\_id)

result = self.check\_page.exec\_()

return result == QDialog.Accepted

* Coverage Criteria: Statement coverage
* Test case

|  |  |
| --- | --- |
|  | Test Case T1.4.1.1 |
| Coverage Item | Tcover1.4.1.1 |
| Input | card\_id=2023123456 |
| State | UI=ATMUI  processor = Processor()  db=user\_database |
| Expected Output | Dialog accepted |

* Test coverage: 1/1=100%
* Test result: 1 passed

#### T1.4.2: Test handle\_insert\_card ( )

def handle\_insert\_card(self):

dialog = CardInputDialog()

if dialog.exec\_() == QtWidgets.QDialog.Accepted:

card\_id = dialog.get\_card\_id()

if not self.checkPIN(card\_id):

return

self.insert\_card(card\_id)

* Coverage Criteria: Branch coverage
* Test case

|  |  |  |
| --- | --- | --- |
|  | Test Case T1.4.2.1 | Test Case T1.4.2.2 |
| Coverage Item | Tcover1.4.2.1 | Tcover1.4.2.2 |
| Input |  |  |
| State | UI=ATMUI  processor = Processor()  UI.stackedWidget=LogInPage  db=user\_database  db.accounts = {(2023123456,111111,50  0,AccountState.Valid, creation\_time)} | UI=ATMUI  processor = Processor()  UI.stackedWidget=LogInPage  db=user\_database  db.accounts = {(2023123456,111111,50  0,AccountState.Valid, creation\_time)} |
| Expected Output |  | Insert\_card(card\_id) |

* Test coverage: 2/2=100%
* Test result: 2 passed

#### T1.4.3: Test insert\_card ( )

def insert\_card(self,card\_id):

receive\_message = self.processor.process("insert\_card@"+card\_id)

# self.display\_widget.update\_display(f"Card Number: {card\_number}")

if receive\_message.startswith("fail"):

QtWidgets.QMessageBox.warning(None, "Card Insert Failed", "Invalid ID")

else:

# self.display\_widget.enable\_eject\_button()

self.account\_id = card\_id

self.toolButton.setEnabled(False)

self.display\_widget.stacked\_layout.setCurrentWidget(self.selectPage)

* Coverage Criteria: Branch coverage
* Test case

|  |  |  |
| --- | --- | --- |
|  | Test Case T1.4.3.1 | Test Case T1.4.3.2 |
| Coverage Item | Tcover1.4.3.1 | Tcover1.4.3.2 |
| Input | card\_id=”0000000000” | card\_id=”2023123456” |
| State | UI=ATMUI  processor = Processor()  UI.stackedWidget=LogInPage  db=user\_database  db.accounts = {(2023123456,111111,50  0,AccountState.Valid, creation\_time)} | UI=ATMUI  processor = Processor()  UI.stackedWidget=LogInPage  db=user\_database  db.accounts = {(2023123456,111111,50  0,AccountState.Valid, creation\_time)} |
| Expected Output | Insert failed | Insert success  UI.stackedWidget=selectPage |

* Test coverage: 2/2=100%
* Test result: 2 passed

#### T1.4.4: Test handle\_return\_card ( )

def handle\_return\_card(self):

receive\_message = self.processor.process("return\_card")

if receive\_message.startswith("fail"):

QtWidgets.QMessageBox.warning(None, "Return Failed", "Error")

else:

self.account\_id = None

self.toolButton.setEnabled(True)

self.display\_widget.stacked\_layout.setCurrentWidget

(self.display\_widget.EntryPage)

* Coverage Criteria: Branch coverage
* Test case

|  |  |  |
| --- | --- | --- |
|  | Test Case T1.4.4.1 | Test Case T1.4.4.2 |
| Coverage Item | Tcover1.4.4.1 | Tcover1.4.4.2 |
| Input |  |  |
| State | UI=ATMUI  processor = Processor()  UI.stackedWidget=selectPage  db=user\_database | UI=ATMUI  processor = Processor()  UI.stackedWidget=selectPage  db=user\_database |
| Expected Output | return failed | return success  UI.stackedWidget= display\_widget.EntryPage |

* Test coverage: 2/2=100%
* Test result: 2 passed

#### T1.4.5: Test handle\_change\_pin ( )

def handle\_change\_pin(self):

new\_pin = self.ui\_change\_pin.lineEdit.text()

new\_pin2 = self.ui\_change\_pin.lineEdit\_2.text()

if new\_pin != new\_pin2:

QtWidgets.QMessageBox.information(None,"inconsistent new pin","you should enter consistent new pins")

return

receive\_message = self.processor.process("change\_password@"+new\_pin)

if receive\_message.startswith("fail"):

QtWidgets.QMessageBox.warning(None, "ChangePin Failed", "Invalid NEW PIN")

self.ui\_change\_pin.lineEdit.clear()

self.ui\_change\_pin.lineEdit\_2.clear()

self.ui\_change\_pin.lineEdit.setFocus()

else:

QtWidgets.QMessageBox.information(None,"ChangePin succeeded!","SUCCEED")

self.ui\_change\_pin.lineEdit.clear()

self.ui\_change\_pin.lineEdit\_2.clear()

self.ui\_change\_pin.lineEdit.setFocus()

self.display\_widget.stacked\_layout.setCurrentWidget(self.selectPage)

* Coverage Criteria: Branch coverage
* Test case

|  |  |  |  |
| --- | --- | --- | --- |
|  | Test Case T1.4.5.1 | Test Case T1.4.5.2 | Test Case T1.4.5.3 |
| Coverage Item | Tcover1.4.5.1 | Tcover1.4.5.2 | Tcover1.4.5.3 |
| Input | new\_pin = “123456”  new\_pin2 = “111111” | new\_pin = “1234”  new\_pin2 = “1234” | new\_pin = “123456”  new\_pin2 = “123456” |
| State | UI=ATMUI  processor = Processor()  UI.stackedWidget=changePinPage  db=user\_database | UI=ATMUI  processor = Processor()  UI.stackedWidget=changePinPage  db=user\_database | UI=ATMUI  processor = Processor()  UI.stackedWidget=changePinPage  db=user\_database |
| Expected Output | Changepin failed  UI.stackedWidget=changePinPage | Changepin failed  UI.stackedWidget=changePinPage | Changepin success  UI.stackedWidget=selectPage |

* Test coverage: 3/3=100%
* Test result: 3 passed

#### T1.4.6: Test handle\_query ( )

def handle\_query(self,results):

result = results[0]

if results[1] != None:

return

if result:

result\_text = (

f"Account ID: {result['account\_id']}\n"

f"Balance: {result['balance']}\n"

f"Creation Time: {result['creation\_time']}"

)

else:

result\_text = "Account not found"

self.ui\_query.label\_3.setText(result\_text)

self.display\_widget.stacked\_layout.setCurrentWidget(self.queryPage)

* Coverage Criteria: Branch coverage
* Test case

|  |  |  |  |
| --- | --- | --- | --- |
|  | Test Case T1.4.6.1 | Test Case T1.4.6.2 | Test Case T1.4.6.3 |
| Coverage Item | Tcover1.4.6.1 | Tcover1.4.6.2 | Tcover1.4.6.3 |
| Input | results = [{'account\_id': '2023123456', 'balance': 1000, 'creation\_time': '2024-01-01 00:00:00'}, 1] | results = [None, None] | results = [{'account\_id': '2023123456', 'balance': 1000, 'creation\_time': '2024-01-01 00:00:00'}, None] |
| State | UI=ATMUI  processor = Processor()  UI.stackedWidget=selectPage  db=user\_database | UI=ATMUI  processor = Processor()  UI.stackedWidget=selectPage  db=user\_database | UI=ATMUI  processor = Processor()  UI.stackedWidget=selectPage  db=user\_database |
| Expected Output | UI.stackedWidget=selectPage | Query failed  UI.stackedWidget=selectPage | Query success  UI.stackedWidget=queryPage |

* Test coverage: 3/3=100%
* Test result: 3 passed

#### T1.4.7: Test handle\_transfer ( )

def handle\_transfer(self):

receiver\_id = self.ui\_transfer.lineEdit.text()

amount = self.ui\_transfer.lineEdit\_2.text()

if(not amount.isdigit() or (amount.isdigit() and int(amount) == 0)):

QtWidgets.QMessageBox.warning(None, "transfer Failed", "Invalid amount")

self.ui\_transfer.lineEdit.clear()

self.ui\_transfer.lineEdit\_2.clear()

self.ui\_transfer.lineEdit.setFocus()

self.display\_widget.stacked\_layout.setCurrentWidget(self.selectPage)

return

receive\_message = self.processor.process("transfer\_money@"+receiver\_id+"@"+amount)

if receive\_message.startswith("fail"):

QtWidgets.QMessageBox.warning(None, "transfer Failed", "Invalid receiver or lacking balane")

self.ui\_transfer.lineEdit.clear()

self.ui\_transfer.lineEdit\_2.clear()

self.ui\_transfer.lineEdit.setFocus()

self.display\_widget.stacked\_layout.setCurrentWidget(self.selectPage)

else:

QtWidgets.QMessageBox.information(None,"transfer succeeded!","SUCCEED")

self.ui\_transfer.lineEdit.clear()

self.ui\_transfer.lineEdit\_2.clear()

self.ui\_transfer.lineEdit.setFocus()

self.display\_widget.stacked\_layout.setCurrentWidget(self.selectPage)

* Coverage Criteria: Branch coverage
* Test case

|  |  |  |  |
| --- | --- | --- | --- |
|  | Test Case T1.4.7.1 | Test Case T1.4.7.2 | Test Case T1.4.7.3 |
| Coverage Item | Tcover1.4.7.1 | Tcover1.4.7.2 | Tcover1.4.7.3 |
| Input | receiver\_id=self.new\_id  amount=”abc” | receiver\_id=”2023123456”  amount=”100” | receiver\_id=self.new\_id  amount=”100” |
| State | UI=ATMUI  processor = Processor()  UI.stackedWidget=transferPage  db=user\_database  db.accounts = {(2023123456,111111,50  0,AccountState.Valid, creation\_time)(self.new\_id,222222,0,AccountState.Valid,creation\_time)} | UI=ATMUI  processor = Processor()  UI.stackedWidget=transferPage  db=user\_database  db.accounts = {(2023123456,111111,50  0,AccountState.Valid, creation\_time)(self.new\_id,222222,0,AccountState.Valid,creation\_time)} | UI=ATMUI  processor = Processor()  UI.stackedWidget=transferPage  db=user\_database  db.accounts = {(2023123456,111111,50  0,AccountState.Valid, creation\_time)(self.new\_id,222222,0,AccountState.Valid,creation\_time)} |
| Expected Output | Transfer failed  UI.stackedWidget=selectPage | Transfer failed  UI.stackedWidget=selectPage | Transfer success  UI.stackedWidget=selectPage |

* Test coverage: 3/3=100%
* Test result: 3 passed

#### T1.4.8: Test handle\_deposit\_cash ( )

def handle\_deposit\_cash(self):

amount = str(self.my\_ATM.deposit\_amount)

if int(amount) % 100 != 0 or int(amount) == 0:

QtWidgets.QMessageBox.warning(self, "deposit Failed", "Invalid amount")

self.display\_widget.stacked\_layout.setCurrentWidget(self.selectPage)

return

self.ui\_deposit\_cash.label\_2.setText(f"Deposit Amount: {amount}")

receive\_message = self.processor.process("deposit\_cash@"+amount)

if receive\_message.startswith("fail"):

QtWidgets.QMessageBox.warning(None, "deposit Failed", "Invalid amount")

else:

QtWidgets.QMessageBox.information(None,"deposit succeeded!","SUCCEED")

self.ATM\_cashupdate(int(amount))

self.my\_ATM.deposit\_amount = 0

self.display\_widget.stacked\_layout.setCurrentWidget(self.selectPage)

self.ui\_deposit\_cash.label\_2.setText(f"Deposit Amount: {self.my\_ATM.deposit\_amount}")

* Coverage Criteria: Branch coverage
* Test case

|  |  |  |  |
| --- | --- | --- | --- |
|  | Test Case T1.4.8.1 | Test Case T1.4.8.2 | Test Case T1.4.8.3 |
| Coverage Item | Tcover1.4.8.1 | Tcover1.4.8.2 | Tcover1.4.8.3 |
| Input | deposit\_amount = 10 | deposit\_amount = 10000 | deposit\_amount = 10000 |
| State | UI=ATMUI  processor = Processor()  UI.stackedWidget=selectPage  db=user\_database | UI=ATMUI  processor = Processor()  UI.stackedWidget=selectPage  db=user\_database | UI=ATMUI  processor = Processor()  UI.stackedWidget=selectPage  db=user\_database |
| Expected Output | Deposit falied | Deposit falied | Deposit success |

* Test coverage: 3/3=100%
* Test result: 3 passed

#### T1.4.9: Test handle\_withdraw\_cash ( )

def handle\_withdraw\_cash(self):

amount = self.ui\_withdraw\_cash.lineEdit.text()

if not amount.isdigit():

self.ui\_withdraw\_cash.lineEdit.clear()

self.ui\_withdraw\_cash.lineEdit.setFocus()

QtWidgets.QMessageBox.warning(None, "Invalid amount", "You should enter correct amount!")

self.display\_widget.stacked\_layout.setCurrentWidget(self.selectPage)

return

if amount == "" or int(amount) == 0:

self.ui\_withdraw\_cash.lineEdit.clear()

self.ui\_withdraw\_cash.lineEdit.setFocus()

QtWidgets.QMessageBox.warning(None, "Invalid amount", "You should enter correct amount!")

self.display\_widget.stacked\_layout.setCurrentWidget(self.selectPage)

return

if int(amount) > self.my\_ATM.cash:

QtWidgets.QMessageBox.warning(None, "ATM Out of Money", "Sorry! ATM is out of money.")

self.ui\_withdraw\_cash.lineEdit.clear()

self.ui\_withdraw\_cash.lineEdit.setFocus()

self.display\_widget.stacked\_layout.setCurrentWidget(self.selectPage)

return

password, ok = QtWidgets.QInputDialog.getText(None, "Enter Password", "Please enter your password:", QtWidgets.QLineEdit.Password)

if not ok:

return

self.withdraw\_cash(amount,password)

* Coverage Criteria: Branch coverage
* Test case

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Test Case T1.4.9.1 | Test Case T1.4.9.2 | Test Case T1.4.9.3 | Test Case T1.4.9.4 |
| Coverage Item | Tcover1.4.9.1 | Tcover1.4.9.2 | Tcover1.4.9.3 | Tcover1.4.9.4 |
| Input | Amount=”abc” | Amount=” ” | Amount=”10000” | Amount=”100” |
| State | UI=ATMUI  processor = Processor()  UI.stackedWidget=WithdrawCashPage  db=user\_database | UI=ATMUI  processor = Processor()  UI.stackedWidget=WithdrawCashPage  db=user\_database | UI=ATMUI  processor = Processor()  UI.stackedWidget=WithdrawCashPage  db=user\_database  my\_ATM.cash=500 | UI=ATMUI  processor = Processor()  UI.stackedWidget=WithdrawCashPage  db=user\_database  my\_ATM.cash=500 |
| Expected Output | Withdraw failed | Withdraw failed | Withdraw failed | "Enter Password", "Please enter your password:" |

* Test coverage: 4/4=100%
* Test result: 4 passed

#### T1.4.10: Test withdraw\_cash ( )

def withdraw\_cash(self,amount,password):

receive\_message = self.processor.process("withdraw\_cash@"+amount+"@"+password)

if receive\_message.startswith("fail"):

QtWidgets.QMessageBox.warning(None, "Withdraw Failed", "Invalid password or not enough balance")

self.ui\_withdraw\_cash.lineEdit.clear()

self.ui\_withdraw\_cash.lineEdit.setFocus()

else:

QtWidgets.QMessageBox.information(None,"Withdraw succeeded!","SUCCEED")

self.ATM\_cashupdate(-int(amount))

self.ui\_withdraw\_cash.lineEdit.clear()

self.ui\_withdraw\_cash.lineEdit.setFocus()

self.display\_widget.stacked\_layout.setCurrentWidget(self.selectPage)

* Coverage Criteria: Branch coverage
* Test case

|  |  |  |
| --- | --- | --- |
|  | Test Case T1.4.10.1 | Test Case T1.4.10.2 |
| Coverage Item | Tcover1.4.10.1 | Tcover1.4.10.2 |
| Input | amount=”100”  password=”000000” | amount=”100”  password=”111111” |
| State | UI=ATMUI  processor = Processor()  UI.stackedWidget=WithdrawCashPage  db=user\_database  db.accounts = {(2023123456,111111,50  0,AccountState.Valid, creation\_time)}  ui.my\_ATM.cash = 1000 | UI=ATMUI  processor = Processor()  UI.stackedWidget=WithdrawCashPage  db=user\_database  db.accounts = {(2023123456,111111,50  0,AccountState.Valid, creation\_time)}  ui.my\_ATM.cash = 1000 |
| Expected Output | Withdraw failed  ui.my\_ATM.cash = 1000 | Withdraw success  ui.my\_ATM.cash = 900 |

* Test coverage: 2/2=100%

Test result: 2 passed

#### T1.4.11: Test handle\_create\_account ( )

def handle\_create\_account(self):

password, ok = QtWidgets.QInputDialog.getText(None, "Set Password", "Please set your password:", QtWidgets.QLineEdit.Password)

if not ok:

return

else:

self.create\_account(password)

* Coverage Criteria: Branch coverage
* Test case

|  |  |  |
| --- | --- | --- |
|  | Test Case T1.4.11.1 | Test Case T1.4.11.2 |
| Coverage Item | Tcover1.4.11.1 | Tcover1.4.11.2 |
| Input | Password=”123456” | Password=”1234” |
| State | UI=ATMUI  processor = Processor()  UI.stackedWidget=selectPage  db=user\_database | UI=ATMUI  processor = Processor()  UI.stackedWidget=selectPage  db=user\_database |
| Expected Output | create\_account(password) | handle\_create\_account failed |

* Test coverage: 2/2=100%

#### T1.4.12: Test create\_account ( )

def create\_account(self,password):

receive\_message = self.processor.process("create\_account@"+password)

if receive\_message.startswith("fail"):

QtWidgets.QMessageBox.warning(None, "Invalid password", "Your password should be 6 digits")

return

card\_id = receive\_message.split("@")[1]

QtWidgets.QMessageBox.warning(None, "Accound Created", "Your account id is "+card\_id)

self.toolButton.setEnabled(False)

self.display\_widget.stacked\_layout.setCurrentWidget(self.selectPage)

* Coverage Criteria: Branch coverage
* Test case

|  |  |  |
| --- | --- | --- |
|  | Test Case T1.4.12.1 | Test Case T1.4.12.2 |
| Coverage Item | Tcover1.4.12.1 | Tcover1.4.12.2 |
| Input | Password=”123456” | Password=”1234” |
| State | UI=ATMUI  processor = Processor()  UI.stackedWidget=selectPage  db=user\_database | UI=ATMUI  processor = Processor()  UI.stackedWidget=selectPage  db=user\_database |
| Expected Output | Create account success  UI.stackedWidget=selectPage | Create account failed |

* Test coverage: 2/2=100%

Test result: 2 passed

#### T1.4.13: Test go\_to\_selectPage ( )

def go\_to\_selectPage(self):

current\_widget = self.display\_widget.stacked\_layout.currentWidget()

for line in current\_widget.findChildren(QtWidgets.QLineEdit):

if(not line.isReadOnly()):

line.clear()

self.display\_widget.stacked\_layout.setCurrentWidget(self.selectPage)

* Coverage Criteria: Statement coverage
* Test case

|  |  |
| --- | --- |
|  | Test Case T1.4.13.1 |
| Coverage Item | Tcover1.4.13.1 |
| Input |  |
| State | UI=ATMUI  processor = Processor()  db=user\_database  UI.stackedWidget=transferPage |
| Expected Output | UI.stackedWidget=selectPage |

* Test coverage: 1/1=100%

Test result: 1 passed

#### T1.4.14: Test handle\_close\_account ( )

def handle\_close\_account(self):

if self.confirm\_close\_account(3):

ms:str = self.processor.process("close\_account")

if ms.startswith("failed"):

QtWidgets.QMessageBox.warning(self.display\_widget,"Close Account failed","Account still has money.")

else:

self.display\_widget.stacked\_layout.setCurrentWidget(self.display\_widget.EntryPage)

QtWidgets.QMessageBox.information(self.display\_widget,"Close Account","Succeed!")

self.toolButton.setEnabled(True)

else:

QtWidgets.QMessageBox.information(self.display\_widget,"Close Account","Account closure cancelled.")

* Coverage Criteria: Branch coverage
* Test case

|  |  |  |
| --- | --- | --- |
|  | Test Case T1.4.14.1 | Test Case T1.4.14.2 |
| Coverage Item | Tcover1.4.14.1 | Tcover1.4.14.2 |
| Input |  |  |
| State | UI=ATMUI  processor = Processor()  UI.stackedWidget=selectPage  db=user\_database  db.accounts = {(2023123456,111111,50  0,AccountState.Valid, creation\_time)} | UI=ATMUI  processor = Processor()  UI.stackedWidget=selectPage  db=user\_database  db.accounts = {(self.new\_id,222222,  0,AccountState.Valid, creation\_time)} |
| Expected Output | "Close Account failed" | "Close Account Succeed!" |

* Test coverage: 2/2=100%

Test result: 2 passed

#### T1.4.15: Test collectCash ( )

def collectCash(self):

dialog = CashInputDialog()

dialog.check\_cash(str(self.my\_ATM.deposit\_amount))

if dialog.exec\_() == QtWidgets.QDialog.Accepted:

if(not dialog.get\_cash().isdigit()):

# QtWidgets.QMessageBox.warning(self.display\_widget,"Close Account failed","Account still has money.")

return

self.my\_ATM.deposit\_amount = int(dialog.get\_cash())

self.ui\_deposit\_cash.label\_2.setText(f"Deposit Amount: {self.my\_ATM.deposit\_amount}")

* Coverage Criteria: Branch coverage
* Test case

|  |  |  |
| --- | --- | --- |
|  | Test Case T1.4.15.1 | Test Case T1.4.15.2 |
| Coverage Item | Tcover1.4.15.1 | Tcover1.4.15.2 |
| Input | Cash =” abc” | Cash=”100” |
| State | Dialog = CashInputDialog  UI=ATMUI  processor = Processor()  db=user\_database  QtWidgets.QDialog.Accepted==True | Dialog = CashInputDialog  UI=ATMUI  processor = Processor()  db=user\_database  QtWidgets.QDialog.Accepted==True |
| Expected Output |  | deposit\_amount=100 |

* Test coverage: 2/2=100%

Test result: 2 passed

#### T1.4.16: Test ATM\_cashupdate ( )

def ATM\_cashupdate(self,amount):

self.my\_ATM.cash+=amount

* Coverage Criteria: Statement coverage
* Test case

|  |  |
| --- | --- |
|  | Test Case T1.4.16.1 |
| Coverage Item | Tcover1.4.16.1 |
| Input | Amount=100 |
| State | UI=ATMUI  processor = Processor()  db=user\_database  my\_ATM.cash = My\_ATM.cash |
| Expected Output | my\_ATM.cash = My\_ATM.cash+100 |

* Test coverage: 1/1=100%

Test result: 1 passed

### T1.5: ATM Unit Test

#### T1.5.1: Test deposit ( )

def deposit(self, amount):

"""deposit"""

if amount > 0:

self.cash += amount

return True

else:

return False

* Coverage Criteria: Branch coverage
* Test case

|  |  |  |
| --- | --- | --- |
|  | Test Case T1.5.1.1 | Test Case T1.5.1.2 |
| Coverage Item | Tcover1.5.1.1 | Tcover1.5.1.2 |
| Input | amount=500 | amount = -100 |
| State | atm = ATM(1000)  self.cash = 1000 | atm = ATM(1000)  self.cash = 1000 |
| Expected Output | cash = 1500  True | cash = 1000  False |

* Test coverage: 2/2=100%
* Test result: 2 passed

#### T1.5.2: Test withdraw ( )

def withdraw(self, amount):

"""withdraw"""

if amount > 0 and self.cash >= amount:

self.cash -= amount

return True

else:

return False

* Coverage Criteria: Branch coverage
* Test case

|  |  |  |
| --- | --- | --- |
|  | Test Case T1.5.2.1 | Test Case T1.5.2.2 |
| Coverage Item | Tcover1.5.2.1 | Tcover1.5.2.2 |
| Input | amount=200 | amount = 1200 |
| State | atm = ATM(1000)  self.cash = 1000 | atm = ATM(1000)  self.cash = 1000 |
| Expected Output | cash = 800  True | cash = 1000  False |

* Test coverage: 2/2=100%
* Test result: 2 passed

#### T1.5.3: Test check\_balance ( )

def check\_balance(self):

"""check\_balance"""

return self.cash

* Coverage Criteria: Statement coverage
* Test case

|  |  |
| --- | --- |
|  | Test Case T1.5.3 |
| Coverage Item | Tcover1.5.3 |
| Input |  |
| State | atm = ATM(1000)  self.cash = 1000 |
| Expected Output | return 1000 |

* Test coverage: 1/1=100%
* Test result: 1 passed

## T2: Integration Test

This section provides information of integration tests we made for the Banking System. Testing cases with runnable test functions are provided in every test, you can find in the corresponding files.

### T2.1: Processor + Database + APPUI Integration

#### T2.1.1: Test Change Pin

* Test case

|  |  |
| --- | --- |
|  | Test Case T2.1.1 |
| Coverage Item | Tcover1.1.4, Tcover1.1.5, Tcover1.1.7, Tcover1.1.8, Tcover1.1.10, Tcover1.1.14, Tcover1.2.3, Tcover1.2.5, Tcover1.2.7, Tcover1.3.1, Tcover1.3.4, Tcover1.3.5, Tcover1.3.9 |
| Input | "open\_app"  "log\_in@2023123456@111111#1"  "change\_password@123456#1"  "log\_out#1"  "log\_in@2023123456@111111#1"  "log\_in@2023123456@123456#1"  "close\_app#1" |
| State | The database has only one account information with {id: 2023123456, password: 111111, money:500} |
| Expected Output | User successfully loged in to the account and modify its password;  after his logging out, his second attempts to log in to the same account with original password failed and with the new one success.  The app was successfully closed. |

* Test coverage: 13 / 13 = 100%
* Test result: 1 passed

#### T2.1.2: Test Transfer

* Test case

|  |  |
| --- | --- |
|  | Test Case T2.1.2 |
| Coverage Item | Tcover1.1.1, Tcover1.1.3, Tcover1.1.7, Tcover1.1.9, Tcover1.1.11, Tcover1.2.1, Tcover1.2.3, Tcover1.2.4, Tcover1.2.5, Tcover1.2.8, Tcover1.2.10, Tcover1.3.1, Tcover1.3.2, Tcover1.3.3, Tcover1.4.11 |
| Input | "create\_account@222222",  "deposit\_cash@1000",  "open\_app"  "open\_app"  "log\_in@id@222222#1",  "transfer\_money@2023123456@400#1",  "log\_in@20231223456@111111#2",  "query#1",  "query#2" |
| State | The database has only one account information with {id: 2023123456, password: 111111, money:500} |
| Expected Output | A new account was created and 1000 yuan was deposited  Use correct password successfully loged into the new account and successfully transfer 400 yuan  successfully log into app2 with account id 2023123456  both query showed |

* Test coverage: 15 / 15 = 100%
* Test result: 1 passed

### T2.2: Processor + Database + ATMUI + ATM Integration

#### T2.2.1: Test Account Operation

* Test case

|  |  |
| --- | --- |
|  | Test Case T2.2.1 |
| Coverage Item | Tcover1.1.1, Tcover1.1.2, Tcover1.1.10, Tcover1.2.1, Tcover1.2.2, Tcover1.2.7, Tcover1.4.5, Tcover1.4.11, Tcover1.4.12, Tcover1.4.14 |
| Input | "create\_account@222222",  "change\_password@123456",  "close\_account"  "open\_app"  "log\_in@id@222222#1",  "log\_in@id@123456#1" |
| State | The database has only one account information with {id: 2023123456, password: 111111, money:500} |
| Expected Output | The account was successfully created and password has been changed and finally the account was closed successfully  So all attempts to log in the closed account failed |

* Test coverage: 10 / 10 = 100%
* Test result: 1 passed

#### T2.2.2: Test Money Operation

* Test case

|  |  |
| --- | --- |
|  | Test Case T2.2.2 |
| Coverage Item | Tcover1.1.1, Tcover1.1.3, Tcover1.1.4, Tcover1.1.9, Tcover1.1.11, Tcover1.1.13, Tcover1.1.14, Tcover1.2.3, Tcover1.2.4, Tcover1.2.5, Tcover1.2.9, Tcover1.2.10, Tcover1.4.1, Tcover1.4.2, Tcover1.4.3, Tcover1.4.4, Tcover1.4.6, Tcover1.4.7, Tcover1.4.8, Tcover1.4.11, Tcover1.4.12, Tcover1.4.15, Tcover1.4.16 |
| Input | "insert\_card@2023123456",  "deposit\_cash@500",  "withdraw\_cash@200@111111",  "create\_account@222222",  "transfer\_money@id@100",  "query"  "return\_card" |
| State | The database has only one account information with {id: 2023123456, password: 111111, money:500} |
| Expected Output | All operations succeed |

* Test coverage: 23 / 23 = 100%
* Test result: 1 passed

### T2.3: Processor + Database + APPUI + ATMUI + ATM Integration

#### T2.3.1: Test Money Operation

* Test case

|  |  |
| --- | --- |
|  | Test Case T2.3.1 |
| Coverage Item | Tcover1.1.1, Tcover1.1.2, Tcover1.1.3, Tcover1.1.4, Tcover1.1.5, Tcover1.1.6, Tcover1.1.7, Tcover1.1.8, Tcover1.1.10, Tcover1.1.11, Tcover1.1.14, Tcover1.2.1, Tcover1.2.2, Tcover1.2.3, Tcover1.2.4, Tcover1.2.5, Tcover1.2.7, Tcover1.2.8, Tcover1.2.9, Tcover1.2.10, Tcover1.3.1, Tcover1.3.2, Tcover1.3.3, Tcover1.3.4, Tcover1.3.5, Tcover1.3.6, Tcover1.4.9, Tcover1.3.10, Tcover1.4.1, Tcover1.4.5, Tcover1.4.6, Tcover1.4.7, Tcover1.4.8, Tcover1.4.9, Tcover1.4.11, Tcover1.4.12, Tcover1.4.13, Tcover1.4.14, Tcover1.4.15, Tcover1.4.16 |
| Input | "create\_account@222222",  “change\_password@123456”  "deposit\_cash@700",  "open\_app",  "log\_in@id@123456#1",  "transfer\_money@2023123456@400#1",  "withdraw\_cash@200@123456",  “query#1”,  “log\_out#1”,  "close\_app#1",  "transfer\_money@2023123456@100”  “query”  "close\_account" |
| State | The database has only one account information with {id: 2023123456, password: 111111, money:500} |
| Expected Output | All operations success |

* Test coverage: 33 / 33 = 100%
* Test result: 1 passed

## T3: Functional Test

### T3.1: Use Case “Use ATM Machine”

#### T3.1.1: Test “ATM Deposit Cash”

* Test case

|  |  |
| --- | --- |
|  | Test Case T3.1.1 |
| State | The database has only one account information with {id: 2023123456, password: 111111, money:500} |
| Operation | "insert\_card@2023123456",  "deposit\_cash@500",  "query" |
| Expected Behavior | Once the query showed, the updated information is correct |

* Test result: 1 passed

#### T3.1.2: Test “ATM Withdraw Cash”

* Test case

|  |  |
| --- | --- |
|  | Test Case T3.1.2 |
| State | The database has only one account information with {id: 2023123456, password: 111111, money:500}  ATM\_cash = 800000 |
| Operation | "insert\_card@2023123456",  "deposit\_cash@1000",  "withdraw\_cash@300@111111",  "query" |
| Expected Behavior | All operations success  Once the query showed, the updated information is correct |

* Test result: 1 passed

#### T3.1.3: Test “ATM Transfer”

* Test case

|  |  |
| --- | --- |
|  | Test Case T3.1.3 |
| State | The database has only one account information with {id: 2023123456, password: 111111, money:500} |
| Operation | "insert\_card@2023123456",  "deposit\_cash@1000",  "create\_account@222222",  "transfer\_money@id@400",  "query",  “open\_app”,  "log\_in@id@222222#1",  "query#1",  "close\_app" |
| Expected Behavior | All operations success  The first query shows that account: 2023123456 has balance : 600  The second query shows that the newly created account has balance : 400 |

* Test result: 1 passed

#### T3.1.4: Test “ATM Query Account”

* Test case

|  |  |
| --- | --- |
|  | Test Case T3.1.4 |
| State | The database has only one account information with {id: 2023123456, password: 111111, money:500} |
| Operation | "insert\_card@2023123456",  "deposit\_cash@500",  "withdraw\_cash@200@111111",  "create\_account@222222",  "transfer\_money@id@100",  "query" |
| Expected Behavior | All operations success  The query shows that account: 2023123456 has balance : 200 |

* Test result: 1 passed

### T3.2: Use Case “Use Banking APP”

#### T3.2.1: Test “APP Reset Information”

* Test case

|  |  |
| --- | --- |
|  | Test Case T3.2.1 |
| State | The database has only one account information with {id: 2023123456, password: 111111, money:500} |
| Operation | "open\_app",  "log\_in@2023123456@111111#1"  “change\_password@123456#1”  "log\_out#1"  "log\_in@2023123456@111111#1"  "log\_in@2023123456@123456#1"  "close\_app#1" |
| Expected Behavior | User successfully loged in to the account and modify its password;  after his logging out, his second attempts to log in to the same account with original password failed and with the new one success.  The app was successfully closed. |

* Test result: 1 passed

#### T3.2.2: Test “APP Transfer”

* Test case

|  |  |
| --- | --- |
|  | Test Case T3.2.2 |
| State | The database has only one account information with {id: 2023123456, password: 111111, money:500} |
| Operation | "create\_account@222222",  "deposit\_cash@1000",  "open\_app",  "open\_app"  "log\_in@id@222222#1",  "transfer\_money@2023123456@400#1",  "log\_in@20231223456@111111#2",  "query#1",  "query#2",  "close\_app#1" |
| Expected Behavior | All operations success |

* Test result: 1 passed

#### T3.2.3: Test “APP Query Account”

* Test case

|  |  |
| --- | --- |
|  | Test Case T3.2.3 |
| State | The database has only one account information with {id: 2023123456, password: 111111, money:500} |
| Operation | "create\_account@222222"  "deposit\_cash@500"  "open\_app",  "log\_in@id@222222#1",  "transfer\_money@2023123456@100#1",  "query#1" ,  "close\_app#1" |
| Expected Behavior | All operations success |

* Test result: 1 passed

## T4: Model Checking

This section provides an abstract model built in UPPAAL for model checking purposes. You can find the source files in uppaal and run it locally using an UPPAAL application.

### T4.1: Introduction

The Banking System is divided into three components: the processor, the ATM, and the APP. Since the user operation is consistent with the page changing of the ATM UI and APP UI and the invalid operation has been managed and verfied in the former test, our model focuses more on the integration level of the single UI and the processor.

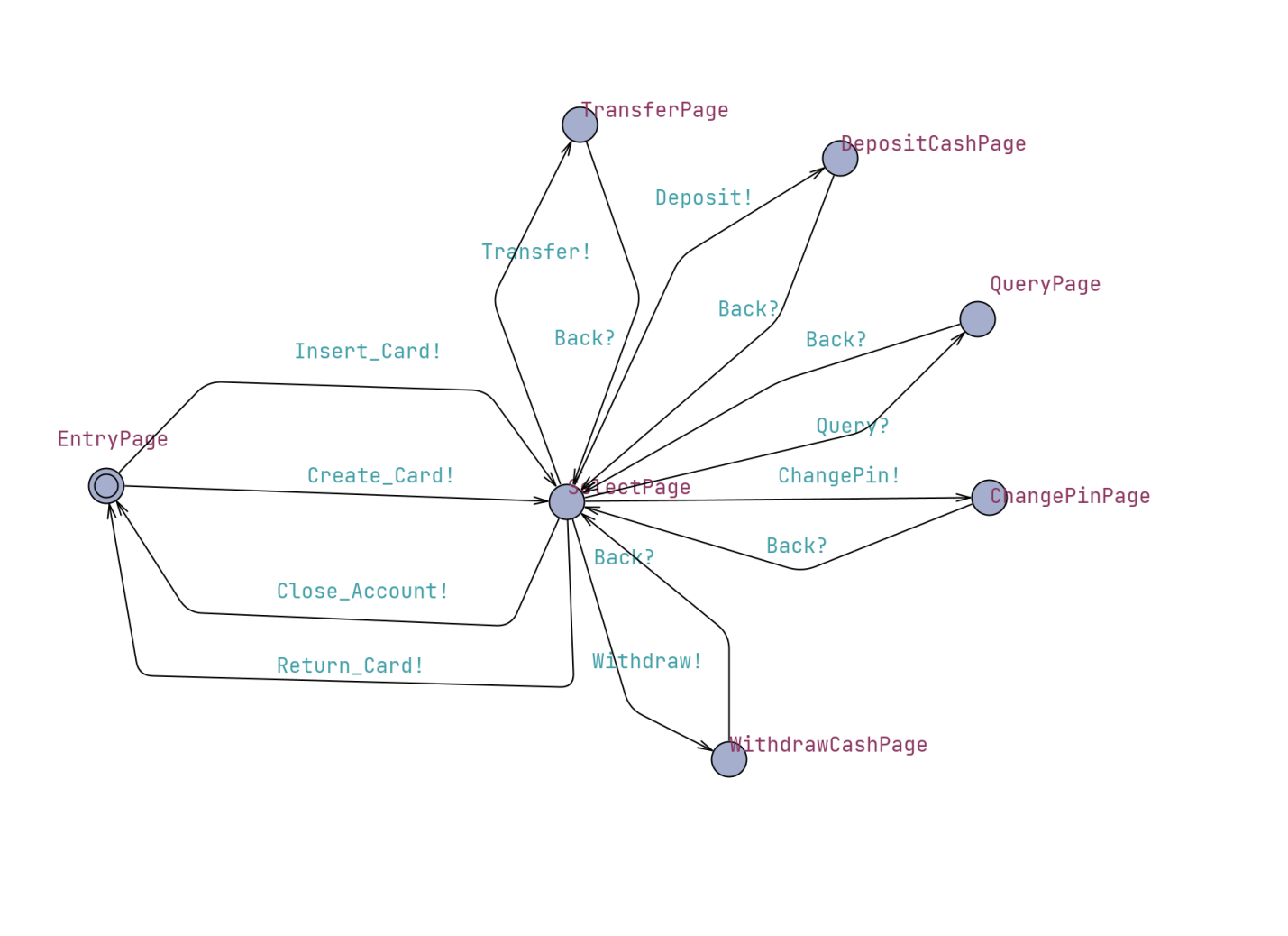
### T4.2: Assumptions

In the real world, there are several cases that can lead to a deadlock. For example, the account holder might not enter a valid PIN, or the ATM might fail to communicate with the bank server, etc. Since these kinds of deadlocks are considered valid in the rules but will affect our validation for invalid deadlocks, some prevention measures are used in our validation model to avoid these valid deadlocks and allow continuous simulation, which may make it slightly different from the development model. These measures are based on the following facts:

* We assume the PIN checking operation after one inserting a card on an ATM always succeeds.
* In any single simulation, outsiders’ operation may not affect the simulation. For example, when we simulate the interaction between an app and the processor, we do not consider the compulsory log out caused by another app logging in to the same account. Meanwhile the updates of the data base by other app or atm are also ignored since we implemente a static data base.

### T4.3: Banking System Model

#### T4.3.1: ATM



The ATM mainly has seven states, each of which corresponds to a Page on the ATM UI.

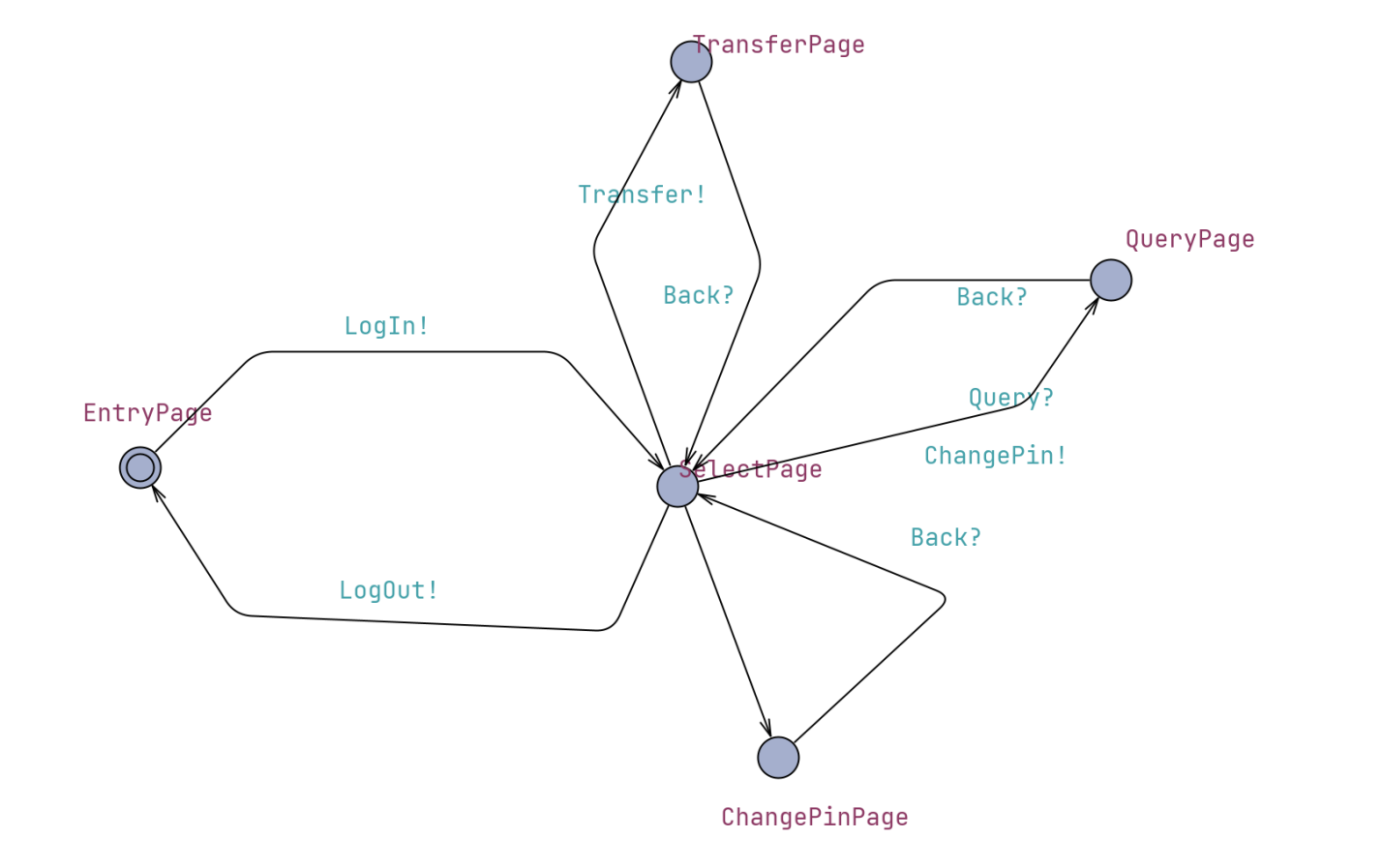
The Entry Page implies that there is no user using the ATM and thus no service is available before an authorized party activing the Service by Inserting a card or create an account.

The Select Page allows users to choose services, and once a service is chosen, the UI will switch to the corresponding page. Note that when a user choose to return card or close account, the ATM UI will go back to the Entry Page immediately and disable all services.

The TransferPage, DepositCashPage, QueryPage, ChangePinPage and WithdrawCashPage do corresponding operation and will at last back to the Select Page when the request is handled.

#### 

#### T4.3.2: APP



The APP mainly has seven states, each of which corresponds to a Page on the APP UI.

The Entry Page implies that there is no user using the APP and thus no service is available before an authorized party activing the Service by Inserting a card or create an account.

The Select Page allows users to choose services, and once a service is chosen, the UI will switch to the corresponding page. Note that when a user choose to log out, the APP UI will go back to the Entry Page immediately and disable all services.

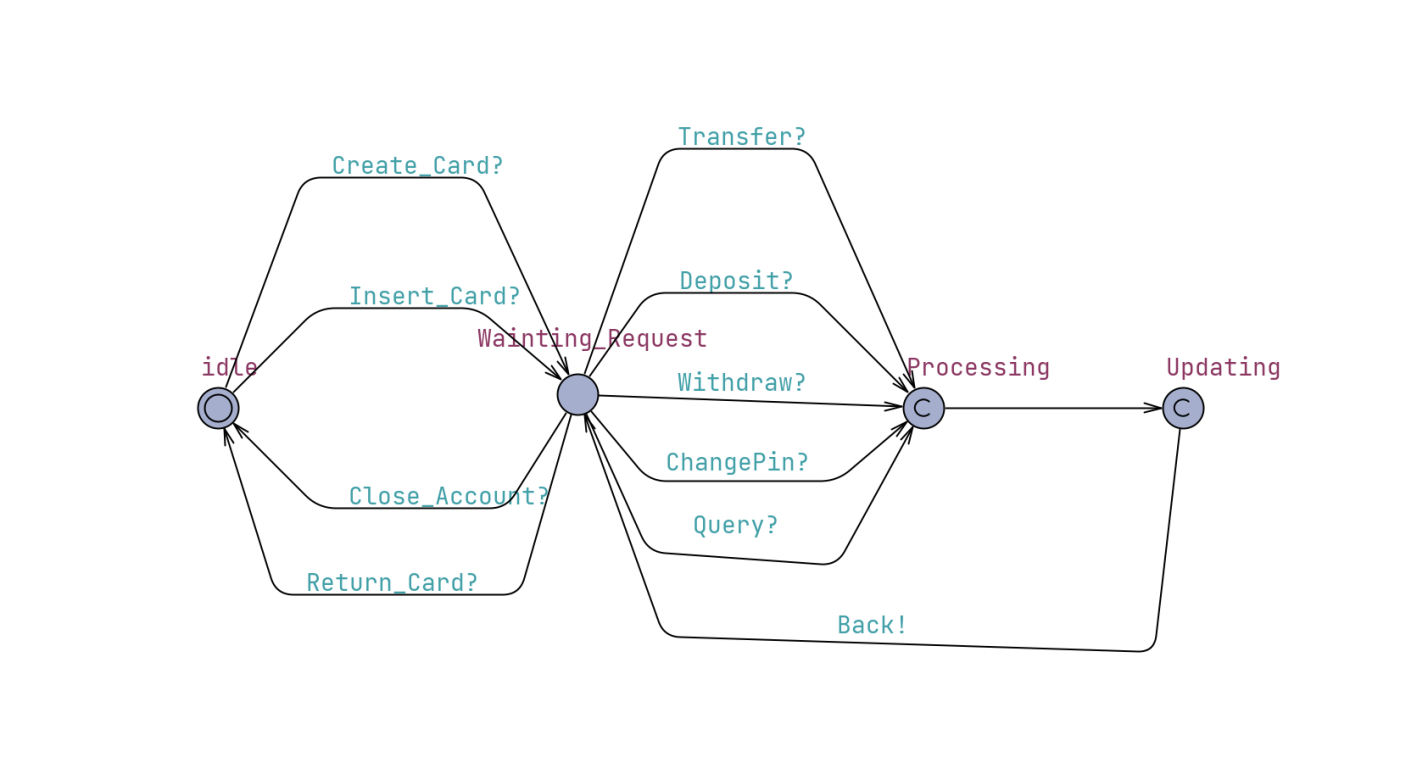
The TransferPage, QueryPage,and ChangePinPage do corresponding operation and will at last back to the Select Page when the request is handled.

#### 

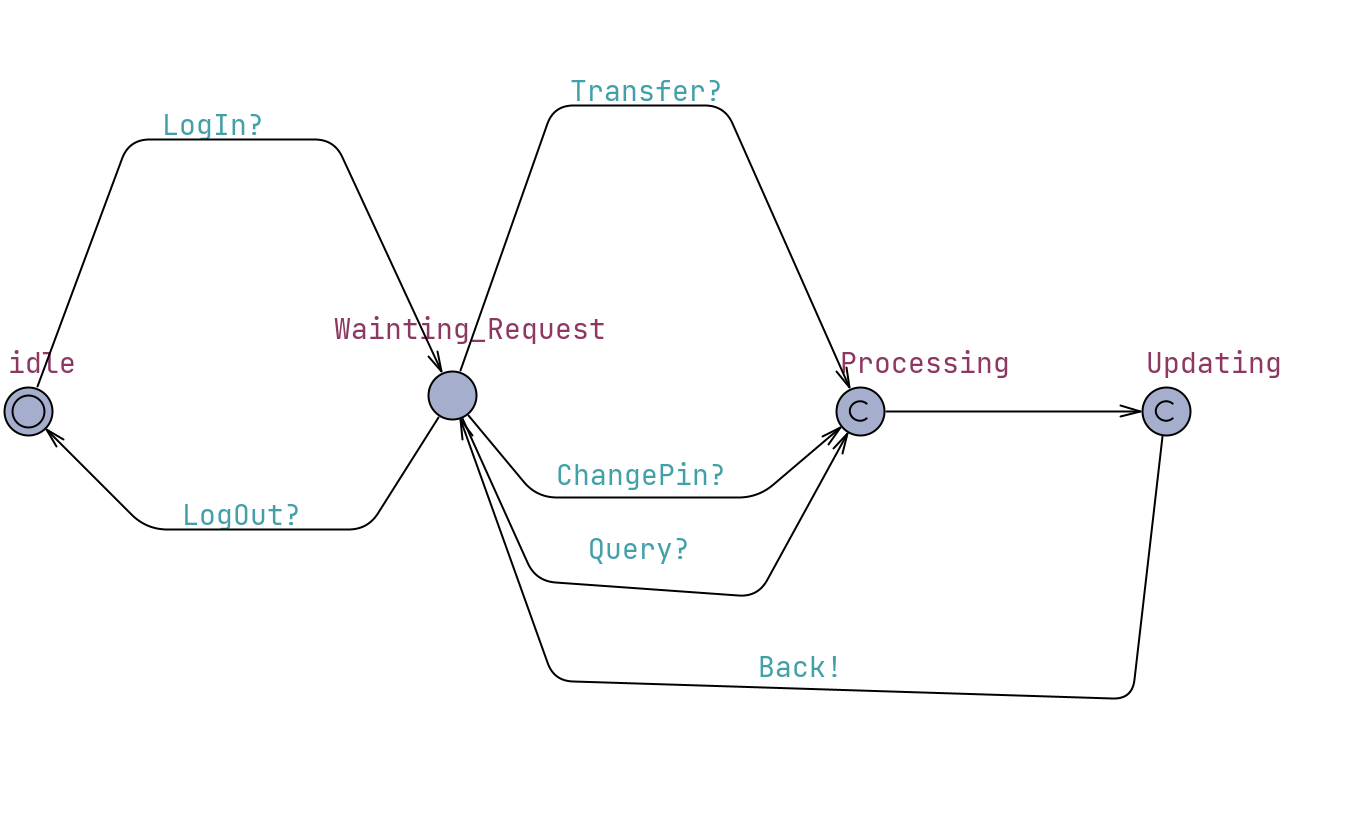
#### T4.3.3: Processor

Since we consider ATM and APP separately, for convenience we separate the processors’ functions correspondingly. The integration of the two constitutes the entire processor.

##### T4.3.3.1 ATM Part



##### T4.3.3.2 APP Part



The processor has four major states: idle, Waiting\_Request, Processoring and Updating.

In the Idle state, the processor is waiting for an external operation to active it.

In the Waiting\_Request State, an authorized party has already actived the service of an app or a ATM, and thus the processor is waiting for potentail request.

In the Processing State, the processor process the message and then go to the Updating State.

In the Updating State, the processor update the data in the database and send a Back siganl to inform the UI that the requset has been handled. Then the processor go to the Waiting\_requst State waiting for the next request.

#### T4.3.4: Check Properties

##### T4.3.4.1

|  |  |
| --- | --- |
| Property | A[] not deadlock |
| Description | There is no deadlock in the intergation of ATM and Processor |
| Result | Passed |

##### T4.3.4.2

|  |  |
| --- | --- |
| Property | A[] ATM1.SelectPage imply Processor.Wainting\_Request |
| Description | The processor is waiting for a request while the ATM1 is on the Select Page |
| Result | Passed |

##### T4.3.4.3

|  |  |
| --- | --- |
| Property | A[] not deadlock |
| Description | There is no deadlock in the intergation of APP and Processor |
| Result | Passed |

##### T4.3.4.4

|  |  |
| --- | --- |
| Property | A[] APP1.SelectPage imply Processor.Wainting\_Request |
| Description | The processor is waiting for a request while the APP1 is on the Select Page |
| Result | Passed |