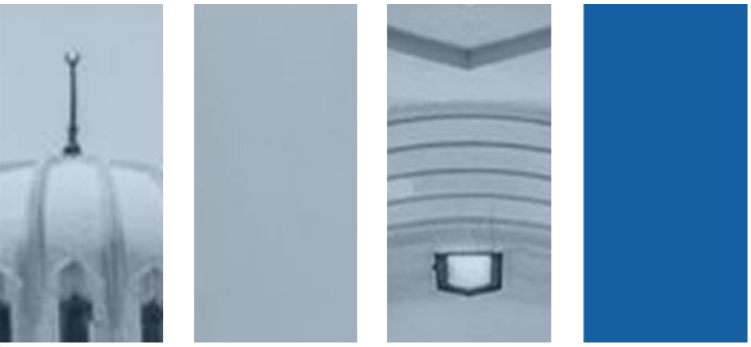


Ain Shams University

Faculty of Engineering

Computer and Systems Engineer

Project Name: ATM Machine

Course Code: CSE337s

Course Name: Software Testing

Team Members:

Ahmed Magdi Mostafa Hosni	1808714
Ahmed Magdy Fahmy Mohamed	1805862
Ahmed Abdallah Mansour Abdel Lateif	1809252
Omar Mohamed Diaaeldin Ibrahim	1802932
Ahmed Mohamed Ahmed Sayed	1804904

Assignment 1

Tester Name: Team 15
Test Environment Details : ATM Machine

White Box Testing using Statement Coverage Technique

Count	Scen. #	Scenario Description	Req #	Test Case#	Test Steps	Test Data	Expected Results	Actual Results	Pass (Y/N)	Tester Name
1	A1	Withdrawal amount greater than Balance	the user must have an account	5	1.Enter The amount of Money to withdraw	<Invalid withdrawal amount>	the transaction doesn't proceed and a message indicating insufficient amount	the trancation proceeds and the balance is shown in negative	N	Omar Mohamed Diaaeldin
2	A2	Withdrawal amount less than Balance	the user must have an account	6	1.Enter The amount of Money to withdraw	<Valid withdrawal amount><Warning! Balance will be empty>	transaction is successful	trancation is successful	Y	Omar Mohamed Diaaeldin
3	B1	User ID	we need a valid user	1	1.Enter your User ID	<Valid User Name>	The user then is able to make a transaction process	The user then is able to make a transaction process	Y	Ahmed Magdy Fahmy
4	B2	User ID not found	we need a invalid user	2	1.Enter your User ID	<InValid User Name>	The user then is unable to make a transaction process	The user then is unable to make a transaction process	Y	Ahmed Magdy Fahmy
5	A3	Withdrawal amount equal Balance	the user must have an account	4	1.Enter The amount of Money to withdraw	<Valid withdrawal amount><Warning! Balance will be empty>	Message indicating "Decline transaction" that the account amount must be at least 100$	proceed transaction and empty the account	N	Ahmed Magdi Mostafa
6	C1	Withdrawal amount more than amount in ATM machine	the user must have an account	10	1.Processing your Request	<Valid withdrawal amount><Warning! Balance will be empty>	Message Indicating "The Transaction doesn't proceed" , that the ATM machine doesn't have enough amount of money in it.	Transaction proceeds	N	Ahmed Abdallah Mansour
7	C2	Withdrawal amount less than lower bound (50$)	the user must have an account	7	1.Processing your Request	<Invalid Withdrawal amount , Click an amount higher than 50$>	Message indicating "Decline transaction" that the lower transaction amount must be at least 50$	Message indicating "Decline transaction" that the lower transaction amount must be at least 50$	Y	Ahmed Abdallah Mansour
8	C3	Withdrawal amount exceeded 10k $(higher bound) at a single transaction	the user must have an account	9	1.Processing your Request	<Invalid Withdrawal amount , Click an amount less than 10k $>	Message indicating "Transaction Failed , you can't withdraw morethan 10k $ at a time"	Message indicating "Transaction Failed , you can't withdraw morethan 10k $ at a time"	Y	Ahmed Mohamed Ahmed Sayed
9	C4	Amount withdrawn is not a multiple of 50	the user must have an account	8	1.Processing your Request	<Amount of withdraw must be a multiple of 50>	Message indicating "Decline transaction" that the withdrawan amount should be at least 50$ or multiples of it	Message indicating "Decline transaction" that the withdrawan amount should be at least 50$ or multiples of it	N	Ahmed Mohamed Ahmed Sayed
10	P1	Wrong Password	user with an existing bank account	3	1.Enter the Correct Password	<Wrong passcode>	an error message indicating wrong passcode	an error message indicating wrong passcode	Y	Ahmed Magdi Mostafa
11	C5	Withdraw more than 15K per day	the user must have an account	11	1.Processing your Request	<Can't withdraw more than 15,000$ per day>	Message indicating "Exceeed Daily Limit , Try again Tomorrow!"	Transaction proceeds	N	Ahmed Magdi Mostafa

Assignment 2

Black box testing

- **Testing after fixing the code after last white box testing (One bug was still unintentionally unfixed so we left it)**

Test 1 | Per transaction

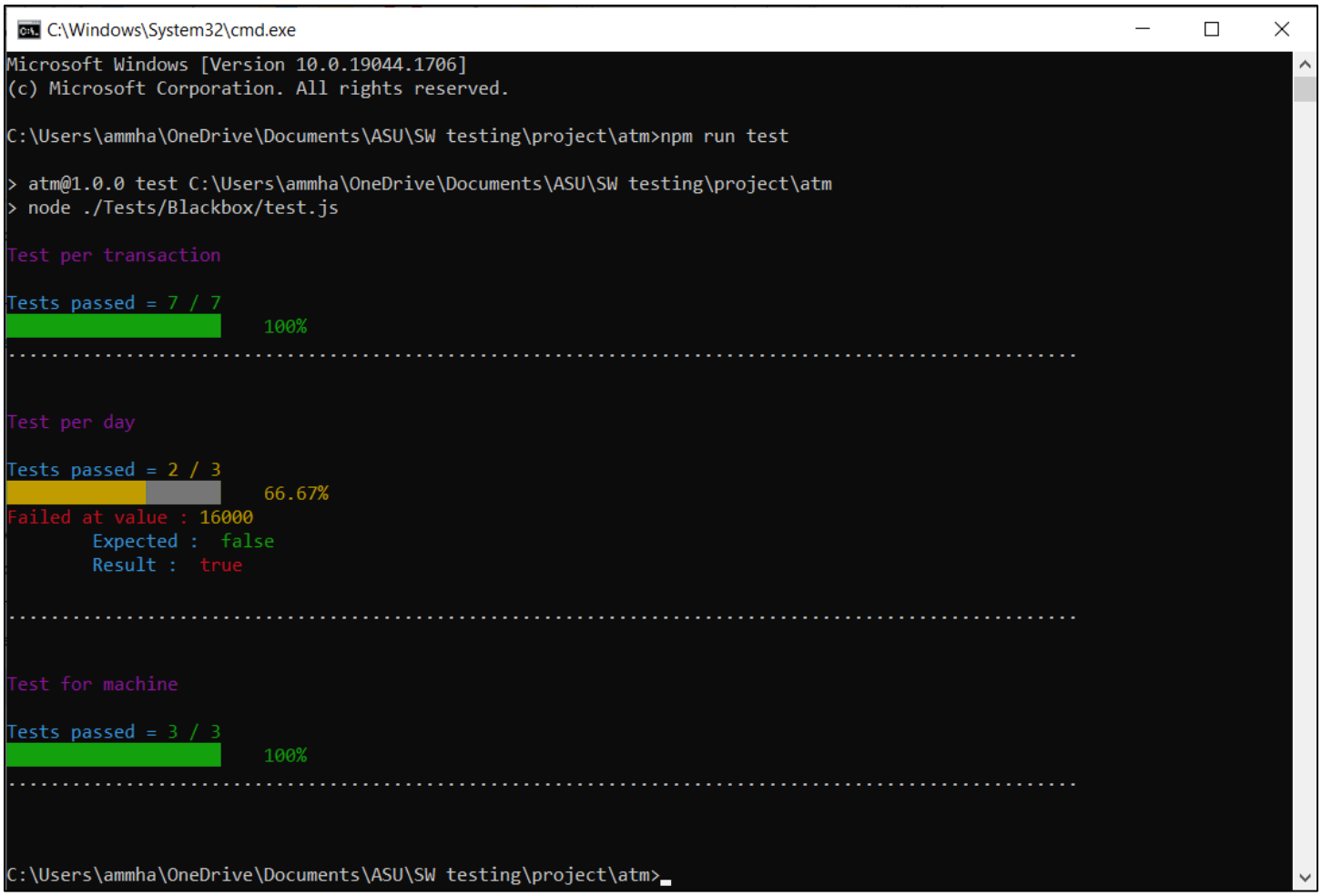
Value	Range	Region name	Expected	Result	Status
-50	t < 0	Region 1	False	False	Pass
45	0 < t < 50	Region 2	False	False	Pass
50	50 ≤ t ≤ 10000, t % 50	Region 3	True	True	Pass
55	50 ≤ t ≤ 10000, t !% 50	Region 3.1	False	False	Pass
5000	50 ≤ t ≤ 10000, t % 50	Region 3	True	True	Pass
10000	50 ≤ t ≤ 10000, t % 50	Region 3	True	True	Pass
12000	10000 < t	Region 4	False	False	Pass

Test 2 | Per day

Value	Range	Region name	Expected	Result	Status
12000	t ≤ 15000	Region 1	True	True	Pass
15000	t ≤ 15000	Region 1	True	True	Pass
16000	15000 < t	Region 2	False	True	Fail

Test 3 | For machine

Value	Range	Region name	Expected	Result	Status
4500	t ≤ 5000	Region 1	True	True	Pass
5000	t ≤ 5000	Region 1	True	True	Pass
5500	5000 < t	Region 2	False	False	Pass



Test after changing one condition from (amount % 50 != 0) to (amount % 50 == 0) so all values divisible by 50 will fail which is the opposite of expected

Test 1 | Per transaction

Value	Range	Region name	Expected	Result	Status
-50	t < 0	Region 1	False	False	Pass
45	0 < t < 50	Region 2	False	False	Pass
50	50 ≤ t ≤ 10000, t % 50	Region 3	True	False	Fail
55	50 ≤ t ≤ 10000, t !% 50	Region 3.1	False	True	Fail
5000	50 ≤ t ≤ 10000, t % 50	Region 3	True	False	Fail
10000	50 ≤ t ≤ 10000, t % 50	Region 3	True	False	Fail
12000	10000 < t	Region 4	False	False	Pass

Test 2 | Per day

Value	Range	Region name	Expected	Result	Status
12000	t ≤ 15000	Region 1	True	False	Fail
15000	t ≤ 15000	Region 1	True	False	Fail
16000	15000 < t	Region 2	False	False	Pass

Test 3 | For machine

Value	Range	Region name	Expected	Result	Status
4500	t ≤ 5000	Region 1	True	False	Fail
5000	t ≤ 5000	Region 1	True	False	Fail
5500	5000 < t	Region 2	False	False	Pass

```
Test per transaction

Tests passed = 3 / 7
                                    42.86%
Failed at value : 50
        Expected :  true
        Result :  false

Failed at value : 55
        Expected :  false
        Result :  true

Failed at value : 5000
        Expected :  true
        Result :  false

Failed at value : 10000
        Expected :  true
        Result :  false

--------------------------------------------------------------------------------


Test per day

Tests passed = 1 / 3
                                    33.33%
Failed at value : 12000
        Expected :  true
        Result :  false

Failed at value : 15000
        Expected :  true
        Result :  false

--------------------------------------------------------------------------------


Test for machine

Tests passed = 1 / 3
                                    33.33%
Failed at value : 4500
        Expected :  true
        Result :  false

Failed at value : 5000
        Expected :  true
        Result :  false

--------------------------------------------------------------------------------
```

➢ Testing Code

```
const { withdraw, resetUser, resetATM, print } = require("./helperFunctions");

//Test per transaction
const testPerTransaction = async () => {
    const testCases = [
        { value: -50, expected: false },
        { value: 45, expected: false },
        { value: 50, expected: true },
        { value: 55, expected: false },
        { value: 5000, expected: true },
        { value: 10000, expected: true },
        { value: 12000, expected: false },
    ];
    const fail = [];
    for (let test of testCases) {
        await resetUser();
        await resetATM();
        const result = await withdraw(test.value);
        if (result != test.expected) fail.push(test);
    }
    return {n: testCases.length, fail};
};

//Test per day
const testPerDay = async () => {
    const testCases = [
        { value: 12000, expected: true },
        { value: 15000, expected: true },
        { value: 16000, expected: false },
    ];
    const fail = [];
    for (let test of testCases) {
        await resetUser();
        await resetATM();
        await withdraw(10000);
        const result = await withdraw(test.value - 10000);
        if (result != test.expected) fail.push(test);
    }
    return {n: testCases.length, fail,};
};
```

```
//Test for machine
const testForMachine = async () => {
    const testCases = [
        { value: 4500, expected: true },
        { value: 5000, expected: true },
        { value: 5500, expected: false },
    ];
    const fail = [];
    for (let test of testCases) {
        await resetUser();
        await resetATM(5000);
        const result = await withdraw(test.value);
        if (result != test.expected) fail.push(test);
    }
    return {
        n: testCases.length,
        fail,
    };
};

const runTest = async () => {
    let result;
    //Test per transaction
    result = await testPerTransaction();
    print("Test per transaction", result);

    //Test per day
    result = await testPerDay();
    print("Test per day", result);

    //Test for machine
    result = await testForMachine();
    print("Test for machine", result);
};

runTest();
```

Assignment 3

Integration testing & ADUP Coverage

- **ADUP Coverage:**
  - **DU Pairs:**

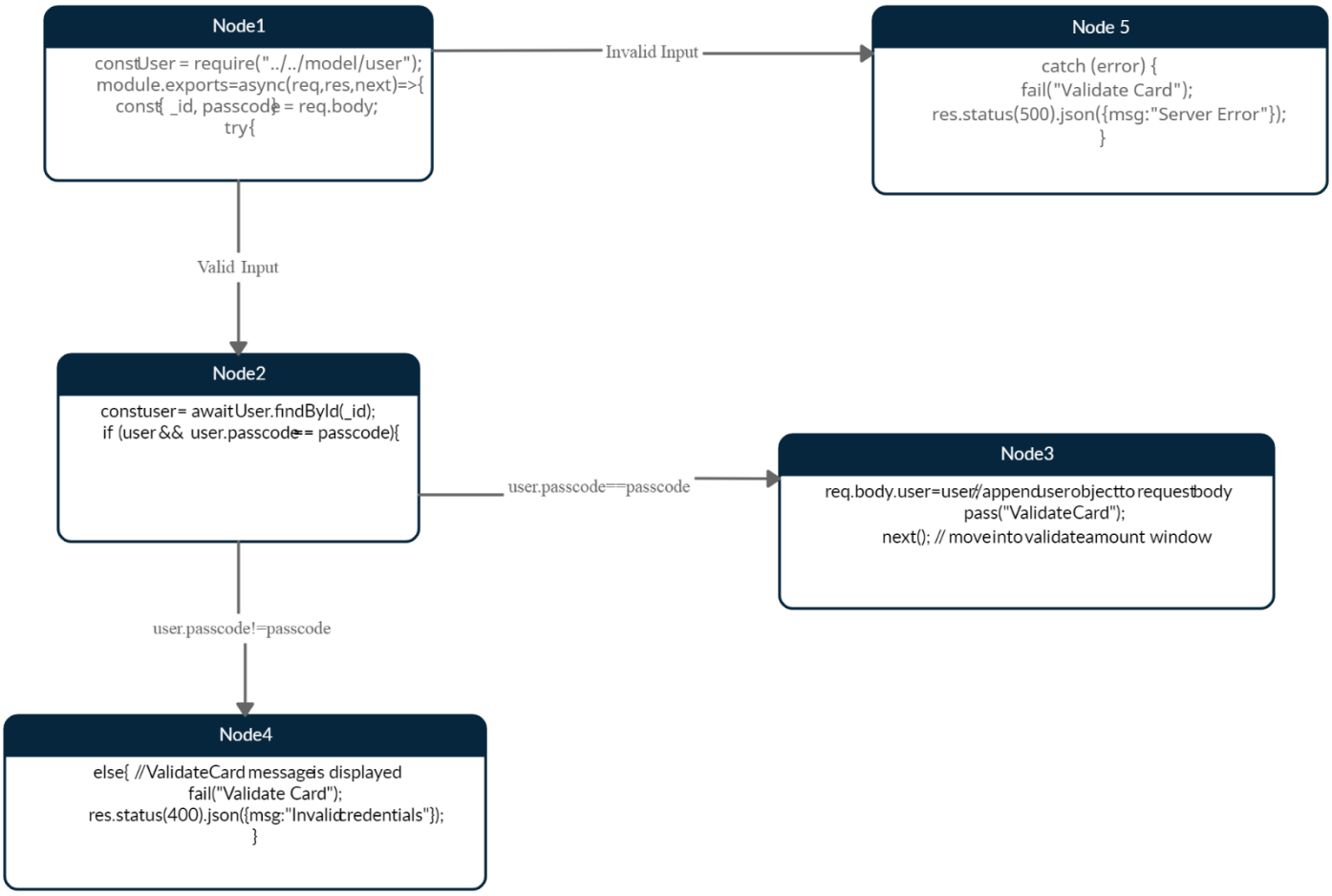


Figure 1: Flowchart of Validate Card function

For Variable **req.body** : variable req.body is a computational use (c-use)

DU-Pair	Paths
(1,3)	<1,2,3>

For Variable **passcode**: variable req.body is a predicate use (p-use)

DU-Pair	Paths
(1,2)	<1,2>
(1,<1,2,3>)	<1,2,3>
(1,<1,2,4>)	<1,2,4>

- **Test Cases:**
  - **Variable req.body:**
  - Consider a test case that executes the path: t1:<1,2,3> which is

```
await withdraw(3,12000);
```

DU-Pair	Paths
(1,3)	<1,2,3> ✓

- **Variable passcode:**
- Consider test cases that executes the path: t1:<1,2,3> , t2: <1,2,4> , which is

```
await withdraw(3,12000);
await withdraw(2,100,2542);
```

DU-Pair	Paths
(1,2)	<1,2>✓
(1,<1,2,3>)	<1,2,3>✓
(1,<1,2,4>)	<1,2,4>✓

Therefore , all ADUP coverages has been satisfied.

- **Integration Testing:**
  We will apply top down approach using **Stubs** we will have 4 stubs: (validateCard/validateAmount/validateATM/withdraw)
- validateCard stub code:

```
const {stubPassed: pass,stubFailed:fail} =
require("../../Blackbox/helperFunctions");
const User  = require("../../../model/user");
module.exports=async(req,res,next)=>{
 const { _id, passcode } = req.body;
 try {
     const user = await User.findById(_id);
     if (user && user.passcode == passcode){
        req.body.user=user; //append user object to request body
        pass("Validate Card");
        next();
     }
     else{
         fail("Validate Card");
         res.status(400).json({msg:"Invalid credentials"});
     }
 }catch (error) {
    fail("Validate Card");
    res.status(500).json({msg:"Server Error"});
 }
}
```

- validateAmount stub code:

```
const {stubPassed: pass,stubFailed:fail} =
require("../../Blackbox/helperFunctions");
const Trans = require("../../../model/trans");

module.exports=async(req,res,next)=>{
 const {amount,_id,user} = req.body;
 const date = new Date().toISOString().substring(0, 10);
 const error = () => {
    fail("Validate Amount");
    res.status(400).json({msg:"Invalid withdraw amount"});
 }
 try {
    if (user.balance - amount < 100) error();
    else if (amount < 50) error();
    else if (amount % 50 != 0) error();
    else if (amount > 10000) error();
    else {
        const trans = await Trans.find({ by: _id, date, action: 0
});
        let withdrawnToday = 0;
        trans.forEach((t) => (withdrawnToday += t.amount));
        if (withdrawnToday > 15000) error();
        else{
            req.body.date = date;
            pass("Validate Amount");
            next();
        }
 }
```

- validateATM stub code:

```
const {stubPassed: pass,stubFailed:fail} =
require("../../Blackbox/helperFunctions");
const ATM = require("../../../model/atm");
module.exports=async(req,res,next)=>{
 const {amount} = req.body;
 const atm = await ATM.findOne({}).exec();
 try {
    const balances = atm.balance;
    const atmBalance = Object.entries(balances).reduce(
        (total, [key, value]) => total + value * +key.substring(1),
        0
    );

    if (amount > atmBalance){
        fail("Validate ATM");
        res.status(400).json({msg:"Insufficient amount in ATM"});
    }
    else{
        pass("Validate ATM");
        next();
    }

 }catch (error) {
    fail("Validate ATM");
    res.status(500).json({msg:"Server Error"});
 }
}
```

- withdraw stub code:

```
const {stubPassed: pass,stubFailed:fail} =
require("../../Blackbox/helperFunctions");
const Trans = require("../../../model/trans");
module.exports=async(req,res,next)=>{
 const { _id, user,date,amount} = req.body;
 try {
    const oldBalance = user.balance;
    user.balance -= amount;
    const newTrans = new Trans({
        by: _id,
        amount,
        date,
        action: 0,
    });
    Promise.all([newTrans.save(), user.save()]).then((result) =>{
        pass("Withdraw");
        res.json({
            amount,
            oldBalance,
            balance: user.balance,
        })
    }
    );
 }catch (error) {
    fail("Withdraw");
    res.status(500).json({msg:"Server Error"});
    console.log(error);
 }
}
```

- Test script :

```
const axios = require("axios").default;
axios.defaults.baseURL = "http://localhost:3000";
withdraw = async (number, amount, passcode) => {
    const data = {
        _id: "6236774d1a42dbfef22613c6",
        passcode: passcode || 1234,
        amount,
    };
    let result;
    try {
        result = await axios.post(`/test/testWithdraw/${number}`,
data);
    } catch (err) {
        result = err.response;
    }
    return result.data.success;
};

const test = async ()=>{
    console.log("Tests started")
    await withdraw(1,100, 1234,"badID");
    await withdraw(2,100, 2542);
    await withdraw(3,12000);
    await withdraw(4,5500);
    await withdraw(5,1000);
    console.log("Tests finished")
}
```

- Routing script:

```
const router = require("express").Router();

const stub1 = require("./stubs/validateCard");
const stub2 = require("./stubs/validateAmount");
const stub3 = require("./stubs/validateAtm");
const stub4 = require("./stubs/withdraw");

router.post("/testWithdraw",stub1,stub2,stub3,stub4);


module.exports = router;
```