TEST PLAN

for

Egret

A Transport Company Computerization Software

Version 1.0 approved

Prepared by

Parth Jindal
Mayank Kumar
Shristi Singh

Department of Computer Science and Engineering,
IIT Kharagpur

March 27, 2021

Table of Contents

1 Introduction **4**

2 Objective **4**

3 Software Risk Issues **4**

4 Features to be tested **5**

5 Features not to be tested **5**

6 Approach **6**
6.1 Testing Levels 6
6.1.1 Unit Testing 6
6.1.2 Application Testing 6
6.2 Test Tools 6
6.3 Test Completeness 7

7 Testing Frontend **7**

8 Testing Backend **7**
8.1 Address Class 8
8.1.1 Unit Test 8
8.1.2 Test Scenarios 9
8.2 Bill Class 9
8.2.1 Unit Test 9
8.2.2 Test Scenarios 10
8.3 Consignment Class 10
8.3.1 Unit Test 10
8.3.2 Test Scenarios 12
8.4 Truck Class 12
8.4.1 Unit Test 12
8.4.2 Test Scenarios 14
8.5 BranchOffice Class 14
8.5.1 Unit Test 14
8.5.2 Test Scenarios 16
8.6 HeadOffice Class 16
8.6.1 Unit Test 16
8.6.2 Test Scenarios 16

8.7 Employee Class 17
8.7.1 Unit Test 17
8.7.2 Test Scenarios 18
8.8 Manager Class 18
8.8.1 Unit Test 18
8.8.2 Test Scenarios 19
8.9 Unit-Test Authorization 20
8.9.1 Registration 20
8.9.2 Login 20
8.9.3 Forgot Password 20
8.9.4 Logout 20

9 Testing Web server and Database **21**
9.1 Error Handling 21
9.2 Unit-test login 21
9.3 Unit-test Register 21
9.4 Unit-test Create Consignment 21
9.5 Unit-test Dispatch Truck 22
9.6 Unit-test View Statistics 22
9.7 Unit-test Database 22

10 Testing Application **22**

1 Introduction

The Test Plan is designed to prescribe the scope, approach, resources, and schedule of all testing activities of the project **Transportation Company Computerization Software**.

The plan identify the items to be tested, the features to be tested, the types of testing to be performed, the resources and the risks associated with the plan.

2 Objective

Objective of Test plan is to define the various Testing strategies and testing tools used for complete Testing life cycle of this project. All the features to be tested are mentioned in detail along with clear objectives to be met for the same.

3 Software Risk Issues

There are several parts of the project that are not within the control of the TCCS project but have direct impact on its correct working and must be checked as well.Untimely logging of consignment by company employees proves as an operational risk to the software.Legacy Support for all inherent third-party libraries remains a functional risk to the Software and the evolving demands of the client.Database exposures are also detrimental to the service.

4 Features to be tested

Module Name	**Applicable Roles**	**Description**
Registration	Manager	A **Manager** can register himself and other employees in the application.
Login	Manager Employee	Both **Manager** and **Employee** can log in using their email and password.
Forgot Password	Manager Employee	Both **Manager** and **Employee** can reset their password using the link mailed to them.
Consignment Status	Consignment	The **status** of a **Consignment** changes as it is **PENDING**, **ALLOTTED** or **DELIVERED**.
Assigned Trucks	Consignment	A Consignment gets one or more trucks assigned to it according to its volume.
Truck Status	Truck	The **status** of a **Truck** changes as it is **AVAILABLE**, **ASSIGNED** or **ENROUTE**.
Add Consignment	Truck Consignment	A **Consignment** is added to a **Truck**.
View Consignments	Truck Consignment	A list of **Consignment** allotted to the **Truck** is displayed.
Add Employee	Manager Employee Branch	A **Manager** can add **Employee(s)** to any **Branch**.
Add Truck	Truck Branch	A **Truck** can be added to a **Branch**.
Add Transaction	Branch Bill	Transaction details of a **Bill** is added to the concerned **Branch**.
Receive Truck	Employee Branch Truck	An **Employee** can receive a **Truck** coming to his **Branch**.
Request for Truck	Manager Employee Truck Branch	An **Employee** can send a request to the **Manager** to add **Truck** in his **Branch**.
Dispatch Truck	Employee Truck	An **Employee** can dispatch a **Truck** when it is full.
View Statistics	Manager Branch Consignment Truck	The **Manager** can check various statistics related to **Consignment**, **Truck** and **Branch**.
Change Rate	Manager	The **Manager** can change rate charged for placing consignment.
Buy New Truck	Manager Branch Truck	The **Manager** can buy a new **Truck** for a **Branch**.
Query Database	Manager Branch Employee Truck	**Data** should be committed and queried from a Database.
View Functions	Employee, Manager	**Employee** and **Manager** can visit

5 Features not to be tested

- All third-party libraries will not be included in the test plan.
- The Hardware will also not be tested as mentioned in the software requirement specifications.
- The GUI of the application will not be put under rigorous testing.

6 Approach

6.1 Testing Levels

There are many different testing levels which help to check the behavior and performance for software testing. Here, the testing of TCCS project will consist only of Unit Tests and Application Tests.

6.1.1 Unit Testing

All the individual units or components of the software will be tested in the Unit Testing so as to validate that each unit of the software code performs as expected. In order to do this, we have written a section of code for every class and its functions. Thus, we will be isolating a section of code and verifying its correctness.

6.1.2 Application Testing

Application testing is a testing carried out to verify the behavior of the complete system. It is done after all the critical defects have been corrected. Application testing is performed to evaluate the end-to-end system specifications and verify the interactions between the modules of the software system. Here, we will be testing all the features of the software together and verifying the correctness of the entire code as a single unit, unlike the unit testing where we have to isolate a section of code and test it. This will be our final test to verify that the product meets the specifications mentioned in the SRS.

6.2 Test Tools

Test Automation and code coverage analysis libraries will be used for testing the application.

- **pytest** framework will be used for writing clean,readable and automated unit tests.This will provide clear statistic for all test cases and code coverage.
- **unittest** framework will also be used in compliance with pytest to write structural application test. using unittest's Modular functionality for writing tests in well-defined classes.

6.3 Test Completeness

The testing will be considered as complete if the following criteria are met:

- 100% test coverage
- All Manual & Automated Test cases executed
- Test all model classes, and all the tables in database
- Test Web server is handling all application requests without any service denial.
- All open bugs are fixed

7 Testing Frontend

Here, GUI, functionality and usability of the web application will be tested. It is ensured that testing is done for overall functionalities so that the presentation layer of the web application is defect free and runs smoothly.

- All front end forms will be tested and navigation bars will be tested to check correct routing and urls.
- All front end submit buttons will be tested along with JS animations in the website.

8 Testing Backend

All Model Classes will be tested in sync with the Database. Since an Object Relation Mapper is used with all model classes,hence there is no need to test database,rather only test commits to Database and querying directly from the model class

8.1 Address Class

8.1.1 Unit Test

- **Address(city,addrLine,zipCode)**
  args:
  - city: string, required
  - addrLine: string, default = None
  - zipCode: string, default = None

  Note: All attributes of Address Class are associated with a property hence all getter setter interfaces can be dealt with the property themselves
  All objects must be initialized with given arguments. All attributes must be asserted to the input values. Overloaded signatures will be checked.

- **getID()**
  return:
  - int

  Assert all id's are unique
  Assert **id** of first constructed object starts from 1

- **getCity()**
  return:
  - string

  Assert if return string is same as the city given while constructing the object

- **getAddressLine()**
  return:
  - string

  Assert if return string is same as the addrLine string given to the constructor

- **getZIP()**
  return:
  - string

  Assert if return string is same as the zipCode given to the constructor

8.1.2 Test Scenarios

Validation errors should be received when invalid data is provided to input arguments. Hence, all negative-test cases will be dealt with proper validation errors.

- Check that the length of city is not more than 64
- Check that the length of addressLine is not more than 128
- Check that the length of zipCode is 6 and each character is a digit

8.2 Bill Class

8.2.1 Unit Test

- **Bill(date, amount, paymentID)**
  args:
  - date: Date, default = today
  - amount: int, required
  - paymentID: string, required

  All objects must be initialized with given arguments. All attributes must be asserted to the input values. Overloaded signatures will be checked.
  id of the object will be asserted to check if all objects have unique id

- **getDate()**
  return:
  - Date

  Assert if return Date is same as the date while constructing the object

- **getAmount()**
  return:
  - int

  Assert if return amount is same as the amount while constructing the object

- **getPaymentID()**
  return:
  - string

  Assert if returned string is same as the paymentID set while constructing the object

8.2.2 Test Scenarios

Validation errors should be received when invalid data is provided to input arguments.Hence All negative-test cases will be dealt with proper validation errors

- Check that date is not a future date
- Check that amount is not negative
- Check that the length of the paymentID is not more than 10

8.3 Consignment Class

8.3.1 Unit Test

- **Consignment**(volume, senderAdress, receiverAddress, destinationBranchID)
  args:
  - volume : int, required
  - senderAddress: Address, required
  - receiverAddress: Address, required
  - destinationBranchID: int, required

  All objects must be initialized with given arguments. All attributes must be asserted to the input values. Overloaded signatures will be checked.

- **getID**()
  return:
  - int

  Assert if id of the objects are unique and the id of the first constructed object is 1

- **getVolume**()
  return:
  - int

  Assert if return volume is same as the volume given while constructing the object

- **getSenderAddress**()
  return:
  - Address

  Assert if returned address object is equivalent the senderAddress while constructing the object

- **getReceiverAddress()**
  return:
  - Address

  Assert if returned address object is equivalent the receiverAddress while constructing the object

- **getStatus()**
  return:
  - enum 'ConsignmentStatus'

  The consignment's current status may be any of the following
  - PENDING
  - ASSIGNED
  - ENROUTE
  - DELIVERED

  All state transitions will be checked between the above mentioned status. The consignment's default status is PENDING. When all of the consignment's volume has been allotted to a truck, its status is changed to ASSIGNED. As soon as the consignment's truck gets full to be dispatched, it's status is changed to ENROUTE.When the consignment is finally received by the destination Branch, it's status is changed to DELIVERED.

- **getSourceBranchID()**
  return:
  - int

  Assert if return value is equal to the ID of the Branch where consignment was placed

- **getDestinationBranchID()**
  return:
  - int

  Assert if return value is equal to the ID of The Destination Branch

- **setCharge(v)**
  args:
  - v: int

  Assert if return value is equal to the charge calculated for that consignment.This will include different test examples where fair depends on some heuristic over the distance and volume.

- **viewAssignedTrucks()**
  return:
  - Trucks[0..*]

  Assert if return list has same truck(s) as the one(s) assigned for the consignment This includes test cases involving one consignment assigned to a single truck and being assigned to even multiple trucks

8.3.2 Test Scenarios

- Check that the volume is not negative
- Check that senderAddress and receiverAddress are not same
- Check that sourceBranch and destinationBranch are not same
- Check that charge is not negative
- Check that trucks list is not empty when status is not PENDING

8.4 Truck Class

8.4.1 Unit Test

- **Truck()**
  args:
  - plateNo: string,required,unique
  - branchID: int,required

  All objects must be initialized with given arguments. All attributes must be asserted to the input values. Overloaded signatures will be checked.
  Assert **id** is unique and id for first created truck should be one Assert if putting same plateNo. for another truck fails or not.(fail-test)

- **getID()**
  return:
  - int

  Assert if return ID is same as the ID while constructing the object

- **getPlateNo**
  return:
  - string

  Assert if returned string is correct from golden output

- **getBranchID()**
  return:
  - int

  Assert if return value is same as its current Branch's ID

- **getDestinationBranchID()**
  returns:
  - int

  Assert if returned value is same as all the destinationID's of consignments assigned to it,else should be asserted to -1

- **getStatus()**
  return:
  - enum 'TruckStatus'

  Assert if return string is same as its current status while constructing the object The truck's current status may be any of the following
  - AVAILABLE
  - ASSIGNED
  - ENROUTE

  All state transitions will be checked between the above mentioned status. The truck's status is AVAILABLE when it is empty. As soon as a consignment is assigned to the truck, it's status is changed to ASSIGNED. When the truck leaves the source branch, it's status is changed to ENROUTE.

- **isFull()**
  return:
  - bool

  Assert if isFull() returns true when obj.volume = 500 else false

- **getVolume()**
  return:
  - int

  Assert if returned value is the volume it has been assigned till yet.

- **addConsignment(consignment)**
  args:
  - consignment: Consignment

  return:
  - bool

Assert all return values when consignments are added are true till total truck Volume is less than 500 and then no consignments must be added

- **viewConsignments()**
  return:
  - Consignment[0.*]

  Assert all consigment objects are returned which were earlier assigned.

- **emptyTruck()**
  return:
  - Consignment[0.*]

  Assert if Truck's consigments list is empty and its Volume must be zero.In Additionally,its status must also be changed and asserted to AVAILABLE and currentBranch will also be checked

  All setter functions will be tested with appropriate givens in the application code.

8.4.2 Test Scenarios

- Check that length of plateNo is not more than 10
- Check that volume consumed is not negative
- Check that status is not AVAILABLE when truck is full
- Check that status is AVAILABLE when consignments list is empty

8.5 BranchOffice Class

8.5.1 Unit Test

- **BranchOffice**(address,phone)
  args:
  - address: address,required
  - phone: string,required

  Assert All inputs are correctly assigned to the attributes of BranchOffice.
  Assert if **id** of objects are unique and starts from 1.

- **getPhone()**
  returns:
  - string

Assert return string is same as phone no. given as input to the constructor

- **addEmployee**(employee)
  args:
  - Employee

  Assert if employee is added to the employee[] of the Office.Additionally employee's BranchID will also be checked.

- **addTruck**(truck)
  args:
  - truck: Truck

  Assert if truck is added to the trucks[] of the office.Additionally truck's BranchID will also be checked.

- **addTransaction(bill)**
  args:
  - bill: Bill

  Assert if obj.transactions contains the bill given as input

- **getID()**
  returns:
  - int

  Assert object id's are unique and start from 1

- **viewTransactions()**
  returns:
  - Bill[0.*]

  Assert all bills that the office contains must be only the one given before

- **receiveTruck**(truck)
  args:
  - truck: Truck

  Assert if truck is correctly added to the current trucks,truck is empty or not and status of consignment of both truck and consignment.

- **removeTruck**(id)
  args:
  - id: int

Assert Truck status and consignments are correctly removed from the truck and consigments list.It will also be checked if a bill is added to the truck concerning the consignment.

- **isBranch()**
  returns:
  - bool

  Assert if isBranch() returns true

- **getRate()**
  returns:
  - int

  Assert if returned value is as set by the manager.

8.5.2 Test Scenarios

- Check that the length of name is not more than 64
- Check that branchID of all employees in the list is equal to current branchID
- Check that branchID of all trucks in the list is equal to current branchID
- Check that the new employee being added is not already present in employees list
- Check that the new truck being added is not already present in trucks list

8.6 HeadOffice Class

8.6.1 Unit Test

All units test for BranchOffice are to be complied for HeadOffice, only the output for isBranch() function is changed to false.

- **setRate**(rate):
  args:
  - rate: int

Assert if static constant of rate is correctly changed.

8.6.2 Test Scenarios

All the test scenarios for BranchOfiice are to be complied for Headoffice.

- Check that rate is more than 0

8.7 Employee Class

8.7.1 Unit Test

- **Employee(name, email, branchID)**
  args:
  - name: string, required
  - email: string, required
  - branchID: int, required

  All objects must be initialized with given arguments. All attributes must be asserted to the input values. Overloaded signatures will be checked.
  branchID of the object will be asserted to check if all objects have unique id and that the branchID of every object should be greater than 0

- **getName()**
  return:
  - string

  Assert if returned string is same as the name given while constructing the object

- **getEmail()**
  return:
  - string

  Assert if returned string is same as the email given while constructing the object

- **getBranchID()**
  return:
  - int

  Assert if returned integer is same as the Branch ID given while constructing the object

- **set_password(password)**
  args:
  - password: string

  Assert if password_hash is same as the calculated hash value of the password string.

- **check_password(password)**
  args:
  - password: string

return:

- bool

Assert if check_password(password) returns true if the calcuated hash of the password is equal to password_hash

- **RequestForTruck**()
  Assert if Mail is sent to Manager for truck request from employee for its branch. Mail should be tested for statistics about branch and waiting time and current trucks at branch.

- **DispatchTruck**(id)
  args:
  - id: int

  Assert if truck requested for Dispatch should be full.

8.7.2 Test Scenarios

- Check that the length of name is not more than 64
- Check that the length of email is not more than 128
- Check that the length of password_hash is not more than 128

8.8 Manager Class

8.8.1 Unit Test

Manager is a derived class of Employee hence all tests of Employee will also comply with Manager Class.In additon other functions must also be tested w.r.t the Manager class

- **viewWaitingPeriod()**
  returns:
  - int

  Assert if average waiting period returned for a consignment is correct when compared to some golden output

- **viewWaitingTime()**
  returns:
  - int

  Assert if waiting time of truck returned is correct

- **viewIdleTime()**
  returns:

- int

Assert if idle time of truck returned is correct

- **changeRate(rate)**
  args:
  - rate: int

  Assert if rate is changed to the given value

- **buyNewTruck**(branchOffice)
  args:
  - branchOffice: BranchOffice

  returns:
  - Truck

  Assert if the truck is added to the given branch

- **viewTruckStatus(truck)**
  args:
  - truck: Truck

  returns:
  - enum 'TruckStatus'

  Assert if the truck status returned is either of AVAILABLE, ASSIGNED or EN-ROUTE

- **viewTruckUsage(truck)**
  args:
  - truck: Truck

  returns:
  - int

  Assert if the usage time returned for the truck is correct

8.8.2 Test Scenarios

Since Manager is a derived class of Employee, all the test scenarios of Employee are to be complied for Manager. In addition to that,

- Check that viewWaitingPeriod() returns a value not less than 0 and not more than the duration between the start time and the current time
- Check that viewWaitingTime() returns a value not less than 0 and not more than the duration between the start time and the current time

- Check that viewIdleTime() returns a value not less than 0 and not more than the duration between the start time and the current time
- Check that the new rate passed as a parameter in changeRate() is not less than 0

8.9 Unit-Test Authorization

8.9.1 Registration

- Assert if email entered is valid before querying the Database
- Assert that the email entered is not present in the Database
- Assert if the password is encrypted as hash code while being stored

8.9.2 Login

- Assert if email entered by the user has a valid format before querying the Database
- Assert if correct email and password is entered by the user
- Assert if the login is successful and the user is redirected to the branch page when both email and password entered by the user are correct
- Assert if login fails and error message is displayed below the respective text field when the email and password entered do not match
- Assert if the login details are auto-filled when the remember me was selected, and the user tries to login again after he has logged out

8.9.3 Forgot Password

- Assert if email entered by the user has valid format before querying the Database
- Assert if a mail is sent to reset the password, when the email entered is registered in the software
- Assert if an error message is displayed below the text field, when the email entered by the user is incorrect
- Assert if the user is redirected to the login page after the password is reset

8.9.4 Logout

- Assert if the logout is successful
- Assert if the user gets redirected to the home page

9 Testing Web server and Database

This section contains some off the functional testing that will be used in the application.This includes different views of the web-application and other functionalities of the software.This section also includes testing the database by committing objects to a fake database url and quering to check the result from it.

9.1 Error Handling

- All non-mapped urls must return status code 404 when sending POST and GET requests.
- Error messages of status 500 will be checked when producing fake database errors.

9.2 Unit-test login

Unit Test for login will be done by creating a fake client for the flask application and sending requests at the concerned routes of different blueprints.

- GET request will be tested to check 200 status code is received
- POST request will be tested to check if 200 status code is received when provided with valid input
- POST request will be tested for invalid data with response.data must contain "invalid email/password" inside it

9.3 Unit-test Register

- GET request will be tested to check 200 status code is received
- POST request will be tested to check if 200 status code is received when provided with valid input
- POST request will be tested for invalid data with response.data must contain "invalid email/password" inside it

9.4 Unit-test Create Consignment

Unit test for create consignment will be done by sending a post request to its appropriate url.

- POST REQUEST with correct form details should be returned with a 201 status code and url should redirect to the home page for that branch recognized by its url

9.5 Unit-test Dispatch Truck

Unit test will be done by sending a post request.It will be checked if the dispatch signal by the employee is correct or not and then proceed forward. Correct status code must be returned by the same

- POST request with valid scenario should return with a 201 response status and with invalid should have 303

9.6 Unit-test View Statistics

Unit test will be done by sending a GET Request to the concerned url,The response status code should include 200.

9.7 Unit-test Database

Unit-test will be done by creating the class objects of all models and commiting them to the database.Since ORM has been used,testing syntax of SQL commands become redundant.Only querying will be tested by simply quering from the ORM interface classes and checking them against Golden output. All test cases are mentioned in the test suite regarding unit-testing database.

10 Testing Application

- Manager registers in the software
- Manager adds two employees to each branch
- Manager adds two trucks to each branch
- All truck's status is AVAILABLE initially
- An employee logs in using email and password
- Another employee from different branch uses forgot password to reset his password and then logs in
- First employee adds consignments with destination branch of second employee

- A truck's status is changed from AVAILABLE to ASSIGNED as soon as a consignment is placed
- The consignment's status is ALLOTED when a truck is assigned for it
- compute bill
- The truck's status is changed to ENROUTE when it is full and employee gives a green signal
- The truck is received by second employee
- Another employee from a different branch logs in
- Third employee places consignment with volume greater than 500 units to a branch
- Third employee places various consignment to the same branch
- Third employee sends a request to manager to add more trucks
- Manager queries statistics for each branch
- Manager buys new truck for third branch
- The consignment's status is PENDING when no trucks are AVAILABLE in the branch
- The employees receive trucks coming to their branch
- The truck gets emptied when received and its status changes to AVAILABLE
- When a truck is received, all the consignments in it gets its status changed to DELIVERED
- All employees logout from the application