TEST PLAN

for

SoRaSu
A Transport Company Computerization Software

Version 1.0 approved

Prepared by
Soukhin Nayek (21CS10062)
Raj Pajesh Parikh(21CS30039)
Sukhomay Patra(21CS30066)

Department of Computer Science and Engineering, IIT Kharagpur

April 3, 2023

Contents

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 identifies 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

The objective of the Test plan is to define the various Testing strategies and testing tools used for the 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 a direct impact on its correct working and must be checked as well. Untimely logging of consignments 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	Customer Employee	• An employee (including Manager and Driver) can register himself. • A customer can also register himself.
Login	Customer Employee	• An employee (including Manager and Driver) can login himself. • A customer can also login himself.
Forgot Password	Customer Employee	• An employee (including Manager and Driver) can reset his password. • A customer can also reset his password.
Place Consignment	Customer	• A customer can place a consignment with source and destination.
Track Consignment	Customer	• A customer can select and track a consignment.
Invoice generation and history	Customer	• A customer can view his present bill/invoice generated. • A customer can also view his past bills/invoices generated.

Module Name	Applicable Roles	Description
Validate/Approve Consignments	Employee	• An assigned employee can approve a truck with enough consignments.
Receive Truck	Employee	• An assigned employee can receive a particular truck.
View Consignment Status	Employee Manager	• An employee can view the consignment list and PENDING consignments. • A manager can view the consignment[PENDING, APPROVED DISPATCHED, ENROUTE, DELIVERED] list.
Buy a new Truck	Manager	• A manager can add a new truck to a branch with relevant details.
View Truck Status	Manager	• A manager can view the status[AVAILABLE, ASSIGNED, ENROUTE].
Viewing Average Waiting Time of Consignment	Manager	• A manager can view the average waiting time[HOURS] of consignments for a branch.
Viewing Average Truck Idle Time and Truck Usage	Manager	• A manager can view the average idle time[HOURS] and usage time[HOURS] of trucks for a branch.
Viewing Branch Consignment Handling	Manager	• A manager can view the Consignment Handling Statistics of a branch.

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 behaviour and performance of software testing. Here, the testing of the 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 the testing carried out to verify the behaviour 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 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

Manual Testing is done with the help of only one software 'DB Browser for SQLite' for database verification.

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 the database

- Test Web server is handling all application requests without any service denial.
- All open bugs are fixed

7 Testing Front-end

Here, the 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 JavaScript animations/implementations on the website.

8 Testing Back-end

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(addr, city, zipCode)**
  args:
  - addr: string, required
  - city: string, required
  - pincode: string, required
  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.

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 addr is not more than 100
- Check that the length of the city is not more than 30
- Check that the length of pincode is 6 and each character is a digit

8.2 Bill Class

8.2.1 Unit Test

- **Bill(date, amount, branch_id)**
  args:
  - date: DateTime, required
  - amount: Double, required
  - branch_id: int, required
  Note: 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 a unique id

- **getAmount()**
  return:
  - int
  Assert if the return amount is the same as the amount while constructing the object

- **getDate()**
  return:
  - Date
  Assert if the return Date is the same as the date while constructing the object

- **getPaymentID()**
  return:
  - string
  Assert if returned string is the 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.

- Automatic present date insertion
- Check that amount is not negative

8.3 Consignment Class

8.3.1 Unit Test

- **Consignment(volume, senderAdress, receiverAddress, destinationBranchID, sourceBranchID, destinationBranchID)**
  args:
  – volume : int, required

– senderAddress: Address, required
– receiverAddress: Address, required
– sourceBranchID: int, required
– destinationBranchID: int, required
Note: 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:
  - double

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

- **setTruckID()**
  Assert if correct truck id is assigned to the consignment.

- **setStatus(ConsignmentStatus)**
  Assert if correct status is assigned to the consignment.

- **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.

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

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

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

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

- **viewAssignedTruck()**
  return:
  - Truck

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

Note: Some of the functions are implemented in routes.

8.3.2 Test Scenarios

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

8.4 Truck Class

8.4.1 Unit Test

- **Truck(truckNumber, branch_id)**
  args:
  - truckNumber: string, required
  - branch_id: int, required

  Note: 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 the id for the first created truck should be one Assert if putting the same plate number(truckNumber) for another truck fails or not(fail-test).

- **getTruckID()**
  return:
  - int

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

- **getCurrentBranch()**
  return:
  - int

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

- **getDestinationBranch()**
  return:
  - int

  Assert if returned value is the 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, its status is changed to ASSIGNED. When the truck leaves the source branch, its status is changed to ENROUTE.

- **setDestinationBranch()**
  Assert if the correct destination is set with the lookup from the database

- **addVolumeConsumed()**
  Assert if the correct volume is set with a lookup from the database

- **emptyTruck()**
  Assert if the Truck's consignments list is empty and its Volume must be zero. Additionally, its status must also be changed and asserted to AVAILABLE and the current Branch will also be checked by looking it up in the database.

Note: All other setter functions will be tested with manual verifications from the database tables.

8.4.2 Test Scenarios

- Check that length of truckNumber is not more than 10
- Check that volume consumed is not negative
- Check that status is not AVAILABLE when truck is full

8.5 Employee Class

8.5.1 Unit Test

- **Employee(username, name, email_address, branchID, role, password)**
  args:
  - username: string, required
  - name: string, required
  - email_address: string, reuquired
  - branchID: int, required
  - role: string, required

- password: string, required
Note: 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 a unique id and that the branchID of every object should be greater than 0

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

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

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

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

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

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

- **password(password)**
  args:
  - password: string Assert if the password_hash is the same as the calculated hash value of the password string.

- **check_password_correction(attempted_password)**
  args:
  - attempted_user: string return:
  - bool

  Assert if the check password(password) returns true if the calculated hash of the password is equal to the password_hash.

- **approve_truck()**
  Assert if the truck status is changed to DISPATCHED and an e-mail is sent to the AVAILABLE driver.

- **receive_truck()**
  Assert if the truck consignments(truck) are available at the destination branch.

- **dispatch_truck()**
  Assert if the truck consignments(truck) are available at the destination branch

Note: Some of the functions are implemented in routes.

8.5.2 Test Scenarios

- Check that the length of the name is not more than 30
- Check that the length of the username is not more than 30
- Check that the length of the email is not more than 50
- Check that the length of the password hash is not more than 60

8.6 Manager Class

8.6.1 Unit Test

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

- **viewWaitingPeriod()**
  return:
  - int

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

- **viewWaitingTime()**
  return:
  - int

  Assert if the waiting time of the truck returned is correct.

- **viewIdleUsageTime()**
  return:
  - int

  Assert if the idle time of the truck returned is correct. Assert if the usage time returned for the truck is correct.

- **changeRate(rate)**
  args:
  - rate: int Assert if the rate is changed to the given value

- **buyNewTruck(branchOffice)**
  args:
  - branchOffice: BranchOffice Assert if the truck is added to the given branch.

- **viewTruckStatus(truck)**
  args:
  return:
  - enum 'TruckStatus'

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

Note: Some of the functions are implemented in routes.

8.6.2 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 viewIdleUsageTime() 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

8.7 BranchOffice Class

8.7.1 Unit Test

BranchOffice is derived from HeadOffice. Its constructor is tested with construction of HeadOfffice.

- **isBranch()**
  return:
  - bool

  Assert if true always.
- **getPhone()**
  return:
  - string

  Assert return string is same as phone no. given as input to the constructor.
- **getID()**
  return:
  - int

  Assert object id's are unique and start from 1.
- **getRate()**
  return:
  - int

  Assert if returned value is as set by the manager.

8.7.2 Test Scenarios

- No constraint as such

8.8 HeadOffice Class

8.8.1 Unit Test

HeadOffice is constructed from the Office abstract class. Its contructor is tested with construction of Office.

- **isBranch()**
  return:
  - bool

  Assert if return false always.

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

  Assert if the static constant of the rate is correctly changed.

8.8.2 Test Scenarios

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

- Check that rate is more than 0

8.9 Customer Class

8.9.1 Unit Test

- **Customer(username, name, email_address, password)**
  args:
  - username: string, required
  - name: string, required
  - email_address: string, reuquired
  - password: string, required

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

- **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
- **password(password)**
  args:
  - password: string Assert if the password_hash is the same as the calculated hash value of the password string.
- **check_password_correction(attempted_password)**
  args:
  - attempted_user: string return:
  - bool

  Assert if the check password(password) returns true if the calculated hash of the password is equal to the password hash. **track_consignment()**
  Assert if the track function displays the correct track timeline.

Note: Some of the functions are implemented in routes.

8.9.2 Test Scenarios

- Check that the length of the name is not more than 30
- Check that the length of the username is not more than 30
- Check that the length of the email is not more than 50
- Check that the length of the password hash is not more than 60

8.10 Unit-Test Authorization

8.10.1 Registration

- Assert if the email_address 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

8.10.2 Login

- Assert if the username entered by the user has a valid format before querying the Database
- Assert if the correct username and password are entered by the user

- Assert if the login is successful and the user is redirected to the branch page when both username and password entered by the user are correct
- Assert if login fails and an error message is displayed below the respective text field when the email and password entered do not match

8.10.3 Forgot Password

- Assert if the username entered by the user has a 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.

8.10.4 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 of the functional tests 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 querying to check the result from it.

9.0.1 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.

9.0.2 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 a response-data must contain "invalid email/password" inside it

9.0.3 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

9.0.4 Unit-test Create Consignment

Unit test for creating 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.

9.0.5 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. The 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

9.0.6 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.

9.0.7 Unit-test Database

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

10 Testing Application

- Manager registers in the software
- Two employees register in the software
- A customer registers in the software
- All truck's status is AVAILABLE initially
- Customer places a consignment
- Customer views the generated bill
- First Employee approves the consignment
- 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.
- The truck's status is changed to ENROUTE when the driver leaves.
- When a truck is received, all the consignments' status changed to DELIVERED
- The truck is received by second employee.
- Manager buys new truck for third branch
- The consignment's status is PENDING when no trucks are AVAILABLE in the branch
- All employees logout from the application
- Customer logout.