Intro to Software Engineering
ECSE 321

Kevin Chuong, 260742781
Jeremy Davis, 260744431
Oscar Décéus, 260744646
Elias Al Homsi, 260797449

DELIVERABLE #3

Presented to Mr. Daniel Varro
Faculty of Engineering

McGill University
March 18, 2018

Table of Contents

1. Unit Test Plan

Class Tested	Methods which need testing
TreePLEService	public Tree CreateTree
TreePLEService	public Resident findResidentByName
TreePLEService	public Tree findTreeById
TreePLEService	public Resident findResidentByEmail
TreePLEService	public Token checkLogin
TreePLEService	public boolean checkTokenValidity
TreePLEService	public Municipality createMunicipality
TreePLEService	public Resident CreateResident
TreePLEService	public Transaction CreateTransaction

Class Tested	Methods which need testing
TreePLEController	public TreeDto createTree
TreePLEController	public ResidentDto createResident
TreePLEController	public TransactionDto createTransaction
TreePLEController	public MunicipalityDto createMunicipality
TreePLEController	public Token login
TreePLEController	public List<TransactionDto> findAllTransactions

Rationale for which classes are to be tested:

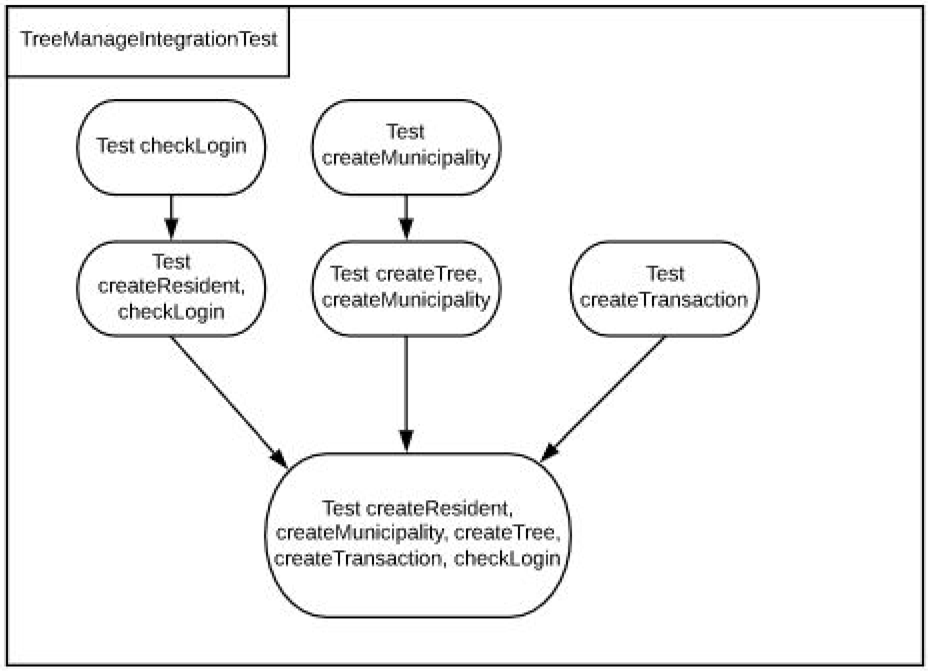
The classes chosen for testing above are chosen based on the following criteria:

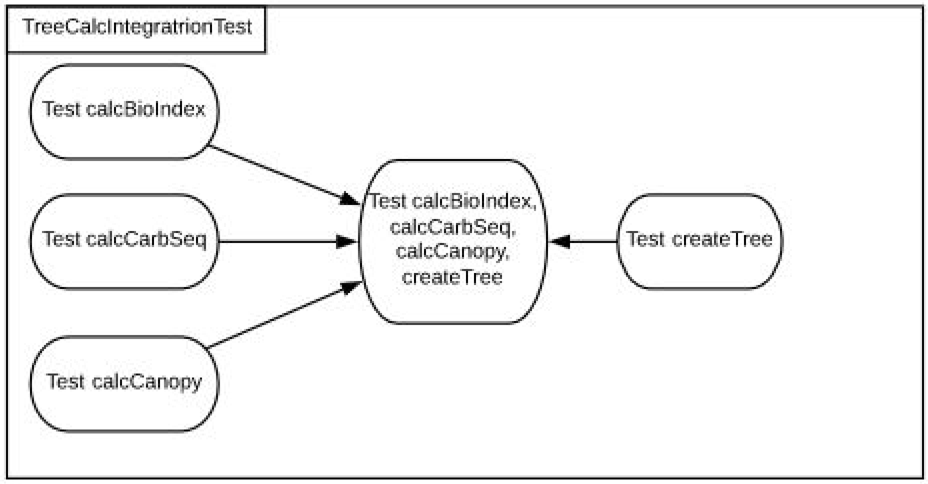
1) The method execution is highly influenced by the user input
2) The output reflects a critical part of the current system status.

To test the controller a stub for service must be created.

2. Integration Strategy

The method chosen for the integration strategy is bottom up. The order required to test depends on the priority and units involved. The following figure presents 2 provisional, but important representations of 2 subsystems.


TreeManageIntegrationTest
Test checkLogin
Test createMunicipality
Test createResident, checkLogin
Test createTree, createMunicipality
Test createTransaction
Test createResident, createMunicipality, createTree, createTransaction, checkLogin



The labels of each representation indicates the name for the marker interface to categorize tests into into marked classes. Additional tests would include unit tests that have not been addresses in this iteration and integration tests taking into account trees (sample data set) from the list provided (Ville De Montreal).

Frontend will use the same strategy (more stubs) than the backend.

No use of tools is anticipated for the creation of integration tests. However, in the event of the need for automated testing, the inclusion of scripts to automate data entry may be introduced.

This method involves lots of driver and a smaller number of stubs. Given initial unit tests, applying this approach allows for an easier and faster transition from individual unit testing to the integration of different parts. Fundamentally, the purpose is to verify and validate sub-system components and test potential problems in the interactions between those components.

3. System Test Plan

Once the Integration testing is done, the whole system would be put under test using exhaustive use cases approach. All use cases would be tested against the system on two different frontends. That is the website and the android application. The list of use cases that is going to be tested:

1. Registering a new Resident under the three different types specified in the description.
2. Login to the system using an existing user
3. Adding a Tree on the map using both front-ends also covering all cases of parameters to the tree.
4. Issuing multiple changes on the current status of trees
5. Creating multiple municipalities.

The system Testing part should also prevent against common attacks like SQL injection and cross site scripting by injecting malicious javascript code. The inputs would preferably be generated using a system fuzzer. Also, The system would be tested to check the number of users that can use the system simultaneously.

Following are two test situations which could be used to test the system, described in further detail:

2. Login to the system using an existing user

The following test would require as input data both the username (email) and password that has been entered by the user. This calls the rest controller function login(@RequestParam(name = "email") String email, @RequestParam(name = "password") String password). This controller function then calls the service function checkLogin(String residentEmail, String password_plaintext), by passing the email and password. If the information entered by the user is correct, a token is generated from the user's email and added to the system to keep track of the user's actions.

5. Creating multiple municipalities.

The following test would require as input data several Strings from the user. Each time a municipality name is entered by the user, the rest controller function createMunicipality(@PathVariable("name") String name) can be called. This passes the municipality name to the service function createMunicipality(String name) in the backend, creating an object for the municipality in the model database. This process would be repeated several times for multiple municipalities.

4. Sample Test Cases for Unit Testing of Backend

TestTreePLEService.java

Business Method	Unit Test Case	Author
public Municipality createMunicipality(String name)	public void testCreateMunicipality()	Jeremy Davis
public Municipality createMunicipality(String name)	public void testCreateMunicipalityNull()	Jeremy Davis
public Municipality createMunicipality(String name)	public void testCreateMunicipalityEmpty()	Jeremy Davis
public Municipality createMunicipality(String name)	public void testCreateMunicipalitySpaces()	Jeremy Davis
public Resident CreateResident(String aName, String aEmail, String aPassword, double lon, double lat, String type)	public void testCreateResident()	Jeremy Davis
public Resident CreateResident(String aName, String aEmail, String aPassword, double lon, double lat, String type)	public void testCreateResidentNull()	Jeremy Davis
public Resident CreateResident(String aName, String aEmail, String aPassword, double lon, double	public void testCreateResidentEmpty()	Jeremy Davis

lat, String type)		
public Resident CreateResident(String aName, String aEmail, String aPassword, double lon, double lat, String type)	public void testCreateResidentSpaces()	Jeremy Davis
public Resident CreateResident(String aName, String aEmail, String aPassword, double lon, double lat, String type)	public void testCreateResidentInvalidLocation()	Jeremy Davis
public Transaction createTransaction(Time aTime, Date aDate, Resident r, Tree t, Transaction.TreeStatus aChangedStatusTo)	public void testCreateTransaction()	Kevin Chuong
public Transaction createTransaction(Time aTime, Date aDate, Resident r, Tree t, Transaction.TreeStatus aChangedStatusTo)	public void testCreateTransactionNull()	Kevin Chuong
public Transaction createTransaction(Time aTime, Date aDate, Resident r, Tree t, Transaction.TreeStatus aChangedStatusTo)	public void testCreateTransactionResidentAndTreeDoNoExist()	Kevin Chuong
public Tree markTree(Tree t, Tree.TreeStatus newStatus)	public void testMarkTree()	Kevin Chuong
public Tree markTree(Tree t, Tree.TreeStatus newStatus)	public void testMarkTreeNull()	Kevin Chuong
public Tree markTree(Tree t, Tree.TreeStatus newStatus)	public void testMarkTreeNotFound()	Kevin Chuong

public Tree CreateTree(TreeSpecies species, TreeStatus status, int diameter, double lon, double lat, Municipality m)	public void testCreateTree()	Oscar Décéus
public Tree CreateTree(TreeSpecies species, TreeStatus status, int diameter, double lon, double lat, Municipality m)	public void testCreateTreeNullOrDefault()	Oscar Décéus
public Tree CreateTree(TreeSpecies species, TreeStatus status, int diameter, double lon, double lat, Municipality m)	public void testCreateTreeInvalidLocation()	Oscar Décéus
public Tree CreateTree(TreeSpecies species, TreeStatus status, int diameter, double lon, double lat, Municipality m)	public void testCreateTreeSmallDiameter()	Oscar Décéus
public Tree CreateTree(TreeSpecies species, TreeStatus status, int diameter, double lon, double lat, Municipality m)	public void testCreateTreeExistingLocation()	Oscar Décéus
public Resident findResidentByEmail(String email)	public void testFindResidentByEmail()	Oscar Décéus
public Resident findResidentByEmail(String email)	public void testFindResidentByEmailEmpty()	Oscar Décéus
public Resident findResidentByEmail(String	public void testFindResidentByEmailNu	Oscar Décéus

email)	ll()	
public Resident findResidentByEmail(String email)	public void testFindResidentByEmailInvalid()	Oscar Décéus
public Tree findTreeById(int id)	public void testFindTreeById()	Kevin Chuong
public Tree findTreeById(int id)	public void testFindTreeByIdNotFound()	Kevin Chuong
createMunicipality()	testCreateMunicipalityWithJavaScriptCode	Elias Al Homsi

TestRestController.java

createMunicipality()	testCreateMunicipality	Elias Al Homsi
createResident()	testCreateResident	Elias Al Homsi
createResident()	testCreateResidentWithNullPassword	Elias Al Homsi
createResident()	testCreateResidentWithNullName	Elias Al Homsi
createResident()	testCreateResidentWithNullEmail	Elias Al Homsi
createResident()	testCreateResidentWithEmptyPassword	Elias Al Homsi
createResident()	testCreateResidentWithEmptyName	Elias Al Homsi
createResident()	testCreateResidentWithEmptyEmail	Elias Al Homsi

createResident()	testCreateResidentWithShortPassword	Elias Al Homsi

5. Updated Work Plan

a) **Major parts of the web-frontend are done.**
b) **The android front-end must be partially complete by the next iteration.**
c) **See github for detailed plan**