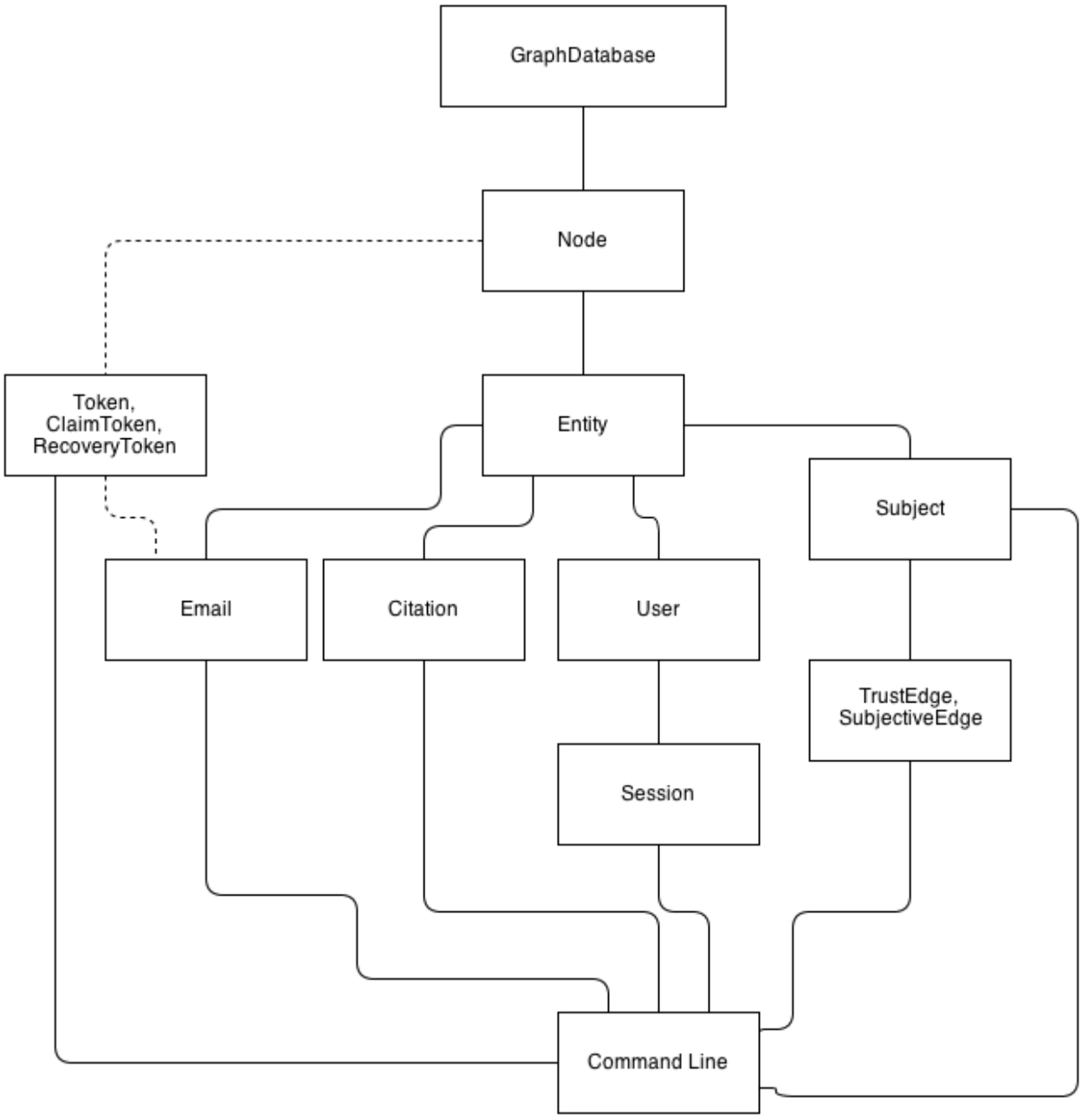
Team 4: Authority Publishing Platform
- Incremental and Regression Test Defect Log -

Date: 09/11/2013 - 09/13/2013

Team: James Miller, Andrew Mack, Christina Atallah, Jason Salter, Mitchell Mounts, Nanthaniel Cherry,

Module Dependency Graph



Regression Strategy

Our regression test strategy was simple and effective. Before turning code into Github.com all team members were required to run `util/test` which would run both unit and integration tests. This meant that once a bug was fixed from one team member, it would not re-introduce a bug into another's code.

Using this method we avoided reintroduction of solved defects into the code base almost completely where in a specific case James merged old code from Christina and overwrote both the integration test and defect fix. It was discovered promptly when doing manual, bigbang testing a few hours later and reverted to a fixed state.

Top-down or Bottom-up

We followed a *mix* of bottom-up and top-down testing. We first used bottom-up testing to leverage our unit tests (for the GraphDatabase, Node, & Entity) and trust that by finishing the integration tests for TrustEdge, Email, Citation, Session and Command Line objects we could save more time than writing stub functions. At the same time we put stub functions into the Command Line module or Cli.java. This let us implement the simple parsing library while other team members could write the integration tests, solving the problem mentioned in lecture about needing to wait for all submodules to be tested first.

Once the Command Line interface was properly functioning our team moved to a fully top down strategy where various api calls were assigned to each team member to implement in Cli.java. A team member would replace the stub function with working code and test manually. We refactored Cli.java many times moving test code to integration tests to find bugs rather than just remove the test code, though some test code was only tested by hand once and not checked into source control (which was a mistake)

Integration tests in test/integrate/Portfolio.java

check_addToPortfolio()

-**Severity**: 3

-**Units integrated for test** : Email, User, Citation

-A new Email object is created by passed a String to app.Email (**input for Email unit**), which should return an Email object (**output of Email unit**). app.User() is called with no parameter to return a new user, which has the password and email set by using the setPassword and

addEmail methods of the User class (Strings are passed as the email and password).
-Now there should be a valid User (**output of User unit**) for which to add citations to their portfolio. The Citation class is passed two Strings, description and resource, to return a new Citation object.
-The user.addToPortfolio method is passed that Citation object, which should now be a part of that user's portfolio.
-Next, an iterator is returned from the user.viewPortfolio method and asserts first that there is a Citation available to the iterator and then tests the contents of the Citation against both garbage and what it should be equal to, to verify it's content is correct.

-**Error** : Initially, the call to viewPortfolio and the assertion of iterator.hasNext() caused a failure, indicating that the iterator had nothing to iterate over. This was most likely because I had not wrapped the calls in a Transaction, which would make sure that the Citation was added to the database before the call to viewPortfolio. Once this was wrapped inside a Transaction the issue was fixed and the integration test passed.

-**Input** : Citation with Strings description and resource to be added to a User portfolio
-**Expected output**: Citation iterator containing Citation with those same Strings description and resource being the same.

check_addAndDeletePortfolio()

-**Severity**: 3
-Units integrated for test: Email, User, Citation

-A new Email and User is created to add Citations to just as described in the integration test above.
-This time I make an array of Citations and call addToPortfolio passing each Citation to be added. After calling viewPortfolio to return an iterator, I iterate over each Citation and call removeFromPortfolio to delete that Citation.

Error: a transaction rollback exposed an error in Entity.java (which Citation extends) in the delete() function, a line of code verifying the success of a transaction was forgotten and added to fixed the bug.

Input: Four citations to be added for a user's portfolio
Expected output: an empty portfolio

Error: calling removeFromPortfolio() with an invalid citation ID caused a break, this was fixed by placing the code in a try-catch(IllegalArgumentException e)

-**Regression testing** : During the build for our application all unit tests are run. When I did a build before pushing my code I passed all the unit tests, indicating that my fix did not introduce a new bug or regress to bugs we had already worked through.

Integration tests in test/integrate/Registration.java

check_multiple_signup()

-**Severity**: 2

-**Units integrated for test** : Email, ClaimToken

-A new Email object is created by requesting a Token.
-This request is repeated for the same e-mail.

-**Error** : User would be unable to call signup on an email in an attempt to re-retrieve a Token for an unclaimed Email. Fixed by checking if Email had a User assigned to it, not just it existed.

-**Input** : Duplicate email address string signup() calls.
-**Expected output**: The same Token is given each time because the user has not claimed the email address using the token.

check_register()

-**Severity**: 1

-**Units integrated for test** : Email, ClaimToken, User

- A new Email is created using email address string.
- New User is created/retrieved and assigns password to string inputted.
- Created email is registered to that User.

-**Error** : The password and it's password verification parameters were not checked for equality before creating User-Email relationship. Fixed by adding check for equality before attempting transaction.

-**Severity**: 3

-**Input** : Password and password verify strings not identical
-**Expected output**: Transaction is dropped and user alerted of non-equal passwords.

-**Error** : ClaimToken was not cleared from Email object after assigning Email relationship to

User. Fixed by calling clearClaimToken() to an Email that had a User assigned to it.

-**Input** : Email assigned to User. ClaimToken then attempted to be retrieved for claimed Email.
-**Expected output**: Returns that ClaimToken does not exist and that User assignment does and command line output describing that the email has already been claimed.

Integration tests in test/integrate/ViewSubjectiveNetwork.java

check_Simple1()

-**Severity**: 2

-**Units integrated for test** : Email, User, TrustEdge, Subject

-User is created and trustedge is added to link the logged in user
-Prints the trust edge, and the network as view by the logged in user

-**Error** : User would not get to view the network because the trustedges were improperly handled.

-**Input** : The test needs the another User object along with an Email object and a TrustEdge connecting the two, along with a subject that matches the created trustedge.
-**Expected output**: The email of the created user object followed by a newline character

check_Simple2()

-**Severity**: 2

-**Units integrated for test** : Email, User, TrustEdge, Subject

-Two user objects are created along with the simulated logged in user
-Each has an email added to identify them by
-Trust edges make a chain from the logged in user to the fake user to follow the patter: User->fakeUserA->fakeUserB

-**Error** : Only the network is printed up to User A or nothing is printed at all.

-**Input** : The test needs two user objects, two trustedges and two email objects to simulate the test scenario. The edges should be connected so that the pattern of the network is: User->fakeUserA->fakeUserB

-**Expected output**: The email of the two fake users, one on each line

check_subject1()

-**Severity**: 2

-**Units integrated for test** : Email, User, TrustEdge, Subject

-The point of this test is to make sure that the traversal only checks Users that are trusted over a general subject. This is done by making a User object and creating a Trustedge that connects the logged in User to the created User over a subject. The method is called with a subject that isn't the subject store in the Trustedge.

-**Error** : The method traverses over users that don't have authority on a subject and they get false credit.

-**Input** : We expect to have a logged in User along with a created User object, an Email for the created User, and a Trustedge with a subject that will not be queried for.
-**Expected output**: There should be no output if the test is successful.

check_subject3()

-**Severity**: 2

-**Units integrated for test** : Email, User, TrustEdge, Subject

-There needs to be three User objects, three Email objects, and three Trustedges. The network should be setup in the following structure:
USER-testing->UserA
UserA-coding->UserB
UserB-testing->UserD
To test if a traversal will go when it is supposed to and stop when it has to cross an inappropriate subject.

-**Error** : The whole network is traversed and printed

-**Input** : There needs to be three User objects, three Email objects, and three Trustedges.
-**Expected output**: The expected output should simply be UserA's Email.

Integration tests in test/integrate/ViewTrustNetwork.java

check_Simple1()

-**Severity**: 2

-**Units integrated for test** : Email, User, TrustEdge, Subject

-Two user objects are created along a Trustedge that contains a subject
-The Trustedge make a chain to follow the pattern : UserA->UserB

-**Error** : Nothing is printed and program crashes

-**Input** : The test needs two User objects, that each have an Email object added to them. There needs to be a TrustEdge connecting the two Users that contains a Subject.

-**Expected output**: The output should be the subject “testing” followed by the trusted User’s Email address

check_Simple4()

-**Severity**: 2

-**Units integrated for test** : Email, User, TrustEdge, Subject

-Two user objects are created along with with a Trustedge connecting them and containing a Subject. The network is attempted to be view from the trusted User.

-**Error** : Nothing is printed because the trusted User doesn’t have any outgoing edges.

-**Input** : The test needs two user objects, a Trustedge and two Email objects. The setup of the network should look like: UserA->UserB and the network should be view from UserB’s perspective
-**Expected output**: There should be no output because no Users are trusted by UserB

Integration tests in test/integrate/Trust.java

check_can_trust_user()

-**Severity**: 1

-**Units integrated for test** : User, Email, TrustEdge, Citation
-Very important part of application, the piece that allows users to actually trust one another in certain subjects.
-New test users are created and filled with dummy dat

-A new TrustEdge object is created

-**Input** : Two User objects to connect, a Subject that connects them and a Citation to explain why they are trusted.
-**Expected output**: A new database object with a Guid that can be referenced. This class allows ViewTrustNetwork or ViewSubjectiveNetwork to actually display results

-Two new Emails are created by passing unique String to app.Email (**input for Email unit**), which should return an Email object (**output of Email unit**). app.User() is called with no parameter to return a new user, which has the password and email set by using the setPassword and addEmail methods of the User class (Strings are passed as the email and password). This is our standard way of creating dummy users to test with.
-Now there should be two valid Users (**output of User unit**) for which to use as **input** for the TrustEdge class.
- A Subject with **input** String input is created using it's constructor which **outputs** a new Subject
- TrustEdge's constructor requires **input** of two Users and a Subject and **outputs** a new TrustEdge written to the database.
- A new Citation is filled with dummy String **input** for it's fields: description, reason and **outputs** a Citation object
- TrustEdge.addCitation takes the Citation object as **input** which is written to the database
- TrustEdge.reasons() is called with no input, it will **output** an Iterable<Citation> which is asserted to have exactly one Citation and that the Citation.toString() **output** matches the dummy citations **input**.

-**Error** : None were found, the test worked but didn't discover any defects in the TrustEdge class

check_can_trust_email()

-**Severity**: 2

-Identical in all aspects to check_can_trust_user() except an Email is passed to the TrustEdge as **input** instead of the second User. This is to simulate the ability for a User to trust someone who is not registered with the application. Both User and Email extend class Entity and are interchangeable as **input** to TrustEdge's constructor.

check_can_untrust_email()

-**Severity**: 2

-**Units integrated for test** : User, Email, TrustEdge, Citation
-Same setup as check-can-trust-user but delete's TrustEdge from database and verifies edge is gone.

-**Input** : Functioning TrustEdge to delete()
-**Expected output**: TrustEdge with Guid passed in is no longer referenceable and doesn't appear when calling ViewTrustNetwork or ViewSubjectiveNetwork.

- Follows check-can-trust-user exactly first
- new TrustEdge from check-can-trust-user has delete() method called with **no input**, it usees it's internal Guid.
- TrustEdge is removed from database
- Assert that searching for it's Guid yields zero results