KloudBook Low Level Document

Version: 1.0

Project Team Pippin

Team Leader	David Soohoo
Secretary	Dave Burgess
Customer Liaison	Caleb Larson
Integration Liaison	Dylan Terry
Coders	Daniel Choi
	Robert Hromada
Project Manager	Timm Allman
Client	Mike Mullen
	Dave Mullen

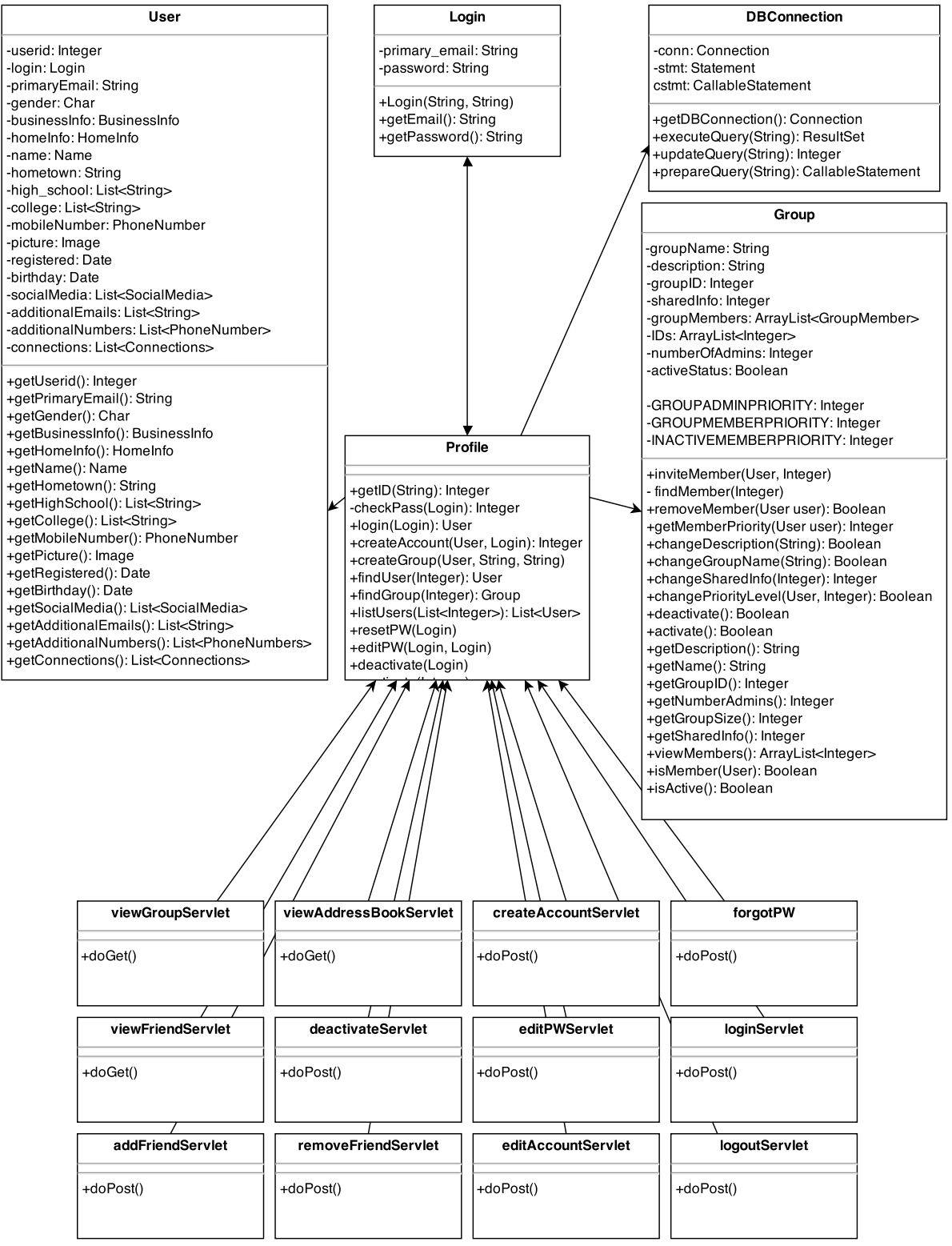
1. Introduction

The purpose of this document is to detail the low-level design and unit testing of the backend. This document provides diagrams that outline essential classes and their functions, and how they relate to one another. Included in this document are the unit tests for testing the functionality of the database. We are working with Cloud 7 (the web application and iOS team) and Team Kilobyte (the Android team) to form a completely integrated social network for KloudBook.

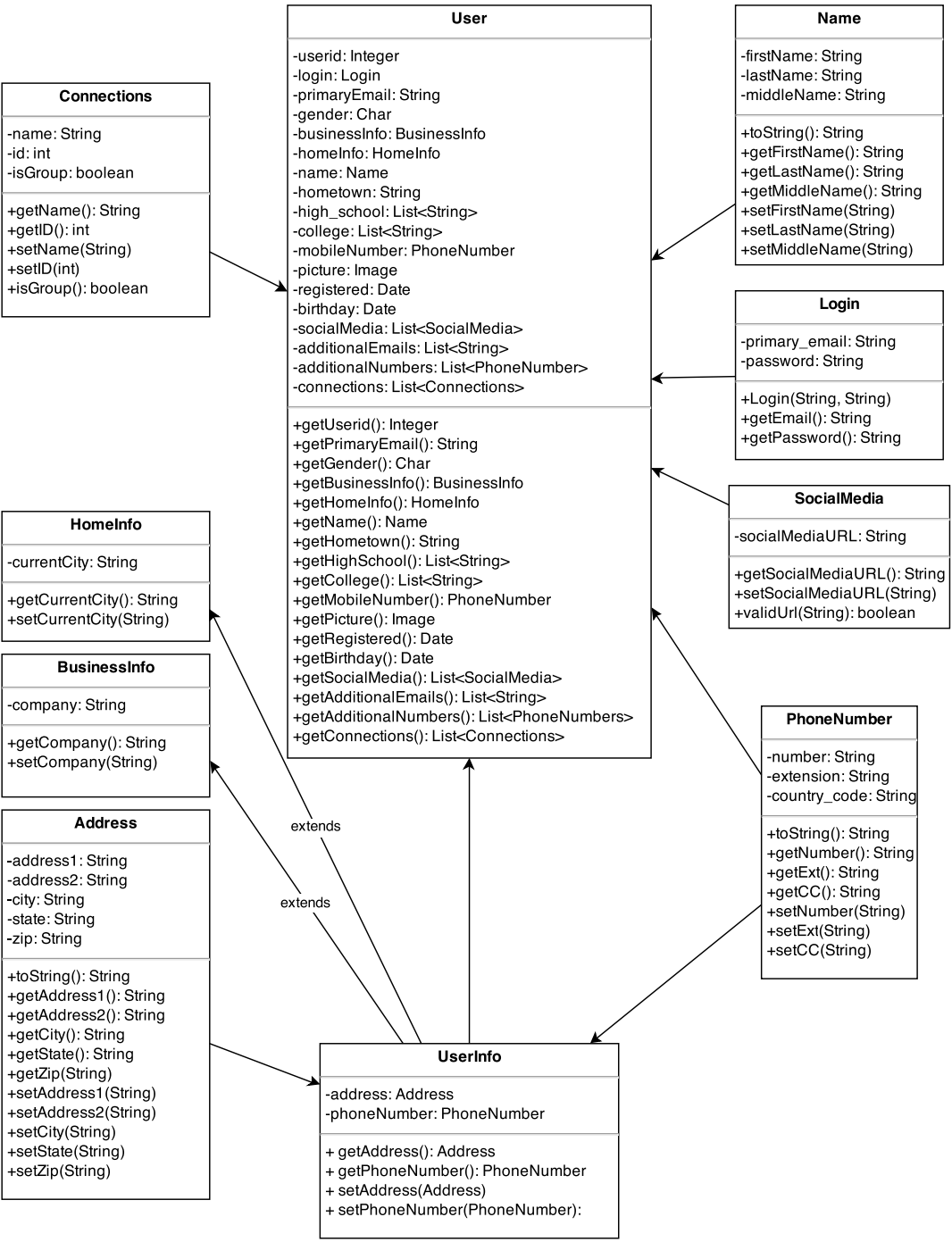
2. Low Level Class Diagrams

- **2.1. Profile –** Attached
- **2.2. User –** Attached
- **2.3. Search –** Attached
- **2.4. Group –** Attached

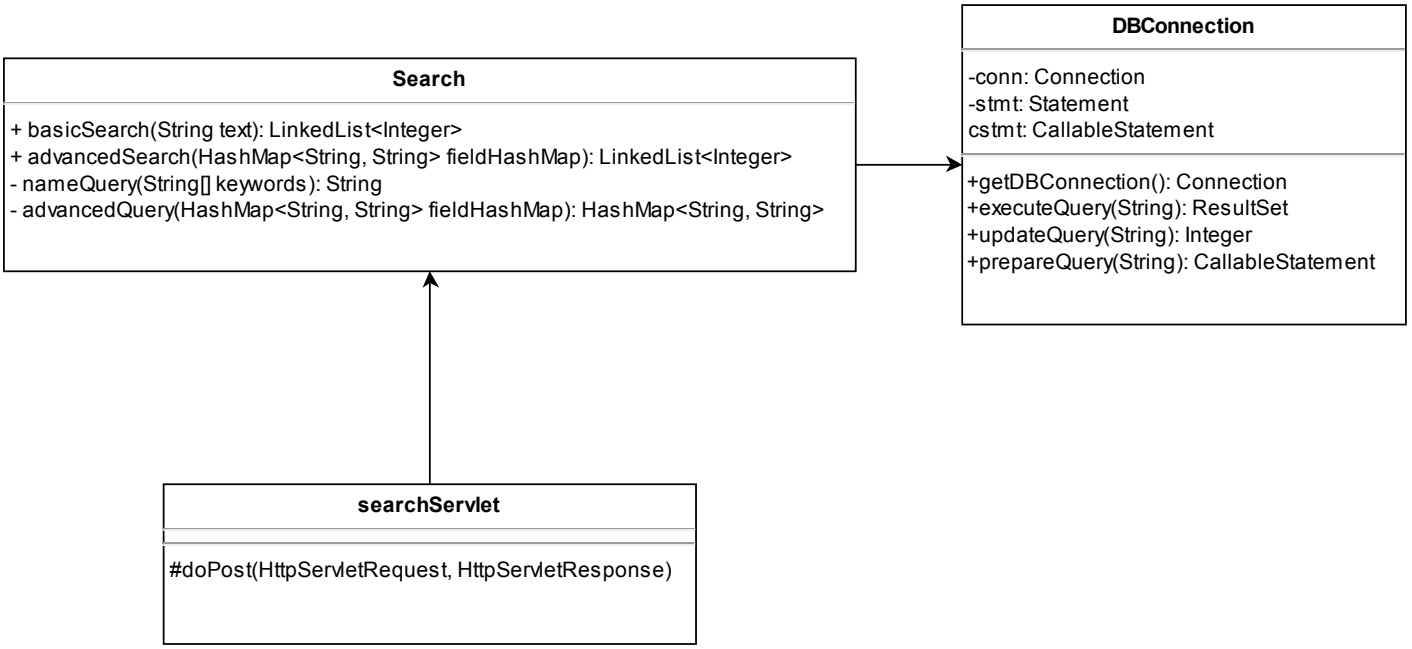
Profile Class Diagram



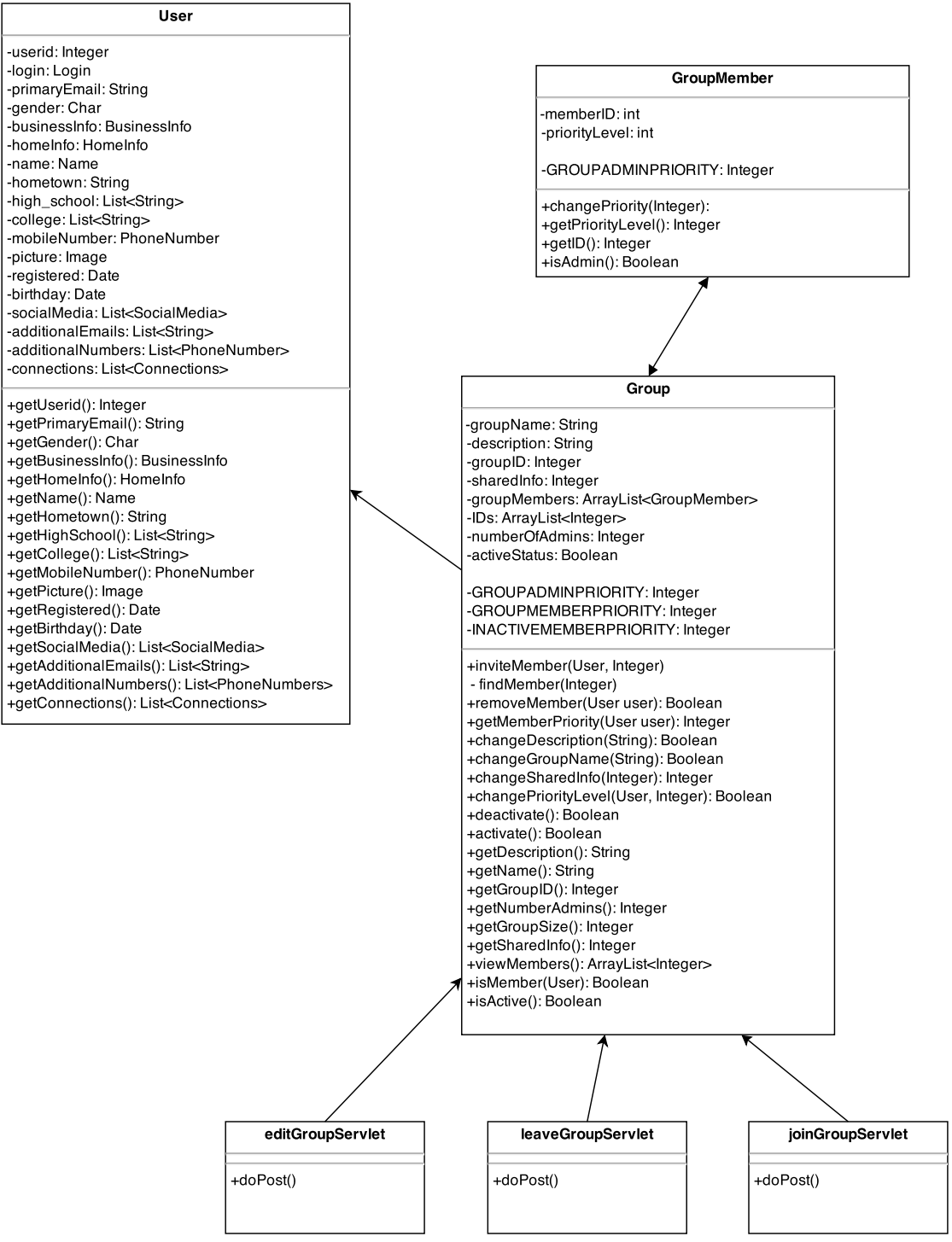
User Class Diagram



Search Class Diagram



Group Class Diagram



3. JUnit Tests

3.1. Profile

3.1.1. getIDofBadEmail
Should return -1 at an incorrect email.

3.1.2. getIDofNonExistEmail
Should return -1 at a non-existent email in the database.

3.1.3. getIDofExistEmail
Should return the ID of a user with the existent email.

3.1.4. loginBadEmail
Should fail at a bad email.

3.1.5. loginGoodEmailBadPass
Should fail at an email that exists and the incorrect matching password.

3.1.6. loginGoodEmailCorrectPass
Should return User object at an existing email and its matching password.

3.1.7. createAccountExistEmail
Should fail if the email already exists in the database.

3.1.8. createAccountMissingEmail
Should fail if the email is missing.

3.1.9. createAccountMissingPassword
Should fail if password is missing.

3.1.10. createAccountSuccess
Should pass if primary email is unique and has a password.

3.1.11. editAccountNewEmail
Should update the email address in the database for a loaded user and check if update occurred properly.

3.1.12. editAccountNewEmailAlreadyExists
Should fail when updating the email to one that already exists in the database.

3.1.13. editAccountPassword
Should update the account password, login attempted with new password to confirm.

3.1.14. findUserDNE
Should return null User object due to user not existing.

3.1.15. findUserPublicID
Should return null User object since the public user cannot be accessed.

3.1.16. findUserGoodID
Should return full User object.

3.1.17. listUsersListEmpty
Should return an empty List as the List passed is empty.

3.1.18. listUsers
Should return a List of User objects in accordance with the List passed of those user IDs.

3.1.19. editPWBadEmailNoPass
Should return false as email did not match a user.

3.1.20. editPWGoodEmailNoPass
Should return false as no new password given.

3.1.21. editPWGoodEmailNewPass
Should return true as email matches and new password updated.
3.1.22. deactivateBadEmailBadPass
Should return false as email and password do not match.
3.1.23. deactivateGoodEmailBadPass
Should return false as password does not match email.
3.1.24. deactivateBadEmailGoodPass
Should return false as email does not match a user.
3.1.25. deactivateGoodEmailGoodPass
Should return true as email and password match same account and account properly deactivated.
3.1.26. reactivateBadID
Should return false as a non-existent ID cannot be activated.
3.1.27. reactivateGoodID
Should return true as an existing ID can be reactivated.

3.2. Group

3.2.1. addMemberTest()
Should return true after adding a new user to a group.
3.2.2. getMemberPriorityTest()
Should return true by asserting the correct priority level of a user.
3.2.3. removeMemberTest()
Should return true after removing a member from a group.

3.3. Search

This test class inserts a dummyinsert.sql file as precondition material for jUnit tests to make queries and assertions about.

3.3.1. BasicSearchTest()
Should return true by finding the correct user.
3.3.2. BasicSearchTestNull()
Should return true by searching and returning nothing.
3.3.3. AdvancedSearchName()
Should return true after conducting an advanced search on a user.
3.3.4. AdvancedSearchTestNull()
Should return true after conducting an advanced search and returning nothing.
3.3.5. AdvancedSearchHighSchool()
Should return true after conducting an advanced search on education information and returning the correct users.
3.3.6. AdvancedSearchEmail()
Should return true after conducting an advanced search on email information and returning the correct user.

3.4. DBConnection

3.4.1. testExecuteMisspelledQuery()
Exception should be thrown at a misspelled query.

3.4.2. testExecuteSelect()
Should pass at correctly spelled SELECT query.

3.4.3. testExecuteWrongQuery()
Exception should be thrown at a query not allowed through executeQuery: UPDATE, INSERT, and DELETE queries.

3.4.4. testUpdateMisspelledQuery()
Exception should be thrown at a misspelled query.

3.4.5. testUpdatePKErrorQuery()
Exception should be thrown for attempting to insert a primary key.

3.4.6. testUpdateDelete1()
Should pass at correctly formatted DELETE query assuming data from @Before is loaded properly.

3.4.7. testUpdateInsert1()
Should pass at correctly formatted INSERT query assuming data from @Before is loaded properly.

3.4.8. testUpdateDelete2()
Should pass at correctly formatted DELETE query with two consecutive deletions assuming data from @Before is loaded properly.

3.4.9. testUpdateInsert2()
Should pass at correctly formatted INSERT query with two consecutive additions assuming data from @Before is loaded properly.

3.4.10. testUpdateQuery()
Should pass at correctly formated UPDATE query assuming data from @Before is loaded properly.

3.4.11. testPrepareQuery()
Should pass at correctly formated procedure call.

3.4.12. testClose()
Should not fail unless database connection is not active.