“Bonjour”
Integration & Regression Test Log

Last Revised: February 28, 2014

Team 7 Members:

Xiangyu Bu

Rishabh Mittal

Yik Fei Wong

Yudong Yang

Introduction

This is the incremental, integration, and regression test log of project "Bonjour" for sprint 2 phase. The class definitions and descriptions are updated according to new implementations and plans.

Classification of Components

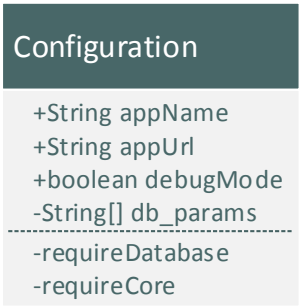
From the broadest view, the project consists of two major components: *Server side* and *Android app client*. The next two sections will list the components of the server and the client, respectively, and a third section provides more diagrams illustrating their interactions.

Server

Server is the part that handles client requests and perform actions accordingly. All components of server are tested according to a ***bottom-up*** strategy because the basic components to model objects and protocols are developed first and the connector and interactor components are developed afterwards. We wrote unit test programs every time a basic component was done. As a result, the bottom-up testing strategy fits best in this part.

Configuration Class

The Configuration class is the component that



- *saves all the constants* used by the server side, such as the name and URL of the server program, and the credentials to connect to the database service
- does some *preliminary check* against the operating system environment (e.g., the PHP version) to ensure compatibility
- *loads and boots up all the necessary classes and libraries* required by any arbitrary operation (e.g., the Core class, the Database class, and the password compatibility library if the PHP version is less than 5.5.0).

Dependency:

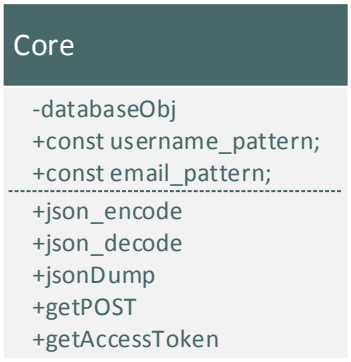
- It does not depend on or call other classes, but is a pre-requisite of Core class, Database class, and ActionRouter class.

Input / Output:

- none.

Core Class

The Core class take a role as a



- *Request parser.* It parses all POST requests from the client, and handles all the queries needed by other components.
- *Response formatter.* To fulfill this work, the Core gets the "error" or "finish" signal from other working components, and sends error message, or the requested data by clients, accordingly, and then clean the memory. Besides, it does the job of transforming data into json type, which is required by the client.
- *Crypto center.* It generates the tokens used to verify requests, and provides all regular expressions for other components to check variables against.

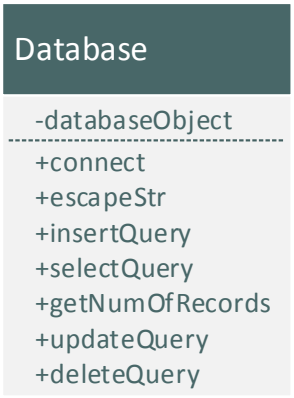
Dependency:

- It needs constants from Configuration class at initialization.
- ActionRouter class and User class need to fetch data from Core class, and depends on Core to handle responses and errors.

Input / Output:

- To parse client requests, it pre-processes the HTTP header (input) from the client, waiting for other components to fetch the fields needed. When a component (e.g., ActionRouter) queries for a field, say, "action" from POST data, it returns the string associated with the key "action" (output).
- When the user request is successfully performed, Core takes an array as input, parses it to json format, and sends the data back to the client (output).
- In order to generate a token requested by the User class, Core takes the user information and server status as input, and returns a long, encrypted string as output.
- ***(Sprint 2)*** Core takes a string from User class as input, and returns a boolean value marking whether it is a valid username, valid password, or valid email address, as per the request.

- **(Sprint 2)** When the action user requests encounters an error, Core will take the Exception raised by the controller class involved (E.g., User class) as input, and parse it to the pre-defined format as output, and return this to the client.
- **(Sprint 2)** Core can take an int value length as argument and return a random string. This is used for generating random passsowrd.



Database Class

The Database class is the PHP implementation of MySQL database controller. It creates a PHP MySQLi object, connects it to the MySQL server, and handles database queries passed to it by other components. The major functionalities are:

- Connecting to a database server and authenticating
- Generating SQL escape string against some kinds of SQL injection attacks
- Performing SELECT queries; getting number of rows in the query
- Performing INSERT / UPDATE / DELETE queries

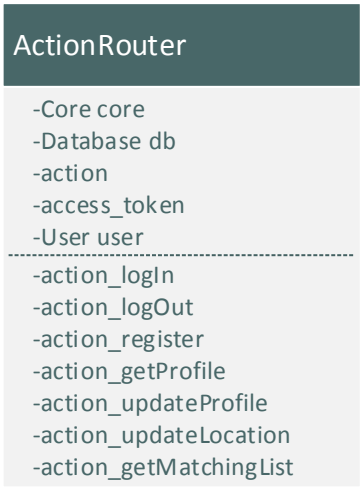
Dependency:

- As an independent basic class, it does not call other components
- Configuration class loads it to memory
- User class and ActionRouter class relies on Database object to perform database queries

Input / Output:

- At instantiation, the db_params array defined in the Configuration class (input) will be fed to the constructor of Database class
- User Class will send SQL queries to the class to execute (input), and fetch the query result, including the number of rows in the last query, through the API functions (output)
- **(Sprint 2)** When an error occurs performing the designated action, it will print the error in json format and terminate the session.

ActionRouter Class



ActionRouter is a component which executes a pre-defined path according to the request received. The functionalities are:

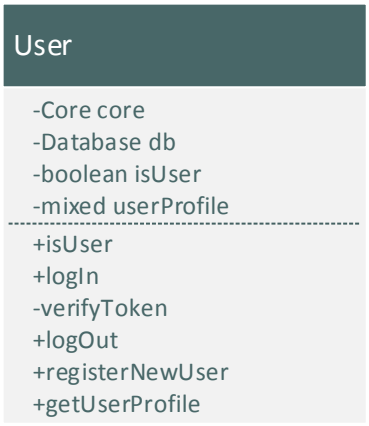
- Get the desired action from the HTTP request
- Handle user log in requests, verify username and password, update token
- Handle user registration requests, verify the validity of each field, and add the record to database if all correct
- Handle profile fetching requests, profile updating requests
- ***(Sprint 2)*** Handle file uploading requests
- Handle location updating requests (to be done)
- Handle matching requests (to be done)
- Handle any exceptions that can be raised, solve them or pass the error signal to Core object as needed

Dependency

- ActionRouter class is the expected *entry point* of the HTTP request
- ActionRouter class will load Configuration class if it is not loaded yet
- ActionRouter class will load and instantiate Core class, which further instantiates Database class, if it is not loaded or instantiated yet, to start processing the HTTP request
- ActionRouter class will instantiate a User object, and interact with it to perform user identity-related requests
- ActionRouter class relies on Database object in the Core object to perform database operations and fetch data from the database
- ActionRouter class passes the result of the action to Core class to be formatter to json format and sent to the client
- ActionRouter class handles exceptions (LoginException, RegisterException) raised by other classes like the User Class (new)
- ActionRouter class sends error signals to Core class
- No classes call or instantiate ActionRouter class

Input / Output

- ActionRouter class takes the HTTP Request data parsed by Core as input, and execute the pre-defined route for a specific request. For example,
    - For user registration action, it takes the username, password, email, and confirmation email fields in POST as input, verify the information, and pass success signal to Core if registration succeeds, or pass error signal to Core if there is any error (output).
    - For user log in action, it takes the username and password fields from HTTP POST as input, and fetches the password hash of the specific user from the database, and check if they can be verified. It returns success signal to Core if passing, or error signal if fails (output).
- When an error occurs, ActionRouter will accept the Exception as input, parse it, and send the corresponding error message to Core to format to json output.



User Class

User class is the model for a user, including guests, who is requesting to perform actions. It also handles the validity check of login and registration operations.

Dependency

- It calls Core object to get the username and email patterns, and to get the token for the specific action of the user.
- It calls Database object to perform SQL queries.
- It raises LoginException object or RegisterException object when there is an error logging the user in or registering the user, respectively.
- It is called by ActionRouter class to perform login, registration, and profile updating operations.
- ***(Updated)*** It interacts with UserProfile object to handle the access and update of user profiles.

Input / Output

- User class takes the username and password as input to handle login operation, and if succeeds, it returns an access token as output, otherwise, raises a LoginException including the error number and message to ActionRouter to handle.
- User class takes the username, password, retype password, and email as input to handle registration operation. If the action is successful, return an access token to notify the client to log in (output); otherwise it raises a RegisterException including the error number and message to ActionRouter to handle.
- ***(Updated)*** User class takes the username as input, fetches the profile of that user, parses the data to a UserProfile object, and returns it to the caller.
- ***(Updated)*** User class takes the username and a UserProfile object as input parameters, generates a SQL query for the database to update the user profile, and ask Database object to execute it.

LoginException Class

LoginException class is raised when an error occurs logging user in.

Dependency

- It is raised by User class
- It is catched and handled by ActionRouter class
- ***(Updated)*** It is parsed to final output by Core class
- It does not call other classes

Input / Output

- It takes the error number and error message from User object as input, and return an Exception that should be catched as output.

RegisterException Class

RegisterException class is raised when an error occurs registering a user.

Dependency

- It is raised by User class
- It is catched and handled by ActionRouter class
- ***(Updated)*** It is parsed to final output by Core class
- It does not call other classes

Input / Output

- It takes the error number and error message from User object as input, and returns an Exception that should be catched as output.

ResetPassException Class (new)

This exception object is raised when an error occurs for password resetting operation.

Dependency

- It is raised by User class
- It is catched and handled by ActionRouter class
- It is parsed to final output by Core class
- It does not call other classes

Input / Output

- It takes the error message as input, and returns the Exception as output.

EmptyFieldException Class (new)

This exception has not been used in the server part yet.

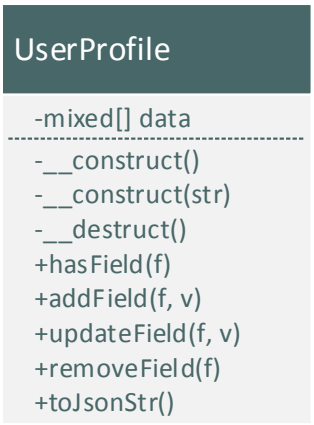
UserProfileException Class (new)

This exception is raised for errors occurred when accessing or updating user profiles.

Its ***dependency*** and ***input / output*** relationships are the same as the `ResetPassException`.

UserProfile Class

UserProfile class aims to model the profile of a user, to facilitate the construction and edition of the user profile, and to unify the two forms of user profile representation – json (client format) and associative array (server format). If needed the class will be added some verification steps in the future.



Dependency

- User class returns a UserProfile object to ActionRouter when the latter needs to get the profile of a specific user.
- ActionRouter parses the json data to a UserProfile object and passes it to User class to perform updating.
- UserProfile class is independent of other classes.

Input / Output

- The constructor takes json text as input to build the associative array that represents the user profile.
- When adding / updating a profile field, it takes the field name and value as parameters and update the data array accordingly.
- When checking if a field exists or removing a field, it takes the field name string as parameter and perform the operation accordingly.
- ToJsonStr method will return the json representation of the associative array as output.

Password Compatibility Library

Password compatibility library is an official library provided by PHP Official Support to add the password utility functions newly introduced in PHP 5.5 to lower versions of PHP. It has been thoroughly tested officially and there is no need to unit test the library again.

Dependency

- It does not depend on other classes to perform operations
- User class depends on it to generate hash for the password and to verify the password hash with the user password
- Core class depends on it to generate the access token

Input / Output

- The official documentation for the functions in this library is at http://www.php.net/manual/en/book.password.php

FileUploadHandler Class (new)

This class handles all the file upload operations. It can get the file path, file size, extension name, and other properties of files stored in $_FILES array of PHP request, do sanity check and store them to a specific folder.

Dependency

- So far it is an independent library.
- ActionRouter calls this library to parse file saving operations.

Input / Output

- It takes the $_FILES object as input, and stores the valid files to the disk.

UnitTest Suite

UnitTest is a series of test drivers used to test a class previously listed when it was first built. Each unit test file tests a specific class, performs the designed test cases and generates an output report. This facilitates regression test. When new features are added to the class, new test cases can be added to the unittest file to perform incremental test.

Dependency

- Each unit test file depends on a particular class listed before, and implicitly depends on the classes that the tested class depends on.
- No functional classes depend on it.

Input / Output

- The input of each testcase is pre-written in the unittest file, and the output is a test report telling which test fails and what are the expected and actual outputs.

Android App

The client provides internal classes to communicate with the server. The signup, login, profile and other activities are provided for users to create their own account, log in and view their profiles. Considering the fact that the client interacts directly with the user, it would be natural to design the UI first. Therefore, we followed a ***top-down*** strategy to develop and test this part – first finish the activity UI, use stubs to give confirmation of the reach of an action, and finish the actions from top to bottom.

SignupActivity Class

The signupActivity class that contains three signup fragment components (Basic, upload, detail). It lets users enter their username, password and other information details. It performs some checking functions before submitting the information to the server.

Dependency

- APIHandler class to check the network and update the register info

Input / Output

- Username and password, uploaded user image, detailed information
- Register Success and failure message

LoginActivity Class

The LoginActivity class is used for displaying two text fields for a user to enter her/his username and password, necessary checking functions are being used in the activity to check the validity of the format of the username and password, also a signup button button is provided for a first time user to register her/his account.

Dependency

- APIHandler class and SQLHandler class are needed for retrieving information from the server and local database

Input/Output

- Username and password
- login success or failure message

UserProfileActivity Class

The UserProfileActivity class shows the profile of the current user by obtaining information from the APIHandler

Dependency

- APIHandler class and SQLHandler class are needed for retrieving information from the server and local database

Input/Output

- Edit user profile
- Current user profile and updated success and failure message

ForgetPasswordActivity Class

This activity handles client password retrieval request.

Dependency

- APIHandler class and SQLHandler class are needed for retrieving information from the server and local database

Input/Output

- New password
- Password updated success or failure message

APIHandler Class

The APIHandler class implements the function of network connecting and protocol functions that communicate with the server.

Main functionality:

- Check the network availability of the mobile phone
- Check the validity of data that send to the server
- Encode data and send the correct format of data
- **(Updated)** Decode JSON response data and acquire correct information from JSON responses
- Fetch user information from the server
- Update and obtain cached user profile and friends list from local SQLite database

Dependency:

- It uses SQLiteHandler class to handle internal SQL operation
- Require the network permission from the AndroidManifest.xml

Input/Output

- Encoded data from the server
- Formatted userProfile object and network availability status

- **(Updated)** Decoded server responses and user tokens

SQLiteHandler Class

The SQLiteHandler class is an internal class to handle the local profile caches to allow users to view their profiles and friends list offline. Multiple user accounts are allowed on the phone.

Main functionality:

- Cache user profiles and friends list of multiple users
- Allow to view the user profile and friends list offline
- Perform local SQLite database operations
- **(Updated)** Cache user access token for each user

Dependency:

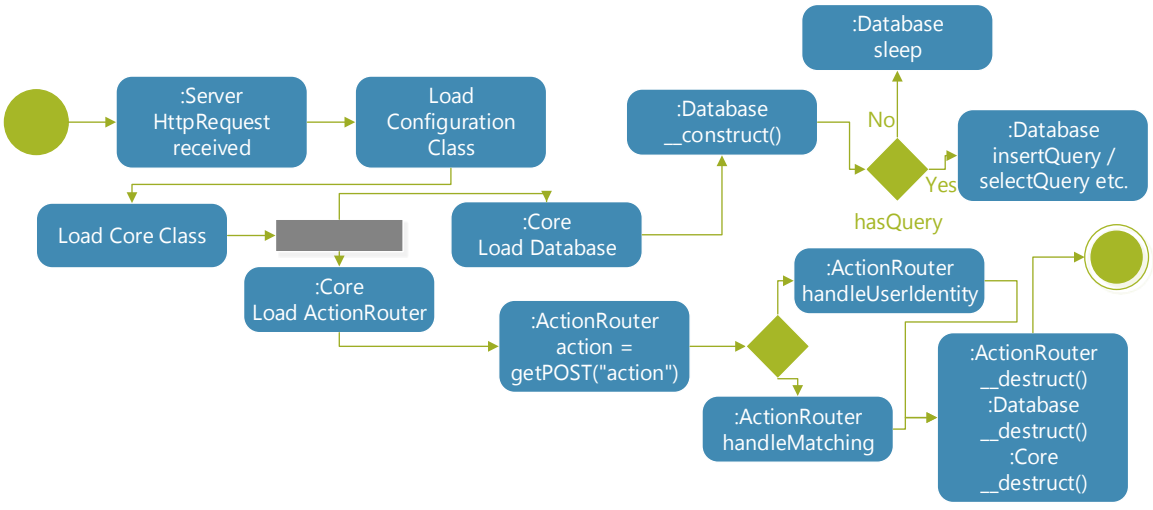
- Require the internal storage permission from the AndroidManifest.xml

Input/Output

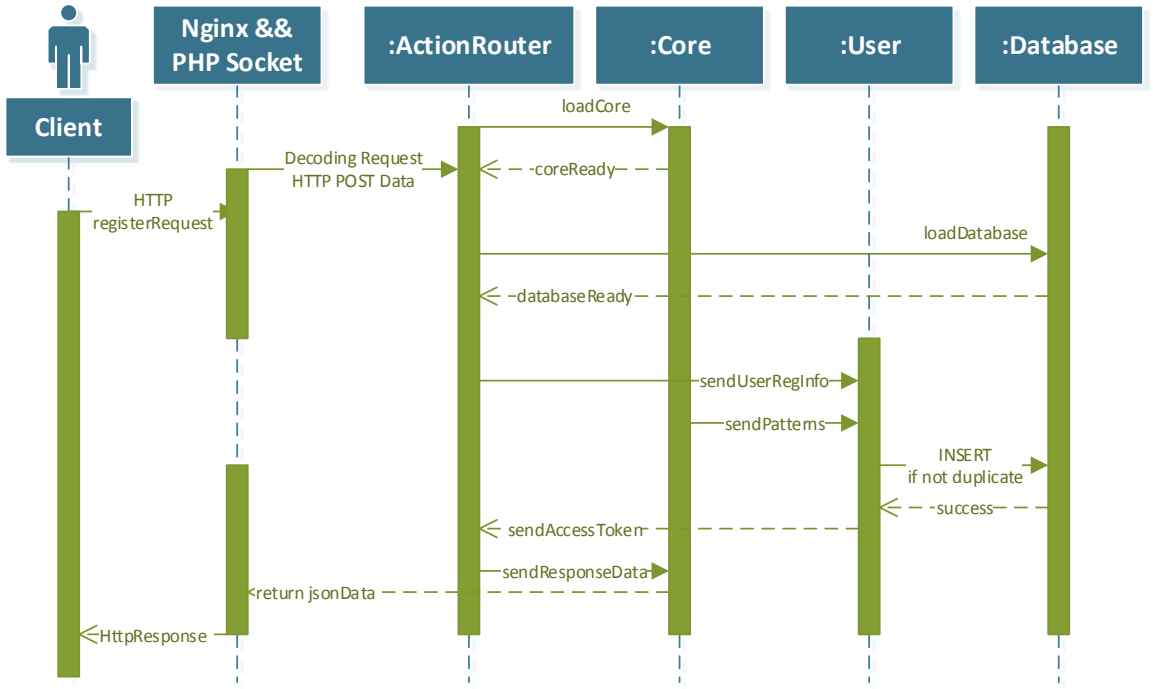
- UserProfile object and friendLists object
- Boolean value of update success or failure in the database

Interactions

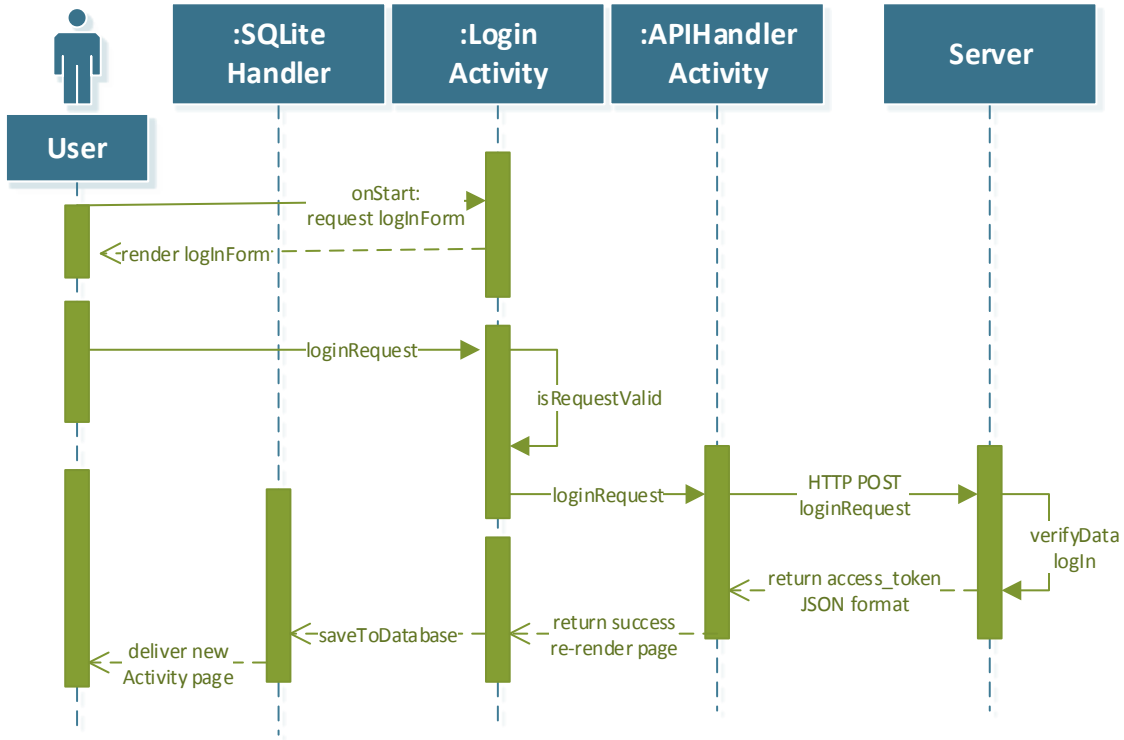
Typical Case: *Interactions inside server at service initialization*



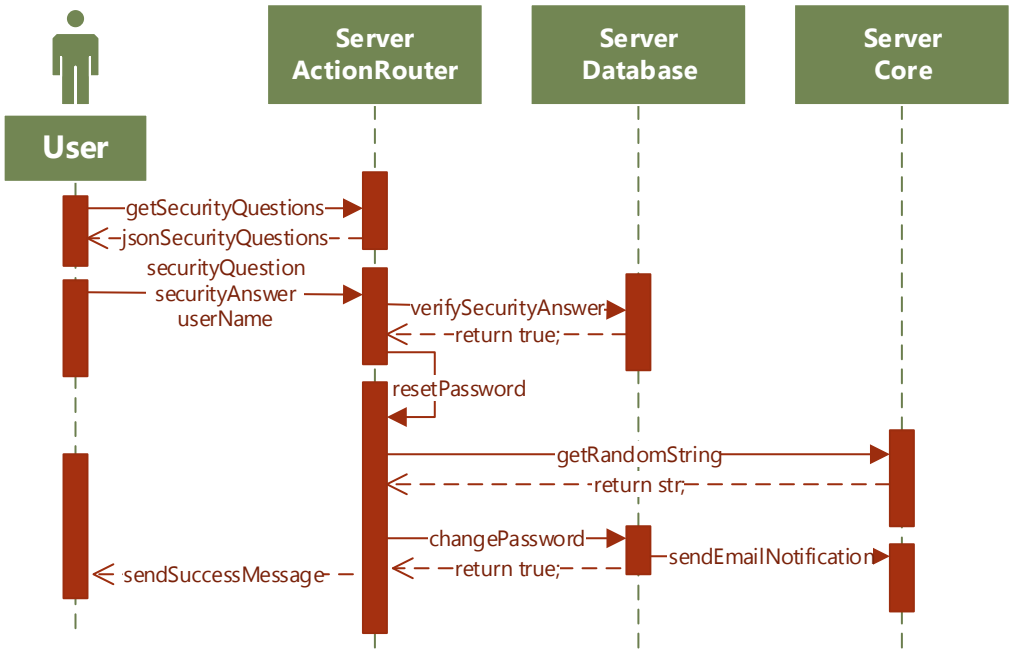
Typical Case: *Interaction among server components when a user registers successfully.*



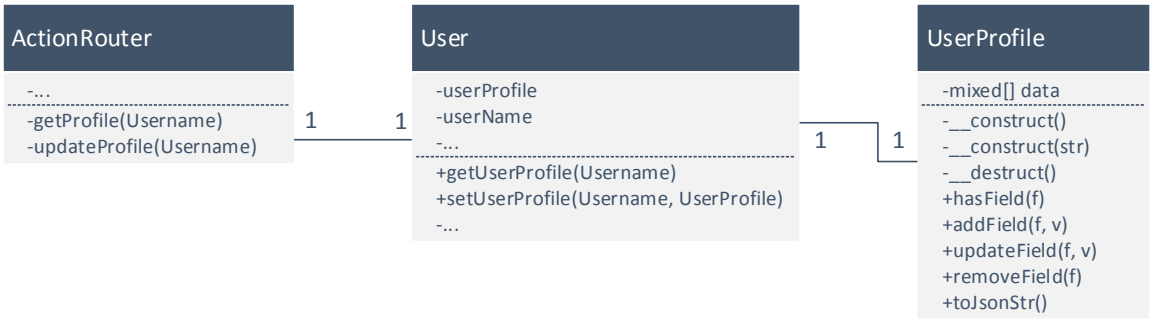
Typical Case: *Client-wise interactions for a valid log-in request*



Typical Case: *Interaction among server components when a user changes password.*



Graph*: The One-to-one relationship between ActionRouter, User and UserProfile classes*



Incremental and Regression Testing

Server Defect Log

For server part, Configuration class, Database class, and Core class were not changed since last test. Only unittest suite was run to make sure no bugs occurred.

Module	Server::Core

Incremental Testing

Defect #	Description	Severity	How to correct
1	The random password generator function always returns a string that each char is unique. This might reduce the level of difficulty of brute-forece attack.	3	Wrote a new hashing algorithm and problem solved.

Regression Testing

Defect #	Description	Severity	How to correct
0	No specific failure found in unit test.		

Module	Server::User and Server::UserProfile

Incremental Testing

Defect #	Description	Severity	How to correct
1	When there is user profile stored, the `UserProfile` class fails to update or remove fields. PHP recorded such error in log.	1	PHP by default parses json as objects. Fixed by parsing json to associative array.
2	Anyone, including an anonymous guest, can fetch the user profile if the username of the target is known.	2	There is no specific design related to this issue. Further discussion is needed.

Regression Testing

Defect #	Description	Severity	How to correct
1	The new, flexible `UserProfile` class does not store `user_name` field, but `User::logOut()` expects	2	When a logged in user sends requests, the username will be stored

	it to do so. This caused failure to logout the user.		as a separate variable for convenience, and `User::logOut()` will use the variable directly.
2	While server has beem implicitly allowing the use of email address as username when performing a SQL query, the sanity check in the functions has forbidden this interchange. But username is defined to be a nickname field.	1	There is no specific fix for this conflict yet. Need to discuss with the client team about revising the API, or to change the primary key to be username.
3	When a guest knows the access token and the corresponding user name from any arbitrary source, he or she can send requests using a faker on behalf of that user. The problem is not critical but needs a better algorithm managing access tokens.	3	Currently the token dies only when the user logs out. Need to talk to the client team regarding the fix.

Notes: #2 and #3 are inspective issue found in regular regression tests. The improvements are likely to involve changes in API and thus be carefully designed before being implemented.

Module	**Server::User::UserExceptions**

No particular issue found for Exception classes.

Module	**Server::User::FileUploadHandler**

Incremental Testing

Defect #	Description	Severity	How to correct
1	When incorporating into the server system, it sometimes fails to recognize asynchronous requests sent by the client.	1	For now most requests from the client are synchronous ones. But this class should be carefully redesigned if we are to introduce asynchronous API.
2	When sending multiple files, FileUploadHandler accepts all of them. This is a failure because only one file (the avatar image) should	3	Fixed by letting the ActionRouter explicitly tell the handler how many files it should

	be uploaded by the client per request.		handle.

Regression Testing

Defect #	Description	Severity	How to correct
0	This module will not affect the functionality of existing modules.		

Module	Server::ActionRouter

Incremental Testing

Defect #	Description	Severity	How to correct
1	Server did not check the validity of the profile field "birthday".	3	Let `ActionRouter` do the sanity check. The date string should be of format `YYYY-mm-dd`.
2	When there are a number of same requests (probably caused by network failure from the clinet), the server may either refuse to work (DDoS defense) or handle all of them one by one. This causes overheads.	3	No ideal fix so far so have to keep the current mechanism, which works, until a new mechanism suitable for mobile networking is designed.

Regression Testing

Defect #	Description	Severity	How to correct
1	The user image uploading action is refused by the server.	2	Re-configured server settings to allow for larger files.

Client Defect Log

Module	Client::UserProfile

Incremental Testing

Defect #	Description	Severity	How to correct
1	When changes are made they are displayed in front of the wrong field.	1	Correct by providing correct id
2	When user changes display picture, the new picture is not	1	Fixed by saving new display in server and

	displayed in the 'View Profile' option		replacing the previous one

Regression Testing

Defect #	Description	Severity	How to correct
1	Some of the text is going outside the boundry of the screen	2	Fixed by changing TextView attribute to wrap_content

Module	Client::SQLHandler

Incremental Testing

Defect #	Description	Severity	How to correct
1	Updated server part needs client part saves the user access token, but no attribute in the user_table can save this token.	1	Add new attribute in the user_table to save the user access token
2	An SQLHandler can be created more than once may cause Database reading problems and synchronization problems when it has more than one thread are reading or writing the database.	2	Use singleton pattern to design the SQLHandler and add a getInstance method to obtain only one SQLHandler object for every activity needs to update local database

Regression Testing

Defect #	Description	Severity	How to correct
1	In the database updating function, it accidently called the getReadableDatabase instead of WritableDatabase.	1	Fix this by calling getWritableDatabase function in the update function
2	An SQLHandler can also be created more than once by calling the constructor of the class.	2	Change the constructor to private to make sure only one object can be created by using getInstance method.
3	Database query returned a cursor object, the object accessed without checking the null value	3	Add checking condition before accessing the cursor object.

Module	Client::APIHandler

Incremental Testing

Defect #	Description	Severity	How to correct
1	Response JSON data is not decoded in the APIHandler and return back directly	1	Parse JSON responses to the data we need
2	JSON Parser is not a standard library in Java, currently no JSON parsing library in the client to be used	1	Add third-party JSON parser library and use it in the APIHandler

Regression Testing

Defect #	Description	Severity	How to correct
1	Response JSON data parsed wrongly and some strings are messed	2	Fix the parse function of JSON in the APIHandler

Module	Client::HomePage

Incremental Testing

Defect #	Description	Severity	How to correct
1	The line dividing TextView and ListView is too thick, is in wrong position and covered the text in TextView under certain resolution (768*1680)	2	Fixed by Modifying layout
2	In ListView layout, the TextView displaying details of hobbies may overlap with the User Image if the list of hobbies are too long.	2	Fixed by Modifying the layout

Regression Testing

Defect #	Description	Severity	How to correct
1	After correcting the position of the line. It overlapped with the top of	2	Fixed by setting "android:layout_below=

	ListView.		"@+id/textViewLine"
2	After correcting the position of the line. It cause the ListView overlapped with the button below.	2	Fixed the layout of ListView

Integration Testing

The integration testing aims to evaluate how well the server and client modules work together in terms of functionality, reliability and performance.

The **network issue** is the first problem we noticed. More specifically,

- The sever sometimes received aborted HTTP requests, or repeated requests which should have been performed once.
- The client sometimes received partial response from the server, which made the json text useless for parsing, and there has not been a mechanism in client to handle this issue.

If time allows, we should transform the API to a way that makes use of push notifications. Otherwise there will be huge overheads caused by network failures.

The **device differences** is another problem we found. Our target device is now Nexus 4, but when another team member emulates in another AVD, the layout slightly changes.

More different resolutions will be tested in the next phase.