“Bonjour”
Incremental & Regression Test Log

Last Revised: February 14, 2014

Team 7 Members:

Xiangyu Bu

Rishabh Mittal

Yik Fei Wong

Yudong Yang

Introduction

This document is to record the components of Bonjour project, and to document the incremental and regression testing performed.

Classification of Components

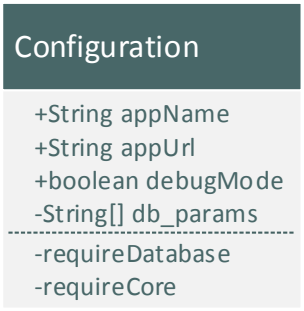
From the broadest view, the project consists of two major components: *Server side* and *Android app client*. The next two sections will list the components of the server and the client, respectively, and a third section provides more diagrams illustrating their interactions.

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:

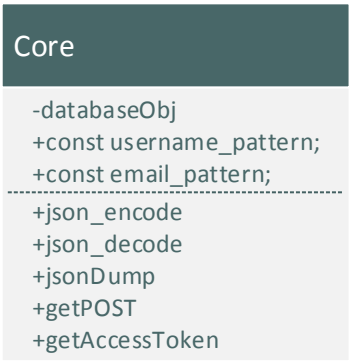
- It does not depend on or call other classes, but is a pre-requisite of Core class, Database class, and ActionRouter class.

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:

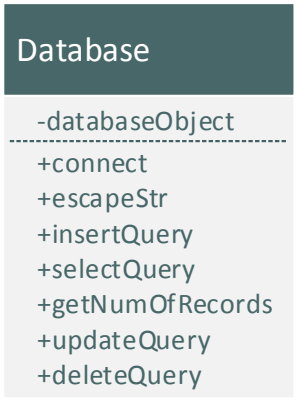
- It needs constants from Configuration class at initialization.
- ActionRouter class and User class need to fetch data from Core class, and depends on Core to handle responses and errors.

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 encounters error, it takes the error number and message from the working component (input), parses them to the pre-defined json format, and sends the json back to the client (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).
- The User class needs to fetch the username and email patterns from Core (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.

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:

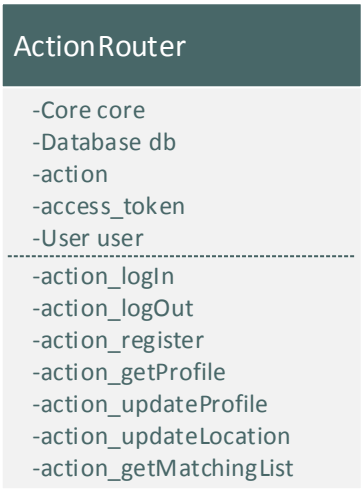
- At instantiation, the db_params array defined in the Configuration class (input) will be fed to the constructor of Database class
- User Class will send SQL queries to the class to execute (input), and fetch the query result, including the number of rows in the last query, through the API functions (output)
- When an error occurs, it raises an error exception (output)

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
- Handle file uploading requests (to be done)
- 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

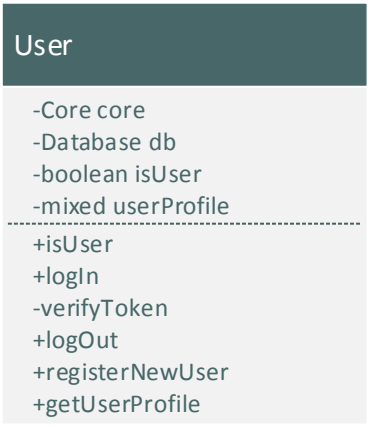
Input / Output

- 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.

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.

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
- 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

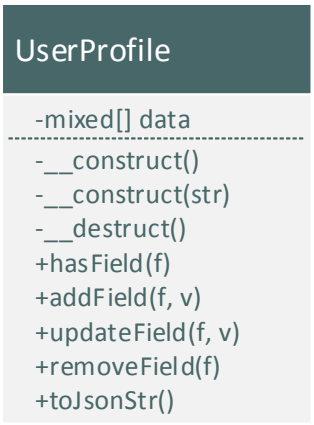
- It is raised by User class
- It is catched and handled by ActionRouter class
- It does not call other classes

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.

UserProfile Class

UserProfile class, introduced in the week of Feb 09, 2014, 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 pass 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

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.

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

ForgetPasswordActivity Class

This activity handles client password retrieval request.

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
- 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

SQLiteHandler Class

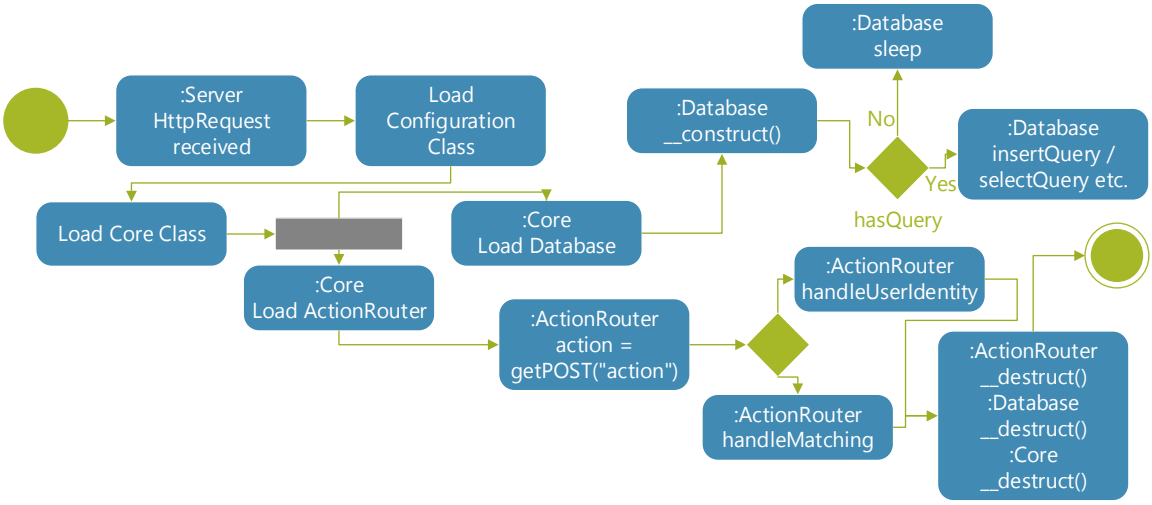
The SQLiteHandler class is an internal class to handle the local profile caches to allow users to view their profiles and friends list offline. Multiple user accounts are allowed on the phone.

Main functionality:

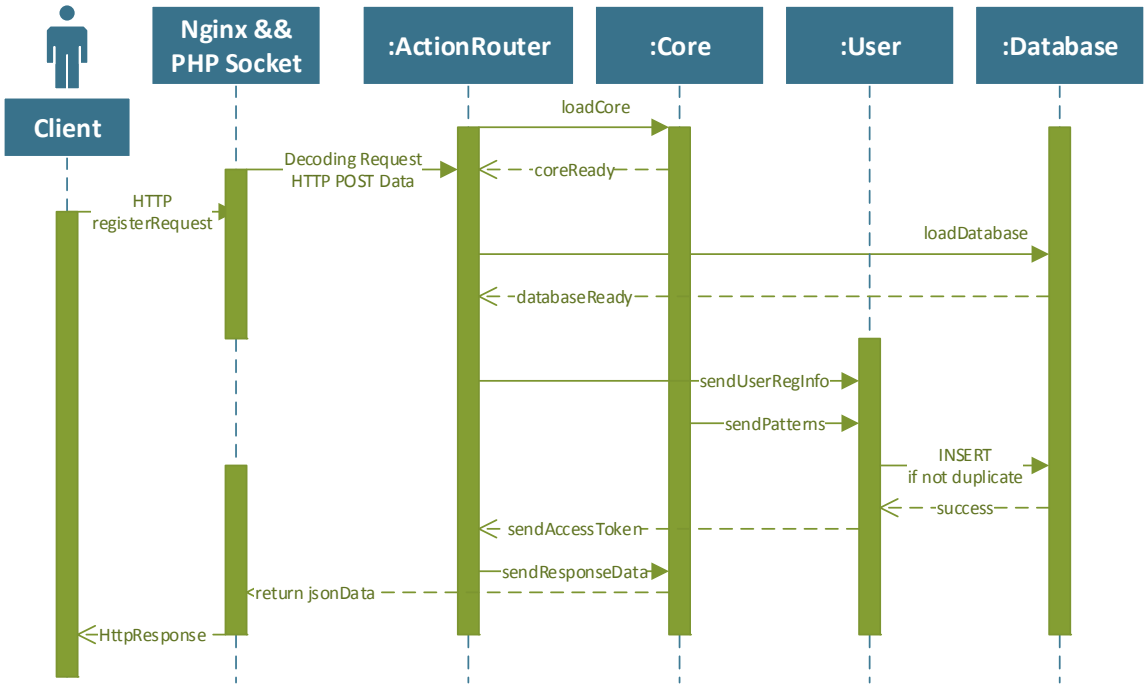
- Cache user profiles and friends list of multiple users
- Allow to view the user profile and friends list offline
- Perform local SQLite database operations

Interactions

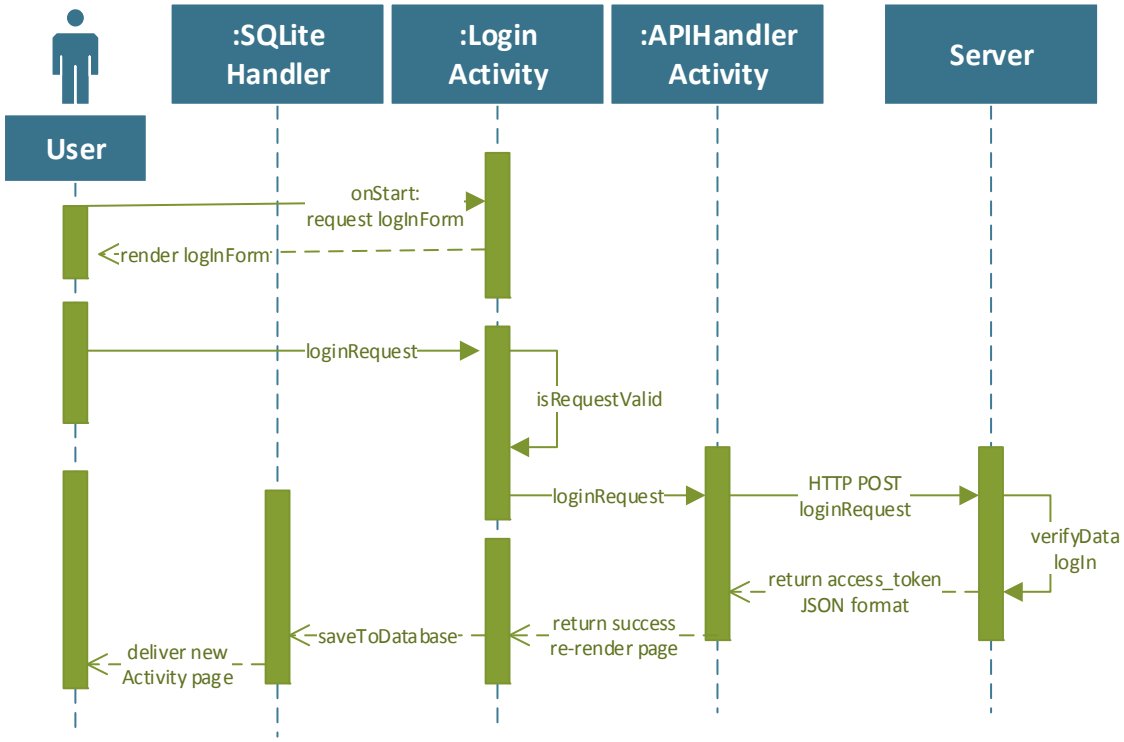
Typical Case: *Interactions inside server at service initialization*



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



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



Incremental and Regression Testing

Server Defect Log

Module	Server::User

Incremental Testing

Defect #	Description	Severity	How to correct
1	Extra information (error string) is returned besides the `LoginException`.	3	Removed the old error handling mechanism in `User.logIn`.
2	Extra information (error string) is returned besides the `RegisterException`.	3	Removed the old error handling mechanism in `User.registerNewUser`.
3	The exception is raised to PHP socket.	1	The core parsed the exception, sent the data, but did not nullify it. Fixed by doing so.

Regression Testing

Defect #	Description	Severity	How to correct
1	An Exception is always returned even when the log in fields are both valid.	1	Corrected the `if..else` blocks related to raising exceptions. Re-ran unit-test to make sure it is fixed.
2	When using faker to send logout request on an arbitrary user, an exception is raised, which contradicts the design that it should ignore all invalid logout requests.	2	Fixed the global checker in `User` class, having it ignore checking if the request is `logout` and `isUser` is `false`.

Module	Server::User::PasswordRetrieval

Incremental Testing

Defect #	Description	Severity	How to correct
1	Password retrieval email is not received. The server sendmail is not set up correctly.	2	Re-configured server settings.
2	The user cannot log in with the newly generated password in the email. The database is not updated before the Core sending the email.	1	Fixed by updating the password hash field in `users` table in database accordingly.

Regression Testing

Defect #	Description	Severity	How to correct
1	User still fails to log in with the newly generated password after resetting the password.	2	The algorithm mistreats salt field as the original password and vice versa. Fixed by sending the actual password.

Module	Server::ActionRouter

Incremental Testing

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

2	The user cannot log in with the newly generated password in the email. The database is not updated before the Core sending the email.	1	Fixed by updating the password hash field in `users` table in database accordingly.

Regression Testing

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

Client Defect Log

Module	Client::UserProfileActivity

Incremental Testing

Defect #	Description	Severity	How to correct
1	When user click View Profile, there is no response	1	The button is now linked to the correct interface
2	When user click View Profile, his photo is displayed in wrong position	2	Fixed the display location
3	No changes are saved after editing the profile	1	The changes are not sent to the server, now is being fixed
4	User's password instead of User name are displayed in the user profile	1	It is being removed and have changed back to user name
5	When another user login using the same device, only the first user's profile will be displayed	2	Data base has been added to store the users' profile

Regression Testing

Defect #	Description	Severity	How to correct
1	Some words are being covered after modifying the display location of photo	2	Modified the text display box
2	Sometimes it will display the wrong user profile	1	The algorithm has some problem and is still fixing

Module	Client::SQLHandler

Incremental Testing

Defect #	Description	Severity	How to correct
1	Multiple users cannot be added into the SQLite Database	1	Add 'users' tables into the database to save multiple user accounts information
2	The profile of users is not associated with its friends list table	2	Add 'friendslist_num' attribute to the profile table to associate with the friends list table

Regression Testing

Defect #	Description	Severity	How to correct
1	UserProfileActivity class cannot deal the situation that database exists multiple user accounts.	1	Add functions to change accounts to another one and obtain correct information for each user.
2	One friends list in the database is associated with multiple users.	2	Default value of friends list id in user table is set to `NULL` and update it by APIHandler.

Module	Client::APIHandler

Incremental Testing

Defect #	Description	Severity	How to correct
1	The format of UserProfile returned from the server cannot be recognized by the APIHandler	1	Add functions to recognize the format of UserProfile from the server.

Regression Testing

Defect #	Description	Severity	How to correct
1	The information returned by the	3	Revise the protocol with

	new UserProfile class is not recognized by client.		the server designer and adapt accordingly.
2	The function that requests userProfile from the server, some characters are not readable characters	2	Decode the character set to UTF-8

Appendix: New Test Cases For Server

Incremental Test Cases

Identity	Server::User::Exceptions.i01	**Severity**	3
Goal	Test if ActionRouter receives LoginException when the log in request is not valid.		
Instructions	1. Use faker to send invalid user email according to `Test Plan/User.properties.email.invalid*`; with valid password, note the server response 2. Use faker to send valid user mail with password of invalid length according to Test `Plan/properties.pwdLen.*`, note the server response		
Output	1. In server log, the LogInException should be recorded for both steps 2. Per the protocol, the server should respond with json error in both cases: `{ "code": 400, "errno": "200", "message": "Username or password not provided or of invalid format." }`		
Notes	Severity 3 at first test; 1 after “all clear” pass		

Identity	Server::User::Exceptions.i01	**Severity**	3
Goal	Test if ActionRouter receives LoginException when the log in request is not valid.		
Instructions	1. Use faker to send invalid user email according to `Test Plan/User.properties.email.invalid*`; with valid password, note the server response 2. Use faker to send valid user mail with password of invalid length		

	according to Test `Plan/properties.pwdLen.*`, note the server response
Output	1. In server log, the LogInException should be recorded for both steps 2. Per the protocol, the server should respond with json error in both cases: `{ "code": 400, "errno": "200", "message": "Username or password not provided or of invalid format." }`
Notes	Severity 3 at first test; 1 after first all clear pass

Regression Test Cases

Identity	**Server::User::Exceptions.r01**	**Severity**	**2**
Goal	Test if log in operation works after exception handling mechanism is introduced.		
Instructions	1. Use the client emulator to send valid user email according to the format specified in Test Plan, and record server response 2. Send valid password length according to the password length requirement in Test Plan, and record server response		
Output	1. Server should return the access token 2. Server should return the access token		
Notes			