CPEN321: Software Engineering

Test Plan Document

28th October 2016

Group: Cyann

Chen Chen (3281 6143)
Howard Zhou (4898 6137)
Justin Toh (2954 9136)
Luvian Wang (1693 9134)
Xi Chu (1735 3137)
Yufei Qiao (4885 7130)

1. Introduction

Cyann is intended to be an learning management system that provides quality user experience. To ensure users can get or post question and comment the backend API endpoints needs to be thoroughly tested.

2. Verification Strategy

We will conduct hallway usability test with random population of students by presenting them a functional product to get feedback of user experience and how the UX compare to other LMS currently on the market.

3. Non-Functional Testing and Results

Refer to the entries highlighted in RED in section 6.

4. Functional Testing Strategy

4.1 Testing Strategy (Frontend)

Since there is no need to test UI fragment individually(they tend to change with software update and maybe represented differently on different OS and devices). The frontend developers will be working on the test together as a team after the UI implementation is completed to ensure quality UX by performing hallway usability testing.

To ensure the accuracy of UI fragment(each component does what it intends to do), the frontend team will be using Appium for its UI automation feature.

Our testing approach consists of simulating onPress event using Appium's automation feature and print the retrieved data in JSON format and compare them against the expected outcome.

4.2 Testing Strategy (Backend)

Every developer is in-charged of writing unit tests for the features/task that they are assigned since each of them would be familiar with the code that they write. To make sure the each feature/API endpoint is interacting together correctly as intended, the backend subteam will work together to write integration tests once all the tasks/features in the current sprint is completed.

A feature is only considered as done when it meet all the written specs, passes all the unit tests & ready to be merged into MASTER branch.

We will be using **Postman** to write unit test & integration tests for our API endpoints since it has an intuitive GUI & a low learning curve for the subteam. These tests will then be executed using **Postman's** built-in test runner.

Our testing approach would be to make API calls to our backend server followed by asserting that we receive a JSON response & ensuring that it matches our expected values.

Unit tests will be (ideally) done before every commit. Integration tests will be done before merging any features to the MASTER branch.

4.3 System testing strategy

Systems level would consist of performing tests on every use cases via our app being deployed on a mobile device & the backend API server running live on a cloud. This is to simulate a real user interacting with our product so that we can catch all the potential bugs that a user might encounter.

System testing will be done at the end of every sprint, making sure that all bugs are caught and resolved before moving on the the next sprint.

4.4 Bug tracking strategy

Our bugs will be actively tracked using a google online spreadsheet so that the team can easily access it and edit them in real-time. We will consider migrating over to a bug-tracking tool once we have completed our backlog task ahead of time.

Bugs will be categorized into 3 classes:

- **A: Affects the app's basic functionality & renders the app usable.**
    - To be fixed ASAP.
    - Eg. Backend's API route not working.
    - Eg. List isn't being rendered on app.
- **B: Doesn't affects the app's basic functionality, app is usable but user experience is greatly affected.**
    - To be fixed by the end of the current sprint.
    - Eg. API Calls taking > 1sec due to slow DB queries.
    - Eg. App's font size is too large, causing text to be chopped off at the end.
- **C: Doesn't affects the app's basic functionality, can be resolved at a later date**.
    - To be fixed by the product's release date.
    - Eg. Title field is out of alignment by 1px.
    - Eg. Button color inconsistent with Hi-fi mockup.
    - Eg. Keys in API's JSON response contains spelling mistake.

5. Adequacy Criterion

5.1 Adequacy criteria for the frontend system:

- Each UI component will be tested for alignment on different devices (iOS/Android with different screen size)
- Ensure that each Ui component will perform its intended usage, for example: pop to a new scene / calling API endpoint.

5.2 Adequacy criteria for the backend system:

- Make sure that at least one test (each containing several test cases) exists for each API endpoint.
- Due to time constraint, it isn’t practical to write a test case for every single method being written.
- Once an API endpoint manages to pass a test, it is safe to assume that any helper methods called during its execution is functioning as intended.

6. Test Cases and Results

6.1 Frontend Tests (Last Updated on 28th October):

Test #	Description	Action/Input	Expected Result	P/F (if failed, show actual results)
1	Users wants to post a question	fetch(posts.createPostsByCourseId)	The ASK button is connect to the create post api endpoint.	P
2	Users wants to view all posts in a course	fetch(posts.findPostsByCourseId)	All posts within a course are presented to the user on screen	P
3	Users go to a course	gotoCourse(courseName)	Scene pops to postView	P
4	Users view files	getFile(url)	The pdf is presented on screen	**NOT YET IMPLEMENTED**
5	Users view question	fetch(comments.findAll)	Scene pops to answer	**NOT YET IMPLEMENTED**
6	Users answer question	fetch(comments.create)	The answer button is connected to the create comment api	**NOT YET IMPLEMENTED**

6.2 Backend Unit Tests (Last Updated on 28th October):

Test #	Description	Action/Input	Expected Result	P/F (if failed, show actual results)
1	Get all courses	courses.findAll	Refer to test files contained in /test folder inside Cyann's backend repository	P
2	Get a specific course	courses.findByID		P
3	Create a course	courses.create		P
4	Get all posts in a course	posts.findPostsByCourseId		P
5	Get a specific post	posts.findPostsByCourseIdAndPostId		P
6	Create a post in a course	posts.createPostsByCourseId		P
7	Edit a post	posts.updatePostsByCourseId		P
8	Delete a post	posts.deleteByCourseId		P
9	Delete all posts	posts.deleteAll		P
10	Get all comments in a post	comments.findAll		P
11	Get a specific comment	comments.findById		P
12	Create a comment	comments.create		P

13	Edit a comment	comments.updateById		P
14	Delete a comment	comments.deleteById		P
15	Set a comment as an answer	comments.setAsAnswer		P
16	Unset a comment as an answer	comments.unsetAsAnswer		P
17	Upvote a comment	comments.upvote		P
18	Downvote a comment	comments.downvote		P
19	Reset a vote on a comment	comments.resetVote		P
20	User Login	userLogin.login		P
21	User Sign up	userLogin.signUp		P
22	Get all users	userInfo.findAll		P
23	Get a specific user	userInfo.findById		P
24	Get all posts from a specific user	userInfo.findPostById		P
25	Get all comments from a specific user	userInfo.findCommentById		P
26	Upload a file	fileUpload.upload		**NOT YET IMPLEMENTED**
27	Download a file	fileUpload.download		**NOT YET IMPLEMENTED**

28	List all files uploaded	fileUpload.showFiles		**NOT YET IMPLEMENTED**

6.3 Backend Integration Tests (Last Updated on 28th October):

Test #	Description	Action/Input	Expected Result	P/F (if failed, show actual results)
1	User SignUp & Login	userLogin.login, userLogin.signUp	User is able to create an account & login immediately	P
2	Instructor creates a course	courses.create courses.findByID	Instructor can create a course & sees it immediately	P
3	User joins a course		User can join a course & start viewing posts	**NOT YET IMPLEMENTED**
4	User posts a question & edits it after.	posts.createPostsByCourseId posts.updatePostsByCourseId posts.findPostsByCourseIdAndPostId	User creates a post & able to edit it after creation	P
5	User votes a comment & makes a comment	Comments.upvote Comments.downvote comments.resetvote comments.create	User able to upvote,downvote or remove vote & make a comment	P
6	Instructor unsets & sets another post as an answer	comments.setAsAnswer comments.unsetAsAnswer	Instructor able to unset a post as answer & set another post as answer	P
7	User request for tutoring service			**NOT YET IMPLEMENTED**
8	Tutor responds to			**NOT YET IMPLEMENTED**

	tutoring service			
9	User uploads file & downloads it	fileUpload.upload fileUpload.download	User able to upload a file & subsequently download it later.	P
10	User deletes file			**NOT YET IMPLEMENTED**

6.4 System Tests (Last Updated on 28th October):

Test #	Description	Action/Input	Expected Result	P/F (if failed, show actual results)
1	User SignUp & Login	User taps SIGN UP button, enter credentials. User taps LOGIN button, enter credentials & taps DONE button	User is able to create an account & login immediately	**NOT YET IMPLEMENTED**
2	Instructor creates a course	User taps CREATE COURSE button, enter course details and taps DONE button	Instructor can create a course & sees it immediately	**NOT YET IMPLEMENTED**
3	User joins a course	User taps on a course and gets enrolled if he/she is not already enrolled in that course	User can join a course & start viewing posts	**NOT YET IMPLEMENTED**
4	User posts a question & edits it after.	User goes to post question view and enter text for question title and content then send enter	User creates a post & able to edit it after creation	**NOT YET IMPLEMENTED**
5	User votes a comment & makes a comment	User goes to comments view and press the post button to answer a question or press the upvote button to upvote someone else's answer	User able to upvote,downvote or remove vote & make a comment	**NOT YET IMPLEMENTED**
6	Instructor unsets & sets another	Instructor picks an answer from a list of all answers of a question	Instructor able to unset a post as answer & set	**NOT YET IMPLEMENTED**

	post as an answer		another post as answer	
7	User request for tutoring service	User goes to tutor list view and press on a name to select tutor		**NOT YET IMPLEMENTED**
8	Tutor responds to tutoring service	Tutor gets the request and send a message back to the person who requested		**NOT YET IMPLEMENTED**
9	User uploads file & downloads it	User goes to file list view to upload/download file	User able to upload a file & subsequently download it later.	**NOT YET IMPLEMENTED**
10	User deletes file	User select his/her own file and press on the delete button		**NOT YET IMPLEMENTED**