CoLab

Functional Test Plan

Model Testing - Black Box Unit Tests

Overview

The core of the CoLab system contains several classes that represent their individual tables within the database. Each class may also contain several helper functions based on the relationships of the database to make gathering and processing data much easier for the development team.

To ensure integrity of the core files test cases were created in PHP Unit that was provided by the Laravel Framework. PHP Unit offers an automatic test suite that can test the output of functions by using true/false assertions and handling any exceptions with stack trace support .

Before the development of a class we created several test cases that would cover what we want that class to achieve. By testing all classes with their expected outputs after every change to the code allowed our developers to quickly find and fix errors with the given line number of that class.

PHP Unit also offers code coverage analysis by tracking the code being testing by each assertion. This shows us a report to easily identify and highlight any code, function or class that is not covered by our existing tests.

The last major feature we implemented during our testing was using Laravel's built in environment handling to create a virtual database into the computer's memory. This allows our existing data to be protected against untested code and also offers the option of testing the software without a physical database present.

Model Unit Test Results


```
PHPUnit 3.7.37 by Sebastian Bergmann.

Configuration read from D:\Users\Jacob\Dropbox\www\colabfinal\phpunit.xml

...................

Time: 25.82 seconds, Memory: 41.25Mb

OK (19 tests, 67 assertions)

Generating code coverage report in HTML format ... done
```

Overall we had 19 test classes that performed 67 assertions. The final build passed all of them and tells us that our previous code was not broken by the implementation of new features. The tests themselves can be found in '/app/tests/' of the provided colab folder.

Model Code Coverage

	Code Coverage								
	Lines			**Functions and Methods**			**Classes and Traits**		
Total		100.00%	93 / 93		100.00%	80 / 80		100.00%	23 / 23
Acknowledgement.php		100.00%	3 / 3		100.00%	3 / 3		100.00%	1 / 1
Comment.php		100.00%	2 / 2		100.00%	2 / 2		100.00%	1 / 1
Datafile.php		100.00%	2 / 2		100.00%	2 / 2		100.00%	1 / 1
Education.php		100.00%	1 / 1		100.00%	1 / 1		100.00%	1 / 1
Interest.php		100.00%	1 / 1		100.00%	1 / 1		100.00%	1 / 1
Job.php		100.00%	1 / 1		100.00%	1 / 1		100.00%	1 / 1
Meeting.php		100.00%	1 / 1		100.00%	1 / 1		100.00%	1 / 1
Message.php		100.00%	2 / 2		100.00%	2 / 2		100.00%	1 / 1
MessageLobby.php		100.00%	7 / 7		100.00%	7 / 7		100.00%	1 / 1
MessageReader.php		100.00%	4 / 4		100.00%	3 / 3		100.00%	1 / 1
Milestone.php		100.00%	6 / 6		100.00%	6 / 6		100.00%	1 / 1
Notification.php								100.00%	1 / 1
Personalachievement.php		100.00%	1 / 1		100.00%	1 / 1		100.00%	1 / 1
Post.php		100.00%	5 / 5		100.00%	5 / 5		100.00%	1 / 1
Project.php		100.00%	25 / 25		100.00%	14 / 14		100.00%	1 / 1
ProjectTeamachievement.php		100.00%	4 / 4		100.00%	4 / 4		100.00%	1 / 1
Rule.php		100.00%	1 / 1		100.00%	1 / 1		100.00%	1 / 1
Skill.php		100.00%	1 / 1		100.00%	1 / 1		100.00%	1 / 1
Stress.php		100.00%	2 / 2		100.00%	2 / 2		100.00%	1 / 1
Task.php		100.00%	4 / 4		100.00%	4 / 4		100.00%	1 / 1
Teamachievement.php		100.00%	1 / 1		100.00%	1 / 1		100.00%	1 / 1
User.php		100.00%	15 / 15		100.00%	14 / 14		100.00%	1 / 1
UserPersonalachievement.php		100.00%	4 / 4		100.00%	4 / 4		100.00%	1 / 1

Legend

Low: 0% to 35%   **Medium**: 35% to 70%   **High**: 70% to 100%

Generated by PHP_CodeCoverage 1.2.17 using PHP 5.5.12 and PHPUnit 3.7.37 at Sat Nov 8 13:58:05 EST 2014.

Using the above mentioned assertions our developers were able to maintain 100% code coverage for our model classes. This tells us that each line and function in our model classes were tested at least once. The full report can be found in the 'Supporting Material and Appendices' section

Controller and View Testing - Case Scenarios

Overview

Due to the nature of how Laravel handles the interface through the use of blades, our group decided it would be faster and easier to develop a set of case scenarios to run through instead of taking the time to learn to use the resources for automated testing. By following the case scenarios with every testing stage we would ensure that the system would display the information passed from the models correctly.

After creating the necessary model for a page, the developer would create a list of features. They would then work implementing those features testing the controller and view components along the way. Once the developer was happy with their results, the code would be pushed to the git repository to be tested by the rest of the team.

Results

Below are some of the most common scenarios that were regularly tested. An enlarged version is available in the 'Supporting Material and Appendices' section

Page Tested	Function Tested	Input	Expected Results	Actual Results
Sign Up	Sign Up	Create account using a unique email and password	Account created. Redirected to login page	Success
	Email already exists	Create account with an existing email	Redirected back to sign up with error message	Success
	Sign up with social media	Selecting a social media account to sign up with	Account created using the details from the social media account	Links do nothing
Login	Login	Existing email and password	Login successful. Redirected to user dashboard	Success
	Login with incorrect details	Email that does not exist or incorrect password	Redirected back to log in with error message	Success
	Login with social media	Selecting a social media account to login	Login successful. Redirected to user dashboard	Links do nothing
Contact Form	Submit contact form	Fill out contact form	Redirected back to current page with success message	Submit does nothing
User Dashboard	Newsfeed		Various information about projects to be displayed	Success
Messages	Send new message	Selected multiple recipients and send a message	Message sent to intended recipients	Success
	Reply to message	Reply to existing message	Message sent as reply	Success
Profile	View Profile		View existing profile data	Success
	Edit Profile	Adjust some of the user profile information	Information updated with new changes	Success
Group Members	View Members		Information regarding the members of the group	Success
	Add Member	Search for a user and add them to the group	The selected user is added to the group	Success
	Remove Member	Search for an existing member and remove them from the group	The selected member is removed from the group	Not implemented
Team Rules	View Rules		Group rules are displayed	Success
	Add rule	Input a rule and add it.	The rule is added to the group	Success
Milestones	View milestones		View current milestones	Success
	Add milestone	Create milestone with appropiate information	Added milestone to group	Success
	Edit milestone	Edit details for a milestone	Milestone information updated	Not Implemented
	Delete milestone	Select milestone to delete	Milestone is deleted	Not Implemented
Tasks	View tasks		View current tasks	Success
	Add task	Create task with appropiate information	Added task to group	Success
	Edit task	Edit details for a task	Task information updated	Success
Stress	View group stress		View information about the stress level of the group and members	Success
	Edit stresss	Change stress level	User stress and group stress updated	Success
Files	View file		View current files	Success
	Add file	Create file with appropiate information	Added file to group	Success
	Delete file	Select file to delete	file is deleted	Success
Meetings	View meetings		View current meetings	Success
	Add meeting	Create meeting with appropiate information	Added meeting to group	Success
	Delete meeting	Select meeting to delete	meeting is deleted	Success