Quiz Manager Submission:

Testing Documentation

Prepared by: Linna Trieu, Associate Software Engineer at Paddle

Qualification: Level 4 Software Development Apprenticeship with Makers Academy

Prepared for: BCS, Digital Industries Apprenticeship

Date: September 2020

TESTING DOC

Approach

Testing is a critical phase of any software engineering project. The objective of my testing will be to validate that the Quiz Manager program meets its requirements.

I have tried to embed Testing throughout the project lifecycle as a continuous integration between software development and testing, instead of just leaving testing to 'one phase' at the end of development,

I have tested throughout development, assessing expected versus actual outcomes. In line with agile testing practices, this has allowed me to keep a tight feedback loop on the development process; for each product feature - continuously developing, testing, improving and iterating on any program flaws identified.

Types of Testing

This document intends to articulate the formalised Testing strategy implemented on the Quiz Manager project.

For Quiz Manager, I have organised my testing strategy in two approaches:

1. **Automated Testing** - writing program test code in realtime as I develop in order validate the actual behaviour of the system is as intended during development.
2. **Manual Testing** - running the application locally so that I go through a set of user journeys and flows to validate the expected behaviour from a user perspective. For each software feature and user story, I conducted a round of manual testing. I formalised this using a Kanban board, ensuring all tickets went into 'Ready for Testing' column. The manual testing round had to be completed in order for the tickets and product feature to be considered 'Done' and moved into that column (figure 3).

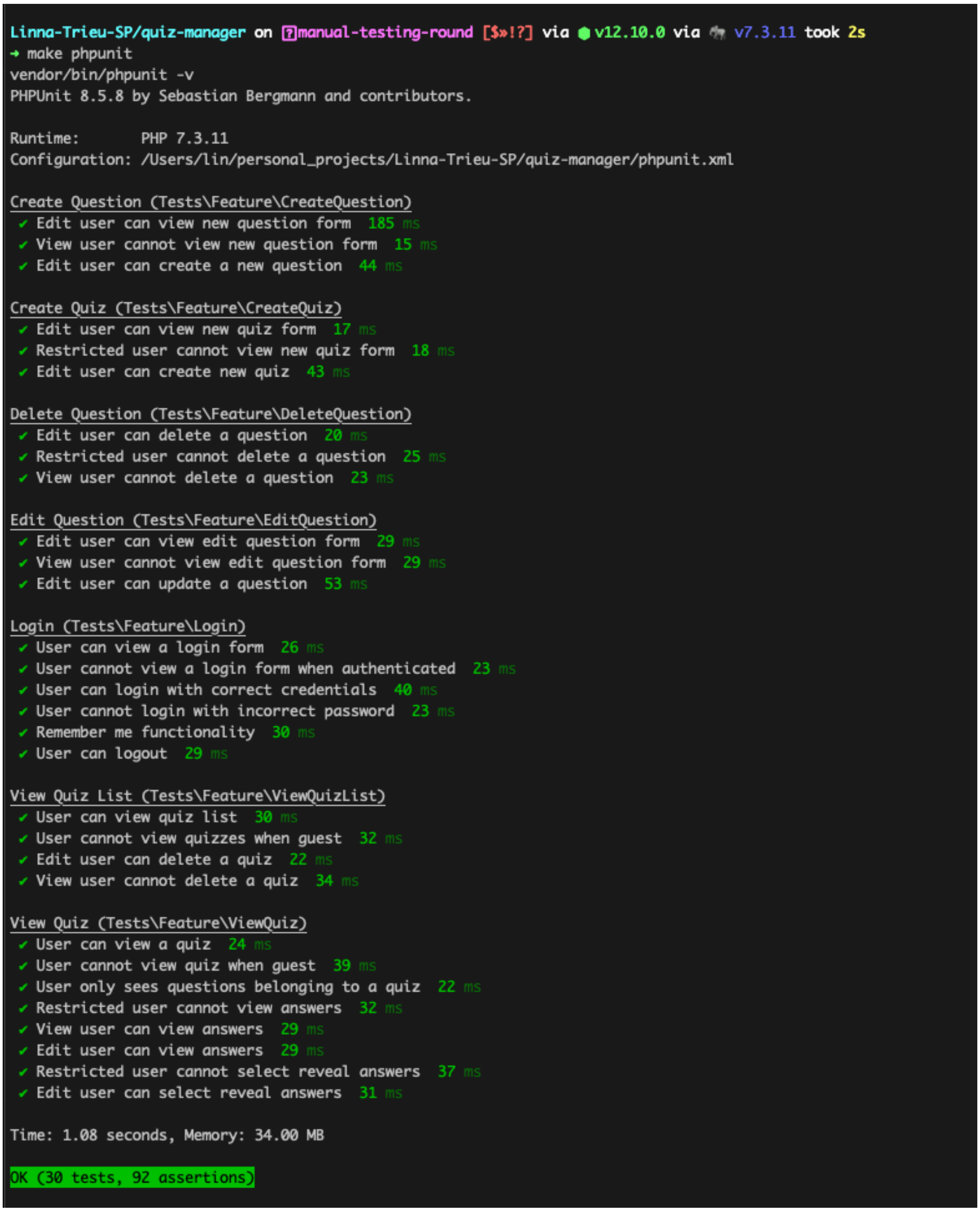
Automated Testing

- Automated tests were written throughout code development in the program code itself.
- I decided to use PHPUnit, because it is a popular industry PHP testing framework, well supported by Laravel and a framework I have the most experience with in my work.

- As this is a relatively small application with a tight timeframe, I elected to use integration and feature tests to test Quiz Manager. This will ensure the application is functioning as intended based on end-to-end interactions. Using feature tests has enabled me to test a slice of functionality in the website, which touches many methods and interacts with dependencies like databases.
- The logic in this program was relatively simple, with the complexity resulting from database CRUD actions (create, read, update and delete). As this is a database-driven website, it is important for the reliability of the website to test the program's integrations with the database.
- Led by Laravel documentation, I decided to use an in-memory SQLite database. As another relationship database type, it has closely mirrored the functionality of mySQL, my development database (and in this case, the production database). Having an in-memory database means an instance of the database can be spun up, configured and disposed of by every test that needs it.
- With this in mind, please see the results of the Quiz Manager program code test suite, in figure 1.
- In line with PHPUnit best practice and PSR-12, I have named tests following convention. Tests are grouped into files which correspond to a product feature, and each test should be in camel case and descriptive of the exact functionality tested.
- In writing every test I have an expected outcome of the functionality. This is formalised in the actual program test code itself (if you navigate to any test code, example in figure 2).
- I have followed the AAA (Arrange-Act-Assert) pattern in my tests, that is to divide my test into three sections. Arrange the code required to set up the test, act by invoking the functionality being tested, and assert your expectations are by checking whether the actual result matches expectation.

Figure 1. Automated Test Suite - Full list of tests run in the program with their results *(30 successful tests with 92 assertions passing)*

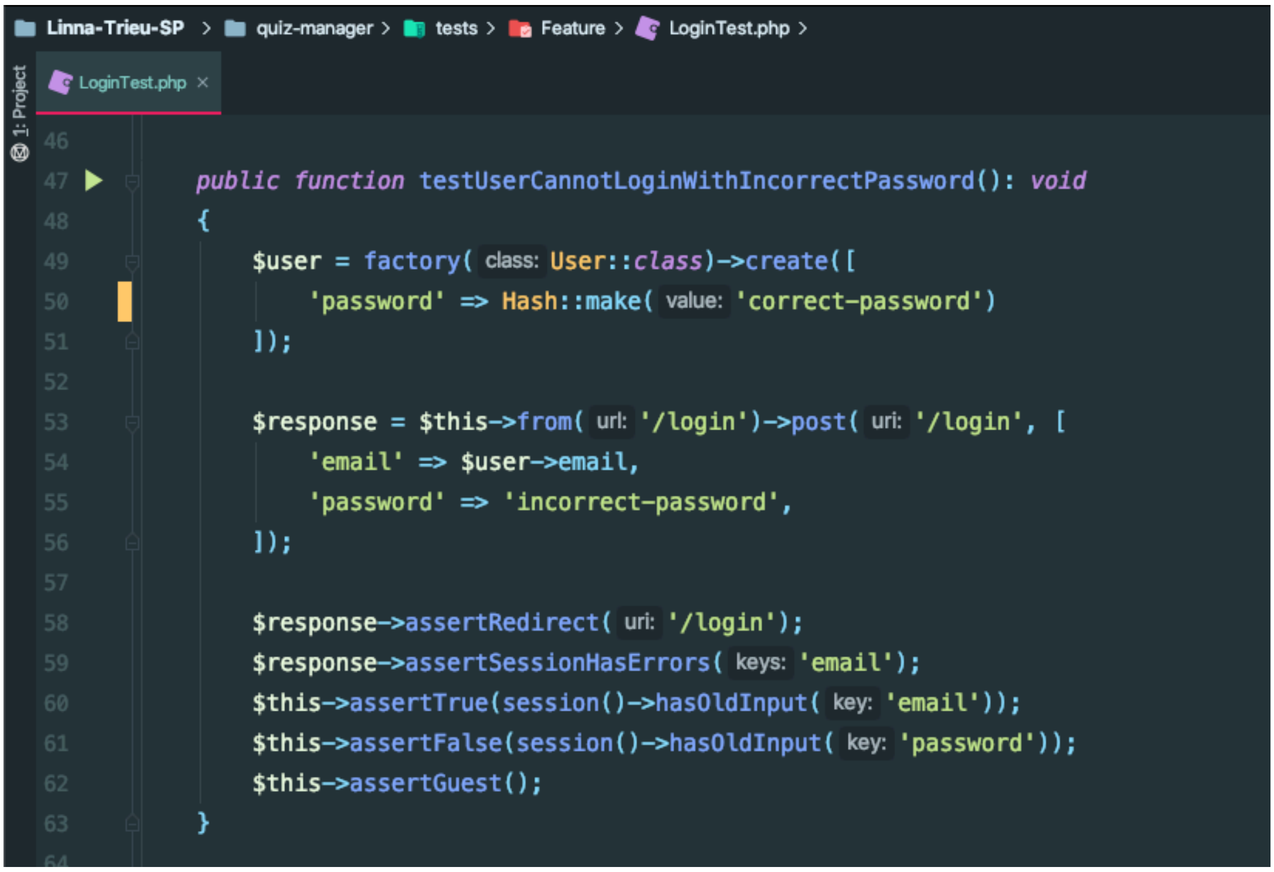
- *Note:* to run the test suite yourself, please navigate to the Quiz Manager program code in your command line/terminal. Set up the application (as per README), and once set up, run the command '*make phpunit*' or '*php artisan test*'. All tests should be passing.



```
Linna-Trieu-SP/quiz-manager on ?manual-testing-round [$»!?] via ● v12.10.0 via 🐘 v7.3.11 took 2s
→ make phpunit
vendor/bin/phpunit -v
PHPUnit 8.5.8 by Sebastian Bergmann and contributors.

Runtime:       PHP 7.3.11
Configuration: /Users/lin/personal_projects/Linna-Trieu-SP/quiz-manager/phpunit.xml

Create Question (Tests\Feature\CreateQuestion)
 ✔ Edit user can view new question form  185 ms
 ✔ View user cannot view new question form  15 ms
 ✔ Edit user can create a new question  44 ms

Create Quiz (Tests\Feature\CreateQuiz)
 ✔ Edit user can view new quiz form  17 ms
 ✔ Restricted user cannot view new quiz form  18 ms
 ✔ Edit user can create new quiz  43 ms

Delete Question (Tests\Feature\DeleteQuestion)
 ✔ Edit user can delete a question  20 ms
 ✔ Restricted user cannot delete a question  25 ms
 ✔ View user cannot delete a question  23 ms

Edit Question (Tests\Feature\EditQuestion)
 ✔ Edit user can view edit question form  29 ms
 ✔ View user cannot view edit question form  29 ms
 ✔ Edit user can update a question  53 ms

Login (Tests\Feature\Login)
 ✔ User can view a login form  26 ms
 ✔ User cannot view a login form when authenticated  23 ms
 ✔ User can login with correct credentials  40 ms
 ✔ User cannot login with incorrect password  23 ms
 ✔ Remember me functionality  30 ms
 ✔ User can logout  29 ms

View Quiz List (Tests\Feature\ViewQuizList)
 ✔ User can view quiz list  30 ms
 ✔ User cannot view quizzes when guest  32 ms
 ✔ Edit user can delete a quiz  22 ms
 ✔ View user cannot delete a quiz  34 ms

View Quiz (Tests\Feature\ViewQuiz)
 ✔ User can view a quiz  24 ms
 ✔ User cannot view quiz when guest  39 ms
 ✔ User only sees questions belonging to a quiz  22 ms
 ✔ Restricted user cannot view answers  32 ms
 ✔ View user can view answers  29 ms
 ✔ Edit user can view answers  29 ms
 ✔ Restricted user cannot select reveal answers  37 ms
 ✔ Edit user can select reveal answers  31 ms

Time: 1.08 seconds, Memory: 34.00 MB

OK (30 tests, 92 assertions)
```

Figure 2. Example of a Program Code Test: ‘testUserCannotLoginWithIncorrectPassword’

- Functionality being tested: a guest user cannot login into the website with incorrect credentials
- Arrange - set up the testing options and pre-requisites for this test. In pseudo-code here, a user needs to be created in the database with password X.
- Act - Invoke the feature being tested. The user needs to attempt to login in with password Y.
- Assert - verify the result against your expectations. Mainly assert that the user is still a guest, has been re-directed to the login page, and that there is an error in the session. I have also asserted that the email field has the previous input in the form, and the password field does not.

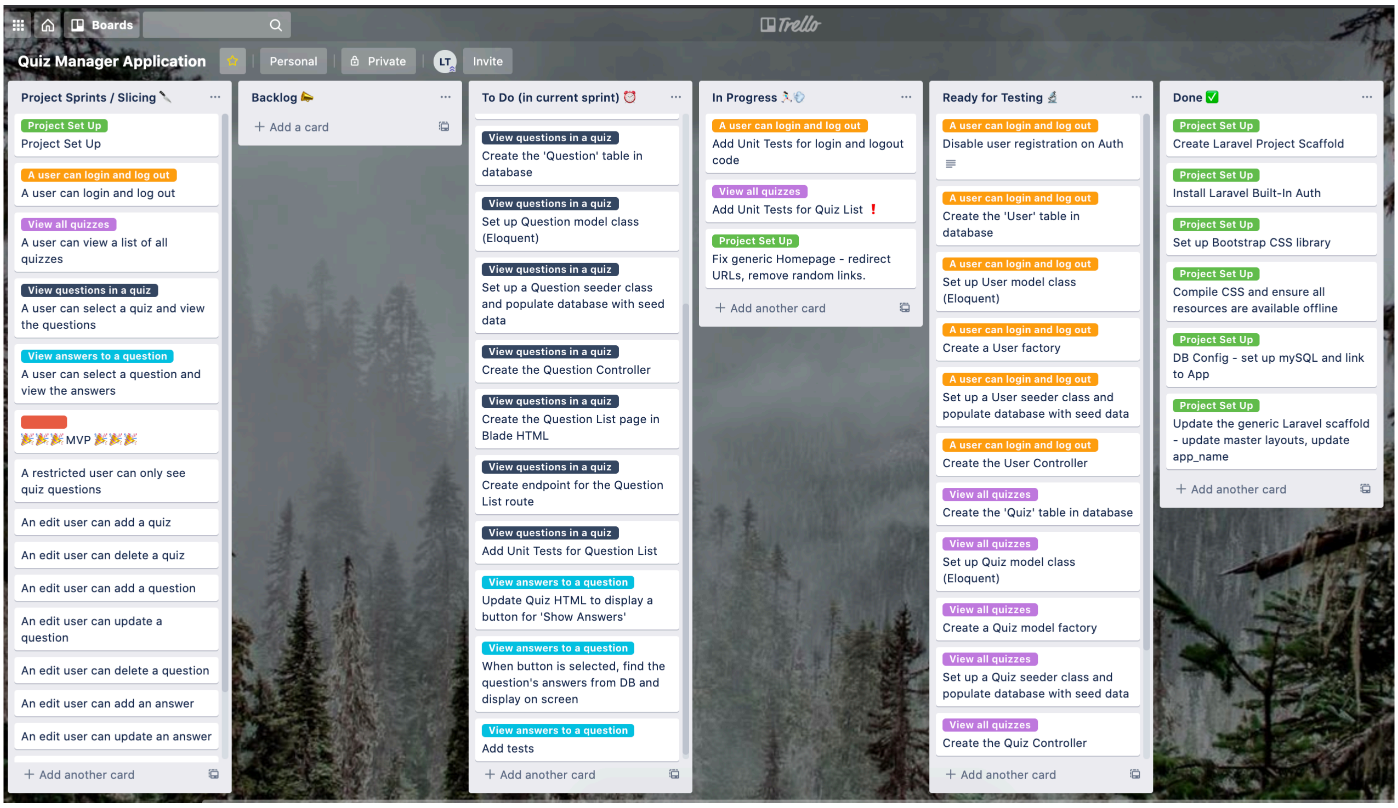


Linna-Trieu-SP > quiz-manager > tests > Feature > LoginTest.php >
LoginTest.php ×
1: Project

```php
public function testUserCannotLoginWithIncorrectPassword(): void
{
    $user = factory( class: User::class)->create([
        'password' => Hash::make( value: 'correct-password')
    ]);

    $response = $this->from( url: '/login')->post( uri: '/login', [
        'email' => $user->email,
        'password' => 'incorrect-password',
    ]);

    $response->assertRedirect( uri: '/login');
    $response->assertSessionHasErrors( keys: 'email');
    $this->assertTrue(session()->hasOldInput( key: 'email'));
    $this->assertFalse(session()->hasOldInput( key: 'password'));
    $this->assertGuest();
}
```

Figure 3. Screenshot of Trello board setup with the 'Ready for Testing' column in workflow.

- All tickets and product features had to be manually tested and passing, before I would consider them ready to be moved to the 'Done' column. The coloured label on each ticket stated the user story it referred to, i.e. 'a user can login and logout' and therefore what behaviour the manual tests were validating.
- Testing was a fully integrated part of the project's software development workflow and embedded throughout the work. Any bugs or deficiencies noted in tested were either immediately fixed as a priority (in order to move the ticket to done); or a new dedicated ticket was created to fix as a separate bug. This tight feedback loop ensured quality and that the implementation of the software project would be more likely to meet its user requirements.



Project Sprints / Slicing
- Project Set Up — Project Set Up
- A user can login and log out — A user can login and log out
- View all quizzes — A user can view a list of all quizzes
- View questions in a quiz — A user can select a quiz and view the questions
- View answers to a question — A user can select a question and view the answers
- 🎉🎉🎉 MVP 🎉🎉🎉
- A restricted user can only see quiz questions
- An edit user can add a quiz
- An edit user can delete a quiz
- An edit user can add a question
- An edit user can update a question
- An edit user can delete a question
- An edit user can add an answer
- An edit user can update an answer
- + Add another card

Backlog
- + Add a card

To Do (in current sprint) ⏰
- View questions in a quiz — Create the 'Question' table in database
- View questions in a quiz — Set up Question model class (Eloquent)
- View questions in a quiz — Set up a Question seeder class and populate database with seed data
- View questions in a quiz — Create the Question Controller
- View questions in a quiz — Create the Question List page in Blade HTML
- View questions in a quiz — Create endpoint for the Question List route
- View questions in a quiz — Add Unit Tests for Question List
- View answers to a question — Update Quiz HTML to display a button for 'Show Answers'
- View answers to a question — When button is selected, find the question's answers from DB and display on screen
- View answers to a question — Add tests
- + Add another card

In Progress
- A user can login and log out — Add Unit Tests for login and logout code
- View all quizzes — Add Unit Tests for Quiz List ❗
- Project Set Up — Fix generic Homepage - redirect URLs, remove random links.
- + Add another card

Ready for Testing
- A user can login and log out — Disable user registration on Auth
- A user can login and log out — Create the 'User' table in database
- A user can login and log out — Set up User model class (Eloquent)
- A user can login and log out — Create a User factory
- A user can login and log out — Set up a User seeder class and populate database with seed data
- A user can login and log out — Create the User Controller
- View all quizzes — Create the 'Quiz' table in database
- View all quizzes — Set up Quiz model class (Eloquent)
- View all quizzes — Create a Quiz model factory
- View all quizzes — Set up a Quiz seeder class and populate database with seed data
- View all quizzes — Create the Quiz Controller
- + Add another card

Done ✅
- Project Set Up — Create Laravel Project Scaffold
- Project Set Up — Install Laravel Built-In Auth
- Project Set Up — Set up Bootstrap CSS library
- Project Set Up — Compile CSS and ensure all resources are available offline
- Project Set Up — DB Config - set up mySQL and link to App
- Project Set Up — Update the generic Laravel scaffold - update master layouts, update app_name
- + Add another card