11/18/2016

V-Menu Restaurant Management System

Building the Restaurant of Tomorrow Today

School: University of North Florida

Course: Software Engineering (CEN 4010)

Instructor: Dr. Douglas Leas

Team #2

Lead Developer: Matthew Boyette

Developer: William Mejia

Developer: Demetrius Myers

Lead Designer: Slaven Popadic

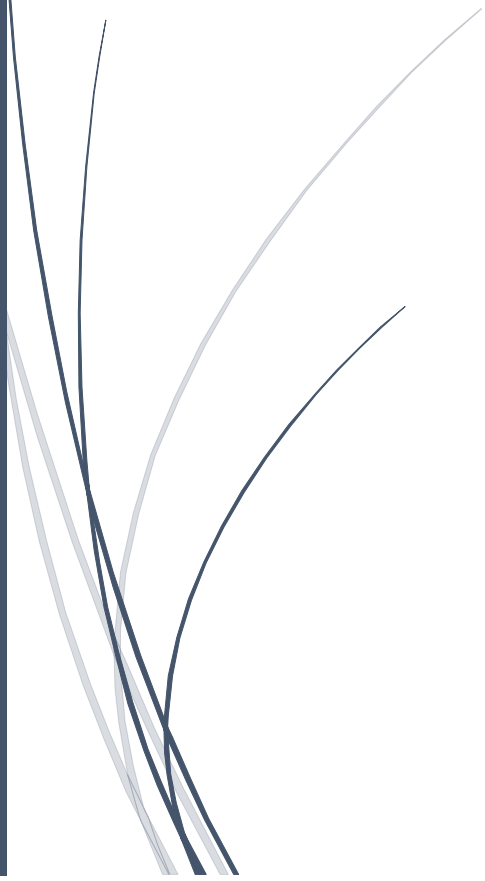

Table of Contents

Section 1: Overview

V-Menu is a system designed to turn the day-to-day logistics of managing a restaurant into a problem solvable using distributed computing. By turning the individual human elements of the restaurant into computable problems, we hope to push the boundaries and take the next step toward a completely automated restaurant. Although a fully automated restaurant is still relatively infeasible with today's level of off-the-shelf consumer technology, it won't be long before advances in robotics make such an occurrence not only practical, but also commonplace. There are three primary factors of functionality that must be addressed before robotic employees become ready for mainstream adoption.

- Precision of movement. Tasks like cooking and waiting tables require precise movements, and the objects being moved are designed to be manipulated by a human hand.
- Collision avoidance. A restaurant is usually a bustling place with a great deal of movement (not only by employees, but also customers). Being able to avoid colliding with obstacles is critical.
- Communication interface. Computerized speech recognition still has some ways to go before a randomly chosen customer can reliably communicate their orders verbally to a robotic employee.

Many restaurants currently have similar systems in place already, but the feature-set and overall design is inconsistent. Further, many of these systems focus solely on patrons who want food delivered to an exterior location, or patrons who wish to pick up their food at the restaurant and then take it with them to dine elsewhere (usually a home or office).

Our goal is two-fold: to serve the needs of those dining inside the restaurant, and to merge the functionality of existing systems for a consistent design and interface which can be easily customized or extended to suit each individual client.

Section 2: Project Team

- Matthew is the team leader and lead developer. The parts of this deliverable that he contributed are the title page, table of contents, the overview (this page), and the assumptions/risks. In addition, he created some of the test cases presented in this document. Matthew also participated in our Test Plan presentation.
- William is a developer. The part of this deliverable that he contributed is the unit test approach. William also participated in our Test Plan presentation.
- Demetrius is a developer. The part of this deliverable that he contributed is the integration test approach.
- Slaven is the lead designer. The part of this deliverable that he contributed are our test cases.

Additionally, every member of the team participated in generating ideas and discussing implementation related topics during both our regular team meetings and off-hours using our Slack channel.

Section 3: Unit Test Approach

Our project employs both visual code reviews and formal testing of the code to maximize the number of faults discovered. At the core of our project is the Drupal content management system (CMS). Drupal has built-in features designed to assist in the formal testing process by providing an automated testing mechanism. This facilitates unit testing, integration testing (more on that in the next section), and regression testing. Our goal is to have test coverage for most (if not all) of the components and features of the system. This automation is enabled via Cron, the standard Unix time-based task-scheduling application. We run these automated tests before and after every code change to ensure that no change breaks the existing codebase. This makes regression testing simple and efficient.

We have taken an egoless attitude to our test plan. Our Drupal module is just one of many. It needs to play nice with the rest of the Drupal ecosystem and conform to existing Drupal standards. To implement this philosophy, we include tests for every patch. The Drupal nomenclature does not distinguish between patches to fix bugs (patching as we have discussed it in class) and patches to add new features and subsystems (refactoring as we have discussed it in class).

- Patches to Fix Bugs
  - Must include a test that fails without the change and passes with the change. This helps reviewers understand what the bug is, demonstrates that the code fixes the bug, and ensures the bug will not reappear due to later code changes.
- Patches to Implement Features/Subsystems
  - Must include new unit and/or functional tests in the patch. This serves to both demonstrate that the code works, and ensure that later changes do not break the new functionality.

Automated tests are broken up into two different types, unit tests and functional tests. Functional tests facilitate integration testing, and are covered in detail in the next section.

- Unit Tests
  - These use a black box approach to test classes and components at a low-level. A well-defined input is tested to ensure that the class or component deterministically produces an expected output. These tests are internally implemented via the industry-standard PHPUnit framework.
  - The test is defined as a class that extends the \Drupal\Tests\UnitTestCase class.
  - The class name must end with the “Test” suffix.
  - The namespace must be a subspace/subdirectory of \Drupal\vmenu\Tests.
  - The test class file must be named and placed under the vmenu/tests/src/Unit directory per the PSR-4 standard.
  - The test class needs a phpDoc comment block with a description and a @group annotation which gives information about the test.
  - Methods in the test class whose names start with 'test' are the actual test cases. Each one should test a logical subset of the functionality.

Section 4: Integration Test Approach

Integration testing occurs at each stage of the development process using the sandwich approach as discussed in class. This process is facilitated through automated functional tests.

- Functional Tests
  - These use a white box approach to test systems at a high-level. This typically involves GUI input and complex web-based output. These tests are implemented via the custom Drupal-specific SimpleTest module.
  - For functional tests of the web output of Drupal, we define a class that extends \Drupal\simpletest\WebTestBase, which contains an internal web browser and defines many helpful test assertion methods that we can use in our tests. We can specify modules to be enabled by defining a $modules member variable. By default, WebTestBase uses a "testing" install profile, with a minimal set of modules enabled.
  - For functional tests that do not test web output, we define a class that extends \Drupal\KernelTests\KernelTestBase. This class is much faster than WebTestBase, because instead of making a full install of Drupal, it uses an in-memory pseudo-installation (like what the installer and update scripts use). To use this test class, we will need to create the database tables we need and install needed modules manually.
  - The namespace must be a subspace/subdirectory of \Drupal\vmenu\Tests.
  - The test class file must be named and placed under the vmenu/src/Tests directory per the PSR-4 standard.
  - The test class needs a phpDoc comment block with a description and a @group annotation, which gives information about the test.
  - We may also override the default setUp() method, which can set be used to set up content types and similar procedures.
  - In some cases, a test module may need to be written to support the test; We will put such modules under the vmenu/tests/modules directory.
  - Methods in the test class whose names start with 'test', and which have no arguments, are the actual test cases. Each one should test a logical subset of the functionality, and each one runs in a new, isolated test environment, so it can only rely on the setUp() method, not what has been set up by other test methods.

The combination of black box and white box testing we are employing means we can be confident that we have completely tested each component, and we maximize our chance of finding faults in our code.

Section 5: Test Cases

Section 5A: Test Case #01 – View Menu

- Preconditions/Setup: Tester must be at the home page.
- Requirement(s) Tested: View Menu
- Action 1: Tester activates View Menu link.
- Verify 1: Tester confirms that the menu page is displayed within the content pane.

Section 5B: Test Case #02 – Place Order

- Preconditions/Setup: Tester must be at the menu page.
- Requirement(s) Tested: Place Order
- Action 1: Tester selects their desired menu items.
- Verify 1: Tester confirms that the selected items are added to the live preview of their receipt.
- Action 2: Tester activates the Place Order button.
- Verify 2: Tester confirms that the order confirmation page is displayed within the content pane.

Section 5C: Test Case #03 – Confirm Order

- Preconditions/Setup: Tester must have placed an order, and be at the order confirmation page.
- Requirement(s) Tested: Confirm/Cancel Order
- Action 1: Tester activates the Confirm Order button.
- Verify 1: Tester confirms that they are now prompted for payment information.
- Action 2: Tester enters payment information, and activates the Complete Order button.
- Verify 2: Tester confirms that they receive their receipt via email.

Section 5D: Test Case #04 – Cancel Order

- Preconditions/Setup: Tester must have placed an order, and be at the order confirmation page.
- Requirement(s) Tested: Confirm/Cancel Order
- Action 1: Tester fails to activate the Confirm Order button (either by leaving the confirmation page or losing connection to the server).
- Action 2: Tester activates the Orders navbar link.
- Verify 1: Tester confirms that nothing was added to the order status page.

Section 5E: Test Case #05 – Flag Order Completed

- Preconditions/Setup: Tester must have confirmed an order, be logged in as a Kitchen Staff employee, and be at the order status page.
- Requirement(s) Tested: Flag Order as Completed
- Action 1: Tester activates the Completed button.
- Verify 1: Tester confirms that the order leaves the Kitchen Queue.

Section 5F: Test Case #06 – Flag Order Delivered

- Preconditions/Setup: Tester must have marked an order as completed, be logged in as a Wait Staff employee, and be at the order status page.
- Requirement(s) Tested: Flag Order as Delivered
- Action 1: Tester activates the Delivered button.
- Verify 1: Tester confirms that the order leaves the Wait Queue.

Section 5G: Test Case #07 – View Order Statuses (Anonymous Customer)

- Preconditions/Setup: N/A
- Requirement(s) Tested: View Order Status
- Action 1: Tester activates the Orders navbar link.
- Verify 1A: Tester confirms that the order status page is displayed within the content pane.
- Verify 1B: Tester confirms that they are prompted for an order identification number.
- Action 2: Tester enters a valid order identification number.
- Verify 2: Tester confirms that the associated order is displayed on the page.

Section 5H: Test Case #08 – View Order Statuses (Authenticated Customer)

- Preconditions/Setup: Tester must be logged in as a Customer.
- Requirement(s) Tested: View Order Status
- Action 1: Tester activates the Orders navbar link.
- Verify 1A: Tester confirms that the order status page is displayed within the content pane.
- Verify 1B: Tester confirms that all confirmed orders placed by that account are displayed on the page.

Section 5I: Test Case #09 – View Order Statuses (Kitchen)

- Preconditions/Setup: Tester must be logged in as a Kitchen Staff employee.
- Requirement(s) Tested: View Kitchen Status
- Action 1: Tester activates the Orders navbar link.
- Verify 1A: Tester confirms that the order status page is displayed within the content pane.
- Verify 1B: Tester confirms that all confirmed but not yet completed orders are displayed (Kitchen Queue).

Section 5J: Test Case #10 – View Order Statuses (Wait)

- Preconditions/Setup: Tester must be logged in as a Wait Staff employee.
- Requirement(s) Tested: View Wait Status
- Action 1: Tester activates the Orders navbar link.
- Verify 1A: Tester confirms that the order status page is displayed within the content pane.
- Verify 1B: Tester confirms that all completed but not yet delivered orders are displayed (Wait Queue).

Section 5K: Test Case #11 – View Order Statuses (Manager)

- Preconditions/Setup: Tester must be logged in as a Manager.
- Requirement(s) Tested: View Universal Status
- Action 1: Tester activates the Orders navbar link.
- Verify 1A: Tester confirms that the order status page is displayed within the content pane.
- Verify 1B: Tester confirms that all confirmed but not yet delivered orders are displayed (Kitchen & Wait Queues), and are prompted for an order identification number to display any individual past or present order that exists in the database.
- Action 2: Tester enters a valid order identification number.
- Verify 2: Tester confirms that the associated order is displayed.

Section 5L: Test Case #12 – Generate Customer Reports

- Preconditions/Setup: N/A
- Requirement(s) Tested: View Customer Reports
- Action 1: Tester activates the Reports navbar link.
- Verify 1: Tester confirms that the Reports page is displayed within the content pane.
- Action 2: Tester selects the desired report.
- Verify 2: Tester confirms that the selected report is displayed on the page.

Section 5M: Test Case #13 – Generate Manager Reports

- Preconditions/Setup: Tester must be logged in as a Manager.
- Requirement(s) Tested: View Manager Reports
- Action 1: Tester activates the Reports navbar link.
- Verify 1: Tester confirms that the Reports page is displayed within the content pane.
- Action 2: Tester selects the desired report.
- Verify 2: Tester confirms that the selected report is displayed on the page.

Section 5N: Test Case #14 – Request Help

- Preconditions/Setup: Tester must have confirmed an order, and be logged in as a Customer.
- Requirement(s) Tested: Request Help
- Action 1: Tester activates the Request Help link.
- Action 2: Tester logs out, and then logs back in as a Wait Staff employee.
- Action 3: Tester activates the Orders navbar link.
- Verify 1: Tester confirms that the order is displayed on the order status page with a red background.

Section 5O: Test Case #15 – Bulk Data Import

- Preconditions/Setup: Tester must be logged in as a Manager.
- Requirement(s) Tested: Bulk Data Import/Export
- Action 1: Tester activates the V-Menu Settings link from the administrative toolbar.
- Action 2: Tester switches to the Bulk Data Import/Export tab.
- Action 3: Tester supplies a file name and path to upload the file on their local machine.
- Action 4: Tester clicks the Import Data button.
- Action 5: Tester switches back to the Settings tab.
- Verify 1: Tester confirms that the settings have been replaced with those from the supplied data file.

Section 5P: Test Case #16 – Bulk Data Export

- Preconditions/Setup: Tester must be logged in as a Manager.
- Requirement(s) Tested: Bulk Data Import/Export
- Action 1: Tester activates the V-Menu Settings link from the administrative toolbar.
- Action 2: Tester switches to the Bulk Data Import/Export tab.
- Action 3: Tester supplies a file name and path to download the file on their local machine.
- Action 4: Tester clicks the Export Data button.
- Verify 1: Tester confirms that the data file has been saved to the specified location on their local machine.

Section 6: Assumptions and Risks

Section 6A: Assumptions

- We assume each client device will be accessing our application using a supported web browser. We plan on supporting the latest stable versions of Chrome, Safari, Internet Explorer, and Edge. We have chosen these specific browsers because they are the defaults for most mobile devices. Chrome is the default browser on Android-based phones/tablets (and their Chrome OS cousins). Safari is the default browser on iPhones/iPads (and their Mac OS cousins). Internet Explorer and Edge are the defaults on Windows PCs and Microsoft phones/tablets respectively. Therefore, we are confident that the application will be accessible to virtually all users regardless of the make and model of their device.
- We assume that each restaurant owner that chooses to use our product will sign up for an account with the OpenMenu system. This is not strictly necessary, it will allow them to search for and use other menus as a starting point, so that they don't have to create a menu completely from scratch (though they will also be able to do that, if they want).
- We assume that each restaurant owner that chooses to use our product will sign up for an account with Google to use the Google Analytics, Google AdWords, and ReCAPTCHA services. These are not strictly necessary, but the services are legitimately useful. Every owner should seriously consider using them.

Section 6B: Risks

- Matthew, our team leader, is the only member of the team with experience using Drupal. This presents a challenge due to the learning curve involved in learning a new technology. We have been and will continue to put a great deal of effort into overcoming this risk by meeting regularly and having him guide our development. We also considered WordPress, but the team's experience with that platform wasn't much better; the only team members with experience using WordPress are Matthew and Slaven.
- It is entirely possible that there will not be near as much demand for this service as we expected, and if that is the case, then it is unlikely that our application will be adopted in any production environments.