Test Plan

System testing

Register Primary Guardian

Test Case Specification

1. Test case #1
2. Test item(s): Create blank primary guardian with no fields except first name
3. Precondition: First name only contains alphabets (A-Z,a-z), an existing center
4. Input required: First name, Center
5. Test procedure:
    Type in first name of participant
    Click submit
6. Output expected:
    Added to database successfully
    Shows the firstname field as the name that is inputted in the index screen
7. Actual test result:
    Added successfully to database
    Shows the Firstname displayed on the index screen

1. Test case #2
2. Test item(s): Create primary guardian with no children or secondary guardian
3. Precondition: First name and last name only contains alphabets (A-Z,a-z), Phone contains only numeric digits(0-9), no special symbols for every field except email
4. Input required: First name, Last name, email, phone, postal code, language, country, allergy, center
5. Test procedure:
    Type in all of the input specified above
    Click submit
6. Output expected:
    Added to database successfully
    Shows correct data that was inputted by the users in the index screen
7. Actual test result:
    Added successfully to database
    Shows correct data that was inputted by the users in the index screen

1. Test case #3
2. Test item(s): Create primary guardian with one secondary guardian but no children

3. Precondition: First name and last name only contains alphabets (A-Z,a-z), Phone contains only numeric digits(0-9), no special symbols for every field except email, first name of secondary guardian
4. Input required: First name, Last name, email, phone, postal code, language, country, allergy, center
5. Test procedure:
    Type in all of the input specified above
    Click submit
6. Output expected:
    Added to database successfully
    Shows correct data that was inputted by the users in the index screen
7. Actual test result:
    Added successfully to database
    Shows correct data that was inputted by the users in the index screen

1. Test case #4
2. Test item(s): Create primary guardian with multiple secondary guardian but no children
3. Precondition: First name and last name only contains alphabets (A-Z,a-z), Phone contains only numeric digits(0-9), no special symbols for every field except email, multiple names of secondary guardians
4. Input required: First name, Last name, email, phone, postal code, language, country, allergy, center
5. Test procedure:
    Type in all of the input specified above
    Click submit
6. Output expected:
    Added to database successfully
    Shows correct data that was inputted by the users in the index screen
7. Actual test result:
    Added successfully to database
    Shows correct data that was inputted by the users in the index screen

1. Test case #5
2. Test item(s): Create primary guardian with one children but no secondary guardian
3. Precondition: First name and last name only contains alphabets (A-Z,a-z), Phone contains only numeric digits(0-9), no special symbols for every field except email, first name of child
4. Input required: First name, Last name, email, phone, postal code, language, country, allergy, center
5. Test procedure:
    Type in all of the input specified above
    Click submit
6. Output expected:
    Added to database successfully

Shows correct data that was inputted by the users in the index screen

7. Actual test result:
    Added successfully to database
    Shows correct data that was inputted by the users in the index screen

1. Test case #6
2. Test item(s): Create primary guardian with multiple children but no secondary guardian
3. Precondition: First name and last name only contains alphabets (A-Z,a-z), Phone contains only numeric digits(0-9), no special symbols for every field except email, first name of multiple children
4. Input required: First name, Last name, email, phone, postal code, language, country, allergy, center
5. Test procedure:
    Type in all of the input specified above
    Click submit
6. Output expected:
    Added to database successfully
    Shows correct data that was inputted by the users in the index screen
7. Actual test result:
    Added successfully to database
    Shows correct data that was inputted by the users in the index screen

1. Test case #7
2. Test item(s): Create primary guardian with one children and one secondary guardian
3. Precondition: First name and last name only contains alphabets (A-Z,a-z), Phone contains only numeric digits(0-9), no special symbols for every field except email, first name of secondary guardian and child
4. Input required: First name, Last name, email, phone, postal code, language, country, allergy, center
5. Test procedure:
    Type in all of the input specified above
    Click submit
6. Output expected:
    Added to database successfully
    Shows correct data that was inputted by the users in the index screen
7. Actual test result:
    Added successfully to database
    Shows correct data that was inputted by the users in the index screen

1. Test case #8
2. Test item(s): Create primary guardian with multiple children and secondary guardian
3. Precondition: First name and last name only contains alphabets (A-Z,a-z), Phone contains only numeric digits(0-9), no special symbols for every field except email

4. Input required: First name, Last name, email, phone, postal code, language, country, allergy, center
5. Test procedure:
    Type in all of the input specified above
    Click submit
6. Output expected:
    Added to database successfully
    Shows correct data that was inputted by the users in the index screen
7. Actual test result:
    Added successfully to database
    Shows correct data that was inputted by the users in the index screen

Create Drop-in Session

1. Test case #1
2. Test item(s): Create an drop-in session
3. Precondition: A valid date and logged into a staff account
4. Input required: A valid date
5. Test procedure:
    Type in a valid date
    Click submit
6. Output expected:
    Added to database successfully
    Shows correct data that was inputted by the users in the index screen
7. Actual test result:
    Added successfully to database
    Shows correct data that was inputted by the users in the index screen

Drop-in Session Sign In Sheet

Create Resources

Create Referrals

Create Library Items

Create Centers

Generate Report

1. Test case #1
2. Test item(s): Generate report
3. Precondition: Start date and End date are specified using the format: dd/mm/yyyy
4. Input required: Start date and End date
5. Test procedure:
    Type in Start date and End date
    Click generate report
6. Output expected:
    Three tables:
    1. Contained information of number of new parents, children, visit secession for each center.
    2. Contained information of how many different languages are the parents speaking in each center.
    3. Contained the culture background information of parents in each center
7. Actual test result:
    Three tables with correct information displayed in the generate view.

Integration testing

Unit testing

The unit test project documents the unit test plan. The project uses the unit test framework built into Visual Studio 2010 referred to simply as the Microsoft Unit Test Framework. The main pieces that need to be unit tested are the of “Models” and the “Controllers” of the Model View Controller (MVC) architecture. The unit test files can be found in their respective folders in the Test project.

The unit test names will have a verbose naming scheme such that a reader of the unit test can understand the purpose of the test. For example inside a file called HomeControllerTest.cs in the Controllers folder of the test project, there may be a test called Index_NotLoggedIn_ShouldRedirectToLogin. A reader of this test can expect the test to verify that the controller will redirect a user that is not logged in to the login page.

As the application makes extensive use of the database, the database access must be decoupled from the controllers to allow test runs to complete quickly. To accomplish this, the main project makes use of a runtime dependency injection framework called Ninject that injects Interface dependencies into the controllers.

The interfaces and concrete classes that implement these interfaces can be found in the Repositories folder in the project. For example, there is the interface ICenterRepository.cs and the concrete class CenterRepository.cs. There is one more interface that wraps all the repositories into one class called IRepositoryService.cs and the concrete class is called DbRepositoryService.cs. Every controller accesses the database (repository) through IRepositoryService.cs. A similar method is used to inject user authentication dependencies that call static methods to authenticate users. These layers that depend on an actual data repository such as a database are not unit tested but are verified through system and integration testing.

Since data access depends on an interface and not a database, the data repositories’ interfaces can be mocked and passed into the controllers through their constructors. The mocking framework chosen is Moq. The controllers are instantiated with these mocks and results of Controller Actions (methods) are verified with the unit test framework.

Model validation is much simpler using the model data annotations and a method defined in the Controller classes, ValidateModel. This sets the ModelState of a controller and is verified to be valid.

Asserts are used throughout the unit tests. After a test run, the Microsoft Unit Test Framework produces a test result summary and failed tests are marked. If an assert failed, the framework notes the expected value and the actual value. The test also notes If the test failed for another reason, for example, a null pointer exception. You can find the exact line of code that produced the error.