

Team 6: Evan Yang, Eura Shin, Matthew Phelan, Trent Woods

CS 499-001 Fall 2019

December 4th, 2019

Word Count:

Evan: 352

Eura: 248

Matthew: 5,637

Trent: 578

1. Implementation

1.1 Source Code

Link to our GitHub repository: <https://github.com/evanfyang/death-scene-investigation.git>

The purpose of this application is to create and edit death scene investigation forms within an iOS application. The data from the form is then securely submitted to our backend Linux server. The following figure is shown as an example of how this data is transmitted to the server. The first photo is a code snippet from our “authenticate.php” file which allows us to ensure that a correct username and password combination is entered when attempting to access the app.

```
$user=$_POST['email'];
$pass=$_POST['password'];

// Check to make sure the username actually exists
$sql = "SELECT Password FROM Investigator WHERE Email='".$user."'";
$result = mysqli_query($conn, $sql) or die(mysqli_error($conn));
$count = mysqli_num_rows($result);

if ($count == 1) {
    // Check to make sure the password and username match
    $row = mysqli_fetch_row($result);
    if($row[0] == $pass) {
        echo "success!";
    }
    else { // Password does not match
        echo "Incorrect password";
    }
}
else
{
    // Invalid
    echo "No username under ". $user. " found";
}

$conn->close();
```

Figure 1. Implementation within our LoginViewController.swift file.

The next figure, Figure 2, is an example of an HTTPS request from the frontend of our system. Once a user enters data into the form and selects save, an HTTPS request similar to Figure 2 is sent to the backend, where it is handled by the appropriate php file, checked for errors, and if it contains compatible data is saved to our database. This is essentially how our system works. The user interacts with our user interface, these interactions trigger an HTTPS request to our backend, and the data is saved to our database. If data needs to be retrieved, an HTTPS request is sent to the backend, the database is queried, and the data is displayed to the user via the UI.

```
Alamofire.request(url, method: .post, parameters: params).validate().responseString {
    response in

    if let result = response.result.value {
        //if there is no error
        if(result.contains("success")){

            self.goToHomePage()
            self.UserNameTextField.text = ""
            self.UserPassWordTextField.text = ""

        }
        else {
            // Display an alert if an error and database insert didn't work
            DispatchQueue.main.async {
                let alert = UIAlertController(title: "Incorrect Username/Password",
                    message: "Please try again.", preferredStyle: .alert)

                alert.addAction(UIAlertAction(title: "OK", style: .cancel, handler:nil))

                self.present(alert, animated:true, completion: nil)
            }
        }
    }
}
```

Figure 2. Example of an HTTPS request from our application to the backend.

1.2 QA review results

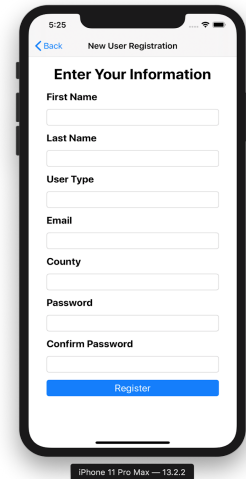
We held a team meeting and reviewed our progress with the application. The following are some QA items from our meeting:

1. When cycling quickly through form sections quickly, the application becomes quite laggy and may affect user experience. This is a small, temporary performance hit and should not affect overall performance.
2. We did not complete the version control functionality of our system, so that requirement remains incomplete.
3. Some pages in the iOS application (the register page, for example) do not have back buttons and have to be swiped down to return to a previous page. This may not be intuitive to some users.
4. The security clearance system of our application has not been tested extensively and needs more refinement to work properly.
5. We still need to publish the application to the App store to see how it will perform when downloaded from the Store to a client's device.

1.3 User's Manual

1.3.1 User does not have an account

If the user is not registered within the system, they will have to request an account. The user will click “New User? Register Here!” on the main homepage. This takes you to a registration page where the user enters his/her first name, last name, user type, email, county, and password. The county is needed because once the account is requested, personnel within the KY Coroner's Office will know where to send the account request with the aforementioned credentials. Only once approved by the KY Coroner's Office will the user be able to access and use the app.

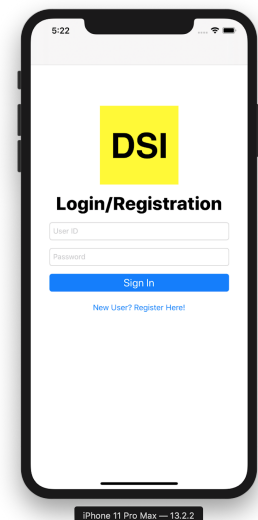


1.3.2 User Has a Registered Account

The user is able to login to the application using the credentials that they provided during registration. The user can then choose to start a new death scene investigation form, view an existing form that is currently in progress of being filled out or view a form that has been published. The preexisting and published forms are listed in tabular format on the home screen under their respective tabs. Starting a new form pulls up a modal containing the new empty form for the user to fill out. Once the user begins filling out the form, they can either continue filling out the form until completion or partially filling out the form and then saving what they have entered. Once they save a form, it is then viewable on the home screen under either the new forms tab or the published forms tab, depending on if the user saved the form before completion or saved the form for publication.

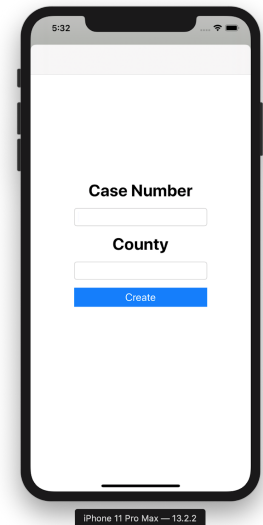
1.3.3 User Login

Once the user's registration is accepted, they are cleared to login and interact with the application. Enter the email address and password you registered with, and the application will accept these credentials (if they are valid) and take you to the home page.



1.3.4 Starting a New Form

On the home page, press the plus sign in the upper right corner. You will be directed to this screen where you must enter the case number for the case you are working on and your county. The county field has an autofill feature, so feel free to type the first few letters of your county and the rest should fill in for you if you select it.



5:32

Case Number

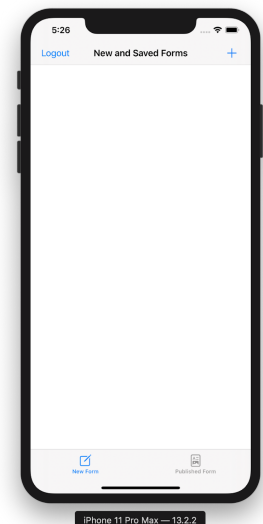
County

Create

iPhone 11 Pro Max — 13.2.2

1.3.5 Selecting a Form in Progress

If you have a form that is not complete and you wish to add more to it, simply select it from the list on the home page (all active forms will be listed on the home page for selection). After that, you are free to enter new information or edit previously edited information while the form is not marked read-only.



5:26

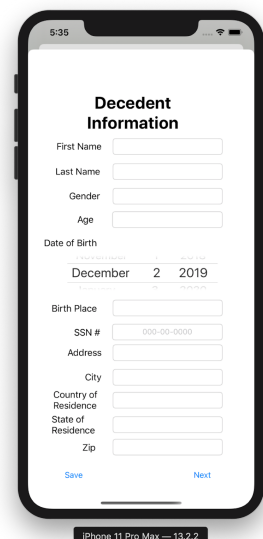
Logout New and Saved Forms +

New Form Published Form

iPhone 11 Pro Max — 13.2.2

1.3.6 Form Navigation and Entry

There are many sections to the application form, similar to the physical death scene investigation form. To navigate between sections, tap next to go to a different section. To enter data, tap what part of the section you want to enter and either type, scroll to, or select your entry (depending on the type of entry).



5:35

Decedent Information

First Name

Last Name

Gender

Age

Date of Birth

December 2 2019

Birth Place

SSN # 000-00-0000

Address

City

Country of Residence

State of Residence

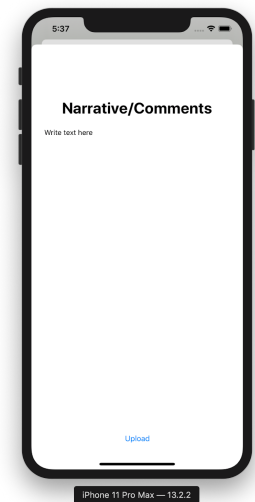
Zip

Save Next

iPhone 11 Pro Max — 13.2.2

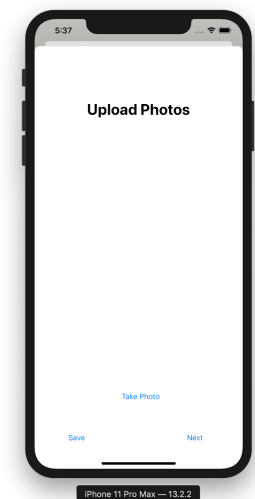
1.3.7 Saving Forms

There are two ways to save your form. The first method is to press “save” on each form section. This will save that individual section to the database. The second method is to fill out the entire form and, once at the last page, press “upload”. This will save all sections of the form to the database.



1.3.8 Uploading Photos

The form allows up to five photographs to be uploaded while investigating. Press “Take Photo” to navigate to your device’s camera and then take a photograph. The photo will then be added to the form and can be saved using the normal saving functionality.



1.4 Administrator’s Manual

Since the data for our application and our backend APIs are hosted on a Linux server in MySQL, the administrator should have access to an SSH client to manage data being stored on the server as well as granting permissions to the appropriate users that register with the application. To access the server, type “ssh your_linkblue_username@statsqltest.as.uky.edu” and enter your password. After that, type “cd var/www/html” to switch to the directory containing all of our backend files. If a specific use case requires an additional attribute to be added in the database, the administrator would have to edit the SQL script that is used to generate our database schema in MySQL to include the new attribute. Subsequently, the administrator would also need to edit the PHP APIs to reflect this change. Other updates to the database or the backend APIs will be carried out in a similar fashion. To uninstall the backend APIs, the administrator can issue the

command 'rm *.php' in the directory containing the PHP files. To uninstall our database, issue the following command in MySQL: 'DROP DATABASE deathrecapp_schema'.

```
mjph225@statsqltest:/var/www/html$ ls
authenticate.php      edit_DSI.php          Insert.php             register.php
createTable1.php      edit_incident_info.php latest_version.php      reset.php
createTable.php        edit_investigation.php load_open_forms.php    retrieve.php
delete.php             edit_narrative_comments.php load_published_forms.php showTable.php
deleteTable.php        edit_next_of_kin.php  login.php              start_death_scene_investigation.php
droplogin.php          edit_pills_on_scene.php new.php                upload.php
edit_case_history.php  index.nginx-debian.html newUser.php
edit_decedent_info.php insert.php             note.txt
mjph225@statsqltest:/var/www/html$
```

Figure 3. All files housed in our backend server.

Due to the fact that this is an iOS application, a Mac computer will be needed to run and perform further development on the application. The app is currently being developed using Xcode version 10.3. Swift is the programming language that is used within Xcode. The current swift version is that is used within the app is version 5.1.

To add new frameworks to the application, the user needs to be familiar with CocoaPods. If a new framework is being implemented within the system, it must be added to the Podfile that can be viewed within Xcode as well as included in each file that requires the framework. The user must be sure to add all of the CocoaPod files to the git repository so all developers are working with the same frameworks.

Updates to the iOS mobile application will be released on the Apple App Store, and once an update is made available the user can directly download the released update. Uninstalling the iOS mobile application is simple: hold down on the application and press the 'x' icon that appears after holding down on the application.

2. Testing

2.1 Unit/API Test Plan

Test Plan ID: 1

Note: Our test plans (unit and acceptance) are based on many (some sections are not applicable to this project) of the sections stipulated in ANSI/IEEE 829-1988 Test Plan Structure.

2.1.1 Unit Test Plan Items and Introduction

Most of our unit testing deals with unit tests for our API and the functionality of our iOS application (note API testing is not pure unit testing, but we have included it in this section). For our API, we elected to use Postman (a collaborative API development platform) to send HTTP POST requests to our database. These tests involve testing all conditions for each php file we have. For example, to test our user authentication php file, we first register a user then we have three tests for authentication (to achieve 100% code coverage for that php file). One test with correct credentials, one for incorrect credentials, and one for missing credentials.

User interface testing is difficult to automate when trying to recreate hypothetical user actions, thus our test cases consist of manual entry and selection of UI elements. Our first tests are unit tests for localized UI elements like buttons and field entries, then the rest of our test cases deal with the app's integration with our database via POST requests. This is a core function of our system, so many of our tests are related to this.

2.1.2 Risks

Identifying and mitigating risks is crucial to delivering a working product. Testing is not devoid of risks, and we have identified some potential risks and mitigating factors for our testing process.

| Risks | |
|---|---|
| Risk | Mitigation |
| We plan to have all testing completed by Sunday, December 1st. What if delays cause us to miss this deadline? | Our coding assignment and presentation are not due until Wednesday, December 4th. We can use the buffer days between Sunday and Wednesday if need be. |

| | |
|--|---|
| Original requirements change or new requirements are added. | We are nearing the end of this assignment and semester and we have met with Dr. Brown several times. All requirements have been finalized, and adding or changing a major requirement is not feasible. |
| Many tests fail and we get bogged down fixing code. | We have been unit testing major portions of our code incrementally throughout the semester. Any failed test cases now are likely to be integration errors that are relatively easy to fix (compared to core code error that may be difficult to fix). |
| The team does not communicate effectively and some core functions go untested. | Each team member is responsible for testing their code contributions. We are all aware of this and it is unlikely that completed code goes untested since we have tested incrementally. |
| Our planned test cases are too complex to be reasonably implemented and run before the deadline. | Our test cases consist mostly of small unit and integration tests for each file or section of our system, thus limiting the overall complexity of each test case. |

2.1.3 Features to Be Tested (Unit Tests/API Tests)

This section specifies what is to be tested from a user's viewpoint of the system. This applies to our unit testing only. Our deliverable for acceptance testing will contain more integration-oriented tests.

| Feature | Trace to Functional Requirement (FR ID) |
|---|---|
| User Login Page | 6.2.1.2 |
| User Registration Page | N/A |
| Locally Storing Data | 6.2.2.2 |
| Form Navigation Between Sections | 6.2.2.4 |
| Photo Upload to iOS Application | 6.2.2.6 |
| Saving Form Data to the Backend (API tests) | N/A |
| Autocomplete Feature | 6.2.2.5 |

2.1.4 Features not to be Tested (Out of Scope for this Test Plan)

This section specifies what not to be tested either because the feature is out of scope of this test plan or the feature will not be included in this particular software release. Examples of this include integration or acceptance tests that will be included in another test plan or features that were listed in our requirements that will be implemented and released in the future and tested before release.

| Feature | Trace to Functional Requirement (FR ID) |
|---|---|
| Version Control | 6.2.2.3 |
| Security Clearances | 6.2.1.2 |
| Data Encryption | 6.2.1.3 |
| SQL Relationship Testing | N/A |
| Editing Permissions for Different Investigators | 6.2.1.1 |
| All Features Related to Application Integration with the Database | N/A |

2.1.5 Test Approach

This section provides an analysis of our requirements, scenarios and expected outcomes, and guidelines for our test cases.

2.1.5.1 Requirements Analysis

The requirements can be found in our Software Requirements Specification document, sections 6 and 7. They generally cover each aspect of our system, but some inconsistencies and gaps in coverage exist. Some of our requirements are too broad for certain test cases, so they cannot be linked to the very specific unit tests (hence the N/A for some of our test case traces in the tables above). These coverage inconsistencies are sparse and should not compromise the effectiveness of our test cases.

The requirements can be broken into three types: application, backend, and integration requirements. Application requirements deal solely with our iOS application functions, backend requirements focus on our backend functions (php and sql related functions), and integration requirements tie the two together. Our test cases reflect these requirements. Our unit testing will

focus on testing our application and backend requirements, while our acceptance testing and integration testing will focus on testing the larger scope integration requirements. This test plan covers application and backend requirements.

Our application requirements will mostly be tested using manual UI testing, and our backend requirements will be tested using API testing with the help of Postman. Postman allows us to automate our API testing by sending HTTP POST requests to our backend to simulate storing data from the iOS application. It is a collaborative platform for API testing which allows our team to access shared test cases and edit them concurrently.

2.1.5.2 Test Scenarios and Expected Outcomes

| Scenario | Expected Outcome |
|--|--|
| User logs in (unit testing iOS login page, not testing integration and backend query). | The system prompts for and allows the user to type their credentials and click submit. |
| User registers. | The system provides a clickable registration page button that redirects the user to our registration page. |
| User selects save when the device is not connected to the internet. | The application saves all data locally. |
| User selects “start new form” button. | The application redirects them to a new form and prompts them to begin filling out form data. |
| User selects “next” on form navigation. | The application redirects to the next page of the form. |
| User selects “previous” on form navigation. | The application redirects to the previous page of the form. |
| User selects “upload photos”. | The application allows the user to upload up to five photos to be stored locally and in the database. |
| The application sends http POST request to save data to the backend. | The appropriate php file is called and the data is saved to our database. |
| User begins entering form data. | The application autocompletes words using the relevant counties or toxicology terminology provided by Dr. Brown. |

2.1.6 Pass/Fail Criteria

This section is a brief discussion of what constitutes a passing test case and what must occur for this test plan to be executed successfully (the overall number of passing test cases).

Our unit tests and API tests pass if the result of executing the test case reflects what we expect and what a user would expect. If an API test to save data to the database does just that and all data is saved correctly and no errors occur, it passes. If a UI test results in an expected UI action (like the form allowing a user to enter data into the fields), then it passes. Conversely, if a valid API request is rejected or accepted data is not saved to the database, it fails and if a UI element does not perform as expected (a disabled button or form that should be enabled, for example) that test fails.

For this test plan to be effective, all unit tests must pass. If a unit test does not pass for our system, it is relatively simple to fix. Thus, all tests should pass eventually. This only includes all executed tests--some features may be out of scope or not implemented. It is not feasible to test these features or count them as failed tests.

2.1.7 Suspension Criteria

It is important to determine when to suspend all or a portion of testing when abnormalities occur or inconsistencies are discovered.

Suspension Criteria for our unit and API testing:

1. When a bug is discovered that affects multiple units and hinders further testing of other units or API functions.
2. If a testing system or environment (like Postman) stops working.
3. If 30% of our test cases fail, we will suspend all testing and reevaluate our system.
4. A change in requirements is requested by the client.
5. Problems with the software hosting our system (our backend server or iOS app) occur.

2.1.8 Test Deliverables

The following artifacts will be delivered as a result of testing (all will be attached in section 2.3 of this document):

1. Test Plan
2. Test Cases
3. Test Administrator List (included with test cases)
4. Test Results (included with test cases)
5. QA Review of Test Plans and Cases

2.2 Acceptance Test Plan

Test Plan ID: 2

2.2.1 Acceptance Test Plan Items and Introduction

Once we have completed our unit, system, and integration testing, we will perform acceptance testing of our system to ensure it complies with our requirements. Beta testing has been requested by our client, so that is our last form of acceptance testing to be performed. Before we test our application with coroners, however, we will perform other acceptance tests mainly through black box testing.

While our unit and API tests use a combination of manual UI testing and automated API testing via Postman, our acceptance testing will be purely through our UI. Our acceptance testing will include testing from our users via beta tests, while we will perform black box testing. Since our iOS application is the face of our system and the only element clients will be interacting with, it is important that we model our acceptance testing to reflect user action. Thus, the platform on which we will be performing our acceptance tests will be our application locally downloaded on our iOS devices and on client iOS devices (before it is published). This will reflect actual download and usage from the App Store.

2.2.2 Risks

The risks associated with acceptance testing differ from our unit testing risks. They are less numerous than those associated with unit testing, but their cost is much greater due to the fact that acceptance testing generally occurs later in the software development lifecycle and it is larger in scope than unit testing.

| Risks | |
|--|--|
| Risk | Mitigation |
| None of us have downloaded an app still in production off Xcode, which requires a developer's license. This could prevent us from performing the majority of our acceptance tests until it is fixed. | We will attempt to download our production app and acquire a developer's license far in advance of the project deadline so we can fix any issues that might arise. |
| We reach the deadline before we are able to travel to Louisville and perform beta testing with our clients in an operational setting. | We will hopefully be able to travel to Louisville before the deadline. However, we realize that not everything can be accomplished for the capstone, especially |

| | |
|--|---|
| | travelling to Louisville and spending time teaching the functionality of our application. |
|--|---|

2.2.3 Features to Be Tested

These features may have similar titles to some features included in the unit testing features, but they apply to the entire collection of functions associated with that feature instead of one unit like a UI element or php file. These acceptance tests connect our frontend to the backend and test what actually will occur when users interact with our application.

| Feature | Trace to Functional Requirement (FR ID) |
|--|---|
| User Login Authentication Successful | 6.2.1.2 |
| User Registration | N/A |
| Form Data Stores Locally and Backs Up to Database | 6.2.2.2 |
| Forms are Accurately Marked Read-Only for Individuals without Edit Permissions | 6.2.1.1 |
| Version Control Works Properly | 6.2.2.3 |
| Non-Cleared Individuals are Restricted from Viewing Certain Forms | 6.2.1.2 |
| Each Form Section can be Saved Individually | N/A |

2.2.4 Features not to be Tested (Out of Scope for this Test Plan)

All the features of our system will be tested by our acceptance tests. Even if they have already been tested, they must be tested again to ensure their integration with other features works properly. Also, once the application is released, all features will be eventually used by our clients, so our acceptance tests should reflect this to uncover any potential bugs that could exist.

2.2.5 Test Approach

This section provides an analysis of our requirements, scenarios and expected outcomes, and guidelines for our test cases.

2.2.5.1 Requirements Analysis

The requirements can be found in our Software Requirements Specification document, sections 6 and 7. Similar to the first test plan, these requirements generally cover each part of our system, but there are a few inconsistencies--you cannot plan for everything. For this test plan, we are

looking to test all features used by customers and we are ensuring all requirements are satisfied. If you recall from the first section, I defined three types for our requirements: application, backend, and integration requirements. This test plan and its accompanying tests will mainly cover integration requirements, thus testing the integration of our application and backend requirements--what our system relies on.

2.2.5.2 Test Scenarios and Expected Outcomes

| Scenario | Expected Outcome |
|------------------------------|---|
| User logs in. | System prompts for user credentials. Once entered and submit is clicked, the application calls the backend to verify credentials and accepts or rejects the user based on this. |
| User registers. | System prompts user to enter required registration fields. Once entered and submit is pressed, the system emails Dr. Brown's office and they accept or reject the registration. Upon acceptance, the user is stored in the database along with their clearance. |
| User clicks save. | If the device is connected to the internet, all form data will be saved to our database. If the device is not connected, all data will be saved locally and backed up once a connection is reestablished. |
| User types form information. | Form accepts all user input and appropriately autofills complex toxicology terms. The user also has the option to save each form structure. |
| User uploads photos. | The application limits user input to five photographs and saves all photographs locally if there is no internet connection or it saves to the database with an internet connection. |

2.2.6 Pass/Fail Criteria

With acceptance testing, it is critical that all test cases pass. The pass/fail criteria for our acceptance tests are simple: does the function perform to our customers specifications and expectations? If it does, the test case for that function passes. If not, it fails and we talk with our client to come up with solutions or we refactor our test cases.

2.2.7 Suspension Criteria

It is especially important to determine when to suspend acceptance testing to correct any pertinent mistakes. Acceptance testing often involves beta testing, and knowing when to temporarily suspend testing involving clients is important to maintaining developer-client trust. If you cannot admit your mistakes and fix them promptly, your final application will not meet the specifications.

Suspension Criteria for our unit and API testing:

1. If any acceptance test fails, we will suspend testing for that section and look at fixing the faulty code.
2. A change in requirements occurs (unlikely since we are late in the development process).
3. If multiple, integrated test cases fail we will suspend all affected portions of testing and concentrate on fixing relevant code.

2.2.8 Test Deliverables

The following artifacts will be delivered as a result of testing (all will be attached in section 2.3 of this document):

1. Test Plan
2. Test Cases
3. Test Administrator List (included with test cases)
4. Test Results (included with test cases)
5. QA Review of Test Plans and Cases

2.3 Unit and Acceptance Testing Deliverables

2.3.1 Unit/API Test Cases and Results

| ID | Type | Description | Test Steps | Expected Results | Status | Test Admin |
|-----|------|---|--|---|------------|------------|
| TC1 | Unit | Registering a new user on the application | 1. Click register user button 2. Enter the required fields 3. Click register | 1. The system accepts any user input into the required fields 2. The register button is clickable 3. Once register is clicked, an HTTP request is sent | Passed | Eura |
| TC2 | Unit | User login | 1. Navigate to user login page 2. Enter any information into the username and password fields 3. Click login | 1. The application redirects user to the login page when the user selects that tab 2. The application allows the user to enter credentials for the username and password field 3. When the login button is clicked, an HTTP POST request is sent with the credentials to be validated | Passed | Trent |
| TC3 | Unit | Saving Data Locally | 1. Turn of WiFi and cellular data on the device 2. Select "Save Data" on the form | 1. The application provides a clickable "Save Data" button 2. Once the button is clicked, all form data is saved locally | Not tested | |
| TC4 | Unit | Upload Photos | 1. User selects "Upload Photos" near the end of the form | 1. Camera application on the respective device is pulled up. 2. Photos can be taken and uploaded to our app | Not tested | |
| TC5 | Unit | Start a new Form | 1. User selects "Start a New Form" on main storyboard | 1. New form is presented to the user 2. Data can be input | Passed | Trent |
| TC6 | Unit | Autocomplete feature | 1. User navigates to any active form 2. User enters data on any form section | 1. The application autocompletes user input based on the toxicology terms and counties provided by Dr. Brown | Passed | Trent |
| TC7 | Unit | Form Navigation | 1. User navigates to any active or inactive form | 1. The application, when next is selected, redirects the user to the next | Passed | Evan |

| | | | | | | |
|-------|-----|--------------------------------|---|---|--------|---------|
| | | | 2. They select next or previous | page on the form 2. The application, when previous is selected, redirects the user to the previous page of the form | | |
| TC8 | API | POST Register Success | 1. POST request sent via Postman containing an email, county, type of investigator, password, firstname, and lastname | 1. All information sent in the post request is accepted by our register.php file and saved correctly in the database | Passed | Matthew |
| TC9 | API | POST Register Existing | 1. POST request sent via Postman using register.php with credentials that already exist in the database | 1. register.php does not interact with the database 2. An error message stating "Error! This username is taken." is sent | Passed | Matthew |
| TC 10 | API | POST Authenticate Successful | 1. POST request sent via Postman to authenticate.php with a valid username and password | 1. authenticate.php accepts credentials 2. Returns a 200 status code and "Success!" | Passed | Matthew |
| TC 11 | API | POST Authenticate Incorrect | 1. POST request sent via Postman to authenticate.php with an invalid password | 1. authenticate.php rejects credentials 2. Returns a 200 status code and "Incorrect password message" | Passed | Matthew |
| TC 12 | API | POST Authenticate Not Found | 1. POST request sent via Postman to authenticate.php with an invalid username | 1. authenticate.php rejects credentials 2. Returns a 200 status code and "No username under (entered username) found" | Passed | Matthew |
| TC 13 | API | POST Start DSI | 1. POST request sent via Postman to edit_DSI.php with the required fields to be stored in the database for the Death Scene Investigation entity (the fields are too numerous to list) | 1. Data is accepted, 200 status code received 2. Data is stored correctly in the database | Passed | Matthew |
| TC 14 | API | POST Edit Case History | 1. POST request sent via Postman to edit_case_history.php with the required fields to be stored in the database for the Case History entity (the fields are too numerous to list) | 1. Data is accepted, 200 status code received 2. Data is stored correctly in the database | Passed | Matthew |
| TC 15 | API | POST Edit Incident Information | 1. POST request sent via Postman to edit_incident_info.php with the required fields to be stored in the database for the Incident | 1. Data is accepted, 200 status code received 2. Data is stored correctly in the database | Passed | Matthew |

| | | | | | | |
|-------|-----|--------------------------------|---|--|--------|---------|
| | | | Information entity (the fields are too numerous to list) | | | |
| TC 16 | API | POST Edit Next of Kin | 1. POST request sent via Postman to edit_next_of_kin.php with the required fields to be stored in the database for the Next of Kin entity (the fields are too numerous to list) | 1. Data is accepted, 200 status code received 2. Data is stored correctly in the database | Passed | Matthew |
| TC 17 | API | POST Edit Investigation | 1. POST request sent via Postman to edit_investigation.php with the required fields to be stored in the database for the Investigation entity (the fields are too numerous to list) | 1. Data is accepted, 200 status code received 2. Data is stored correctly in the database | Passed | Matthew |
| TC 18 | API | POST Edit Decedent Information | 1. POST request sent via Postman to edit_decedent_info.php with the required fields to be stored in the database for the Decedent Information entity (the fields are too numerous to list) | 1. Data is accepted, 200 status code received 2. Data is stored correctly in the database | Passed | Matthew |
| TC 19 | API | POST Edit Narrative Comments | 1. POST request sent via Postman to edit_narrative_comments.php with the required fields to be stored in the database for the Narrative Comments entity (the fields are too numerous to list) | 1. Data is accepted, 200 status code received 2. Data is stored correctly in the database | Passed | Matthew |
| TC 20 | API | POST Edit Pills On Scene | 1. POST request sent via Postman to edit_pills_on_scene.php with the required fields to be stored in the database for the Pills On Scene entity (the fields are too numerous to list) | 1. Data is accepted, 200 status code received 2. Data is stored correctly in the database | Passed | Matthew |

2.3.2 Acceptance Cases and Results

Note: these may appear similar to the previous test cases for unit testing. However, these test cases cover the integration of multiple functions instead of specific UI or backend functions. Also, the internal systems are mentioned for black box testing but these are not known during the test case, they are just included with the expected outcomes.

| ID | Type | Description | Test Steps | Expected Results | Status | Test Admin |
|-------|-----------|------------------|---|--|------------|------------|
| TC 21 | Black Box | Register | 1. Navigate to registration page 2. Enter all required registration fields 3. Press register user | 1. The form allows the user to enter all fields and click register. 2. Once register is clicked, the form queries the backend. 3. Dr. Brown's office is emailed if the user is not already registered and they accept or deny the individual who registered. 4. If accepted, the user is notified and their credentials are stored in the database. 5. Upon acceptance, attempt to login with valid credentials. The user should be able to login. | Not Tested | |
| TC 22 | Black Box | Login | 1. Navigate to the login page. 2. Enter login credentials. 3. Press login | 1. The backend is queried. 2. If credentials are found, the user is allowed to register. 3. If no credentials are found, the user is not allowed to login and the appropriate message is displayed. | Not Tested | |
| TC 23 | Black Box | Saving Form Data | 1. Enter data into form 2. Press save | 1. If there is no connection, data is saved locally. 2. Else, the backend is queried and the data is saved to our database via an HTTP POST request to our php files. 3. All data is saved correctly and is accessible | Not Tested | |
| TC 24 | Black Box | Start a New Form | 1. Press start new form 2. Enter requested information to begin form | 1. The new form is saved as an entity in the database if there is an internet connection. | Not Tested | |

| | | | | | | |
|-------|-----------|------------------------------------|--|--|------------|--|
| | | | 3. Press enter | 2. The new form is saved correctly and is accessible if the user exits the application and comes back. | | |
| TC 25 | Black Box | Edit and Save Case History | 1. Navigate to the case history section of any editable investigation. 2. Edit all fields 3. Save that section | 1. All edited fields should be saved to the database via an HTTP POST request sent to edit_case_history.php. 2. Data should be saved correctly and accessible. 3. If the user exits the form and returns later, the data should be loaded from the database. | Not Tested | |
| TC 26 | Black Box | Edit and Save Incident Information | 1. Navigate to the incident information section of any editable investigation. 2. Edit all fields 3. Save that section | 1. All edited fields should be saved to the database via an HTTP POST request sent to edit_incident_information.php. 2. Data should be saved correctly and accessible. 3. If the user exits the form and returns later, the data should be loaded from the database. | Not Tested | |
| TC 27 | Black Box | Edit and Save Next of Kin | 1. Navigate to the next of kin section of any editable investigation. 2. Edit all fields 3. Save that section | 1. All edited fields should be saved to the database via an HTTP POST request sent to edit_next_of_kin.php. 2. Data should be saved correctly and accessible. 3. If the user exits the form and returns later, the data should be loaded from the database. | Not Tested | |
| TC 28 | Black Box | Edit And Save Investigation | 1. Navigate to the investigation section of any editable investigation. 2. Edit all fields 3. Save that section | 1. All edited fields should be saved to the database via an HTTP POST request sent to edit_investigation.php. 2. Data should be saved correctly and accessible. 3. If the user exits the form and returns later, the data should be loaded from the database. | Not Tested | |

| | | | | | | |
|-------|-----------|------------------------------------|--|---|------------|--|
| TC 29 | Black Box | Edit and Save Decedent Information | <ol style="list-style-type: none"> 1. Navigate to the decedent information section of any editable investigation. 2. Edit all fields 3. Save that section | <ol style="list-style-type: none"> 1. All edited fields should be saved to the database via an HTTP POST request sent to <code>edit_decedent_info.php</code>. 2. Data should be saved correctly and accessible. 3. If the user exits the form and returns later, the data should be loaded from the database. | Not Tested | |
| TC 30 | Black Box | Edit and Save Narrative Comments | <ol style="list-style-type: none"> 1. Navigate to the narrative comments section of any editable investigation. 2. Edit all fields 3. Save that section | <ol style="list-style-type: none"> 1. All edited fields should be saved to the database via an HTTP POST request sent to <code>edit_narrative_comments.php</code>. 2. Data should be saved correctly and accessible. 3. If the user exits the form and returns later, the data should be loaded from the database. | Not Tested | |
| TC 31 | Black Box | Edit and Save Pills On Scene | <ol style="list-style-type: none"> 1. Navigate to the pills on scene section of any editable investigation. 2. Edit all fields 3. Save that section | <ol style="list-style-type: none"> 1. All edited fields should be saved to the database via an HTTP POST request sent to <code>edit_pills_on_scene.php</code>. 2. Data should be saved correctly and accessible. 3. If the user exits the form and returns later, the data should be loaded from the database. | Not Tested | |
| TC 32 | Beta | General Beta Testing | For brevity, this test case covers numerous beta tests. Repeat all above tests with client. | <ol style="list-style-type: none"> 1. Functions perform to what the client expects. They can register, log in, edit forms, and save forms to revisit later. Also, each client can only edit what they are cleared to edit/cleared to view. | Not Tested | |

2.3.3 QA Review Results

We held a QA review session to discuss both of our test plans and case collections. We incrementally analyzed each plan and case collection, looking for discrepancies including missing cases, insufficient code coverage, unattainable/unrealistic test plan criteria and cases, and vague descriptions of the testing process.

For test plan 1 (unit/API testing), we found the following items:

1. Our test plan effectively covers most units of our system.
2. Incrementally unit testing has proved to be an effective method for mitigating risk and improving our code.
3. Our test scenarios could be more specific. However, our test cases mostly make up for this as they are much more descriptive.
4. Our pass/fail criteria for the test plan could be more specific. It is difficult to determine what constitutes a passing or failing test case based on it.

Test plan 2:

1. The test plan could have more description for the test approach section. Some readers may not fully understand how we will be performing our acceptance testing after reading that section.
2. Our acceptance test cases cover every potential function our client might use.
3. Beta testing may be an unrealistic form of testing at this stage in our project (it is due in two weeks and beta testing takes a considerable amount of time).
4. Our suspension criteria may not cover all situations and ambiguity may exist on whether to suspend testing or not if one of these situations arises.
5. Some features and test cases are not linked to a specific requirement and may be difficult to judge as passing or failing.

3. Technical Metric Collection

1. Estimated story points:

User Stories: 29

1. As a death scene investigator, I need to be able to work on the investigation form offline and sync when online so that I can gather data wherever I need it, regardless of internet connection.

Acceptance Criteria:

- The application is fully functional when there is no connection is available.
- User should be able to view and work with current death scene form and have the data cached on the device until a connection is available.
- When a connection is available, the cached data is imported into the database.

Story Points: 5

2. As a death scene investigator, I need autocomplete on the app so that I can quickly fill out the form without having to type long toxicology terms.

Acceptance Criteria:

- The application can recognize common toxicology terms as the user types and displays a list of closely matching terms for the user to select to fill out a specific field.
- This feature should be implemented for the fields under the Toxicology section of the form.

Story Points: 8

3. As a death scene investigator, I need three levels of login security clearance so that access to sensitive data can be limited to cleared individuals.

Acceptance Criteria:

- The application should show or hide certain information based on the security clearance of the individual.
- The application should also restrict edit permissions for all forms depending on whether a third party or a death scene investigator is accessing the data.
- Death scene investigators should be allowed to access, edit, and view forms that they have created, whereas a third party such as a district attorney can only view forms that they have

requested access to.

4. As a death scene investigator, I need the Android and iPhone versions of the application to be identical so that there are no discrepancies in the collection of legal evidence.

Acceptance Criteria:

- The functionality and aesthetics of the application are identical on both versions.
- Both applications should use the same method and format when interfacing with the backend server.

Story Points: 8

5. As a death scene investigator, I need to have access to previous versions of the forms that I have submitted in case a mistake is made in its collection or revision.

Acceptance Criteria:

- Forms that have been submitted can be edited and resubmitted, if need be.
- The user should only be able to access and edit forms that they have submitted in the past.
- The backend database will update the data accordingly.

Story Points: 40

6. As a death scene investigator, I need to be able to upload pictures of the death scene and attach them to the corresponding form that I will be filling out for the death scene.

Acceptance Criteria:

- Both versions of the app (iOS and Android) will be able to successfully access the camera of the device.
- The camera functionality of the application should be able to capture images from the death scene and attach images to the form.
- The application should be able to upload the images corresponding to a specific form to the database.

Story Points: 20

7. As a death scene investigator, I need to ensure that all communication and connections between the mobile application and the database server is secure and that all data being transmitted is encrypted.

Acceptance Criteria:

- Industry-leading security standards will be implemented to ensure all sensitive data is protected. Security is built into the application rather than being an afterthought.
- All data transmitted between the mobile application and the database should be encrypted. This can be done with a mutual authentication scheme, where SSL certificates are exchanged between the application and the database server to ensure that the application is only interfacing with a trusted server (prevents man in the middle attacks).

Story Points: 13

8. As a death scene investigator, I need the application to be easily accessible and usable on different devices of the same platform (iPhones and iPads on iOS, for example).

Acceptance Criteria:

- The format and functionality of the app will be the same regardless of screen size, device specs, etc.
- The application should be able to scale to the appropriate screen size and ensure that all sections and elements of the death scene investigation form are included and are clearly visible on all devices.

Story Points: 5

2. Actual lines of code as implemented:

| Swift Files (Front-End) | Lines of Code |
|--------------------------------|---------------|
| AppDelegate.swift | 58 |
| CaseHistory.swift | 514 |
| CaseNumberController.swift | 203 |
| deathsceneInvestigation.swift | 170 |
| DecedentInformation.swift | 358 |
| HomePageTabBarController.swift | 18 |
| IncidentInformation.swift | 294 |

| | |
|---|-------------|
| Investigation.swift | 644 |
| LoginRegistrationNavigationController.swift | 17 |
| LoginViewController.swift | 194 |
| MDSIDocProperties.swift | 347 |
| NarrativeComments.swift | 545 |
| NewFormViewController.swift | 120 |
| NextofKin.swift | 155 |
| PendingFormViewController.swift | 39 |
| PillsOnScene.swift | 161 |
| PublishedViewController.swift | 36 |
| RegisterNewUserViewController.swift | 76 |
| RoundButton.swift | 31 |
| TableOfContentsTableViewController.swift | 39 |
| UIViewControllerExtensions.swift | 146 |
| UploadPhotos.swift | 117 |
| Total | 4282 |

| PHP Files (Back-End) | Lines of Code |
|------------------------|---------------|
| authenticate.php | 46 |
| delete.php | 39 |
| edit_case_history.php | 145 |
| edit_decedent_info.php | 74 |
| edit_DSI.php | 62 |

| | |
|-------------------------------------|-------------|
| edit_incident_info.php | 73 |
| edit_investigation.php | 138 |
| edit_narrative_comments.php | 40 |
| edit_next_of_kin.php | 48 |
| edit_pills_on_scene.php | 56 |
| latest_version.php | 46 |
| load_open_forms.php | 54 |
| load_published_forms.php | 54 |
| register.php | 57 |
| retrieve.php | 58 |
| start_death_scene_investigation.php | 109 |
| Total | 1099 |

3. McCabe cyclomatic complexity of each module:
 - a. authenticate.php: **4**
 - b. delete.php: **4**
 - c. edit_case_history.php: **3**
 - d. edit_decedent_info.php: **3**
 - e. edit_DSI.php: **3**
 - f. edit_incident_info.php: **3**
 - g. edit_investigation.php: **3**
 - h. edit_narrative_comments.php: **3**
 - i. edit_next_of_kin.php: **3**
 - j. edit_pills_on_scene.php: **3**
 - k. latest_version.php: **2**
 - l. Load_open_forms.php: **2**
 - m. Load_published_forms.php: **2**
 - n. Register.php: **4**
 - o. retrieve.php: **4**
 - p. start_death_scene_investigation.php: **4**
4. Complexity of the system: **3** weighted methods per class (in the object oriented Swift files)
5. Product size: **8** user stories, **20** acceptance test plans

6. Product effort (number of person hours):
 - a. Evan: **25**
 - b. Eura: **25**
 - c. Matthew: **25**
 - d. Trent: **25**
7. Defects: **4** (refer to section 2.3.3 QA Analysis)

4. Github wiki and Repository

Link to our Github wiki (includes our developer notes):

<https://github.com/evanfyang/death-scene-investigation/wiki>

Link to our Github Repository:

<https://github.com/evanfyang/death-scene-investigation.git>