**C868 – Software Capstone Project Summary**

**Task 2 – Section C**

|  |  |
| --- | --- |
| **Capstone Proposal Project Name:** | http://www.idevnews.com/views/images/uploads/general/wgu_logo.png  Dossier Management System |
| **Student Name:** | Phuc Thach |

**Table of Contents**

Contents

[Table of Contents 2](#_Toc93247833)

[Application Design and Testing 3](#_Toc93247834)

[Design Document 3](#_Toc93247835)

[Class Design 3](#_Toc93247836)

[UI Design 5](#_Toc93247837)

[Unit Test Plan 8](#_Toc93247838)

[Introduction 8](#_Toc93247839)

[Purpose 8](#_Toc93247840)

[Overview 8](#_Toc93247841)

[Test Plan 8](#_Toc93247842)

[Items 8](#_Toc93247843)

[Features 8](#_Toc93247844)

[Deliverables 8](#_Toc93247845)

[Tasks 8](#_Toc93247846)

[Needs 8](#_Toc93247847)

[Pass/Fail Criteria 8](#_Toc93247848)

[Specifications 9](#_Toc93247849)

[Procedures 9](#_Toc93247850)

[Results 10](#_Toc93247851)

[C4. Source Code 10](#_Toc93247852)

[C5. Link to Live Version 10](#_Toc93247853)

[References 11](#_Toc93247854)

# Application Design and Testing

# Design Document

## Class Design

The project uses Command Query Responsibility Segregation Design Pattern and SOLID Principles in its class design diagram. CQRS Design pattern maximizes the application performance, scalability, and security. The SOLID principles encourage developers to create more maintainable, understandable, and flexible software while reducing code complexity.

The SOLID Principle of object-oriented design is a set of five programming rules. If followed, the application is likely to be less rigidity, less fragility, less immobility, and less viscosity (Martin R.C, 2000).

* Single Responsibility – A class should only have one responsibility
* Open-closed Principle – A class should be open for extension but closed for modification
* Liskov Substitution – Subclasses should be substitutable for their base classes
* Interface Segregation – Many clients specific interfaces are better than one general-purpose interface
* Dependency Inversion Principle – Depend upon abstracts. Do not depend on concretions.

The mainstream approach to interacting with information systems is via the CRUD approach (create new records, read records, update existing records, and delete records). This convoluted approach merges software design in a single representation, making interaction with information systems confusing, chatty, and underperforming. The CQRS focuses on how the system reads and writes data for a data store by separating operations into query models for reading and command models for writing data (Fowler M., 2006). This separation of concerns enables the minimization of complexity, making the application more maintainable, extensible, and flexible while maximizing the application’s performance, scalability, and security.

The project Class Diagram in Figure 1 and Figure 2 shows relationships among classes. There are three class types; entity classes correlate with database table structures, behavior classes are responsible for reading and writing data per CQRS Design Pattern, event classes handle the side effects of behavior classes. Behavior classes have “CRUD” prefixes followed by the entity class. For example, CreateIncidentCommand class follows Single Class Responsibility in SOLID principles as it only creates a dossier and stores it into the database system. Attach to behavior classes are event classes responsible for adding audit logs for historical tracking. Events classes are triggered by behavior classes in a series of event collaborations, where classes work together by communicating with each other by sending events when their internal state changes.

A picture containing diagram

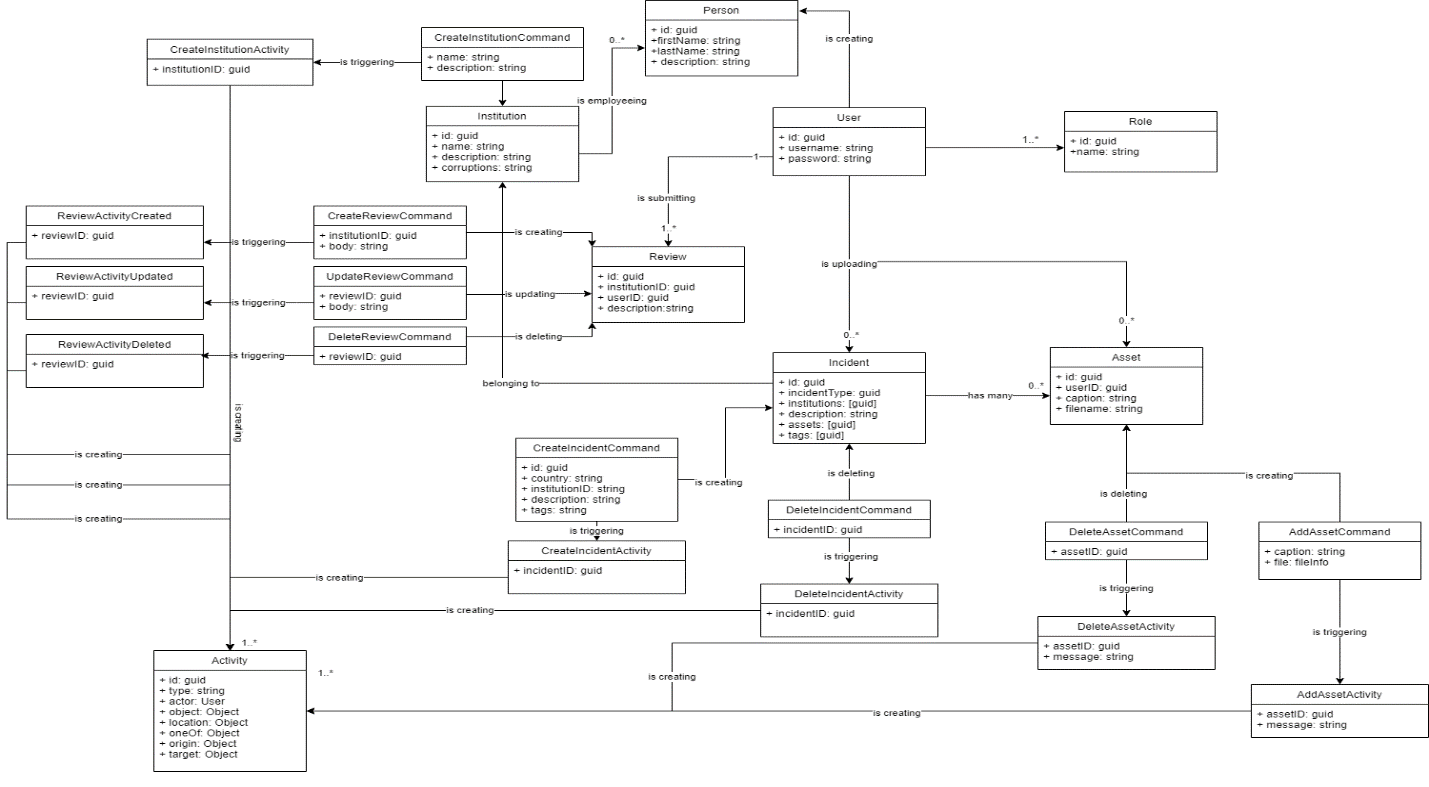
Description automatically generated

Figure 1. Class Design using the CQRS Design Pattern, http://github.com/thachp/c868/classes\_diagram.png

Figure 2. An Incident is created by CreateIncidentCommand,   
which trigger the CreateIncidentActivity to create an event log

## UI Design

There are two applications associated with this project: mobile and web applications. The mobile application targets activists who need to securely submit dossiers and other sensitive data via the mobile app. The web application targets analysts who need to download, review, and analyze dossiers. Both applications interface with the same backend services and are integral parts of the Dossier Management System.

The mobile application (Dossa) in Figure 3 shows a mockup of the main screen, requiring the user to press the “Report a corruption issue” button before submitting dossiers. The user will be taken to the Dossier Submission Form screen when clicked. The user will need to enter required inputs (institution, corruption types, description) and optional inputs like hashtags and file assets. Once all required fields are filled, the user clicks on the submit button located in the top right. Lastly, the user will be taken to the Processing screen, where a spinning indicator shows that their submission is in progress and other indicators show the entire process. Dossiers submission usually takes a few minutes, but the user may exit the screen anytime.

The web application (Dossier Management), in Figure 4 through Figure 6, shows mockups for three scenes: dashboard, institutions, and reporting. The dashboard scene consists of six widgets: number of total activists, number of dossiers, a bar chart of dossiers submission by months, list of institutions with most dossiers, recently added dossiers and recently added notes made by other analysts in the organization. The institutions’ scene enables analysts to search institutions by country, name, tag, and corruption types—the Dossier Management web application categories dossiers by the institution. Click on an institution record, and the user will be taken to the Institution Profile scene. The institution profile scene shows the institution name, a list of dossiers related to the institution, and a list of notes left by analysts. Click on a dossier open the Dossier Detail modal, where analysts can see actual dossier description, corruption types, and file assets such as images, audios, videos, documents, etc. Lastly, the reporting scene has four widgets: list countries by dossiers count, a list of institutions ordered by dossiers count, a corruption ration pie-chart displaying corruption types in percentage, and a hashtags widget showing popular hashtags relating to dossiers.

Other features are not in scope but are intended to release in later versions. These features aim to provide analysts enhanced ability to analyze raw dossiers.

* A feature to list people associated with the reported institution. Corruption occurred in the middle-high tiers, and higher tiers need to enforce accountabilities to reduce crime in lower levels. An organization chart will do wonders in understanding the institutions’ internal structure.
* Case management where dossiers can be grouped; create cases, edit a case, assign cases to an institution, associate dossiers to a case profile.
* A way for analysts to bookmark dossiers for viewing later, a scene to view individual asset detail, and functionality that rates institutions.

Graphical user interface

Description automatically generated

Figure 3. Mockup for the mobile application Dossa

Graphical user interface, application, PowerPoint

Description automatically generated

Figure 4. Mockup for the Dashboard scene

Text

Description automatically generated with medium confidence

Figure 5. Mockup for the Institutions List scene

Graphical user interface

Description automatically generated

Figure 6. Mockup for the Reporting scene

# Unit Test Plan

## Introduction

### Purpose

The project test plan intends to find bugs and fix them as soon as possible. The test plan uses three testing techniques: unit testing at the development level, acceptance testing, and black-box testing. Unit testing ensures code stability at the functions level is extremely important as it is the first chance to catch bugs. Acceptance testing determines whether the finished application meets users’ requirements. In black-box testing, testers know what a feature is supposed to do, but testers have no idea how it works. So, testers send random values to form inputs to simulate actual-world experiences. The mobile application, Dossa, will be used widely; therefore, it must be tested rigorously and a priority of all other unit tests.

### Overview

A feature of the Dossier Management System is to provide anonymity for when activists submit dossiers via the mobile application. The following test plan intends to test the functionalities of the dossier submission form and verify that user connections are secured through a third VPN application client called WireGuard Client. A successful test ensures better safety for activists.

## Test Plan

### Items

The test will be performed against the Dossa App, VPN Server, and backend services.

### Features

The test will perform tests on two features; VPN Connection, which provides user anonymity, and the Dossier Submission Form, which allows activists to submit dossiers.

### Deliverables

A Successful Submission upon dossiers submitted.

### Tasks

As an activist, perform test tasks on the Dossa mobile application.

* + Launch the Dossa App > Click on the “Report a corruption issue” button. Enter required inputs > Click Submit.
  + Verify that the application displays checks next to all progressing items to indicate successful submission. Specifically, a check next to Done.

### Needs

Three of the four subsystems of the Dossier Management System must be running. Dossa must be installed on a mobile device, VPN Server started and running, and the Parse backend services must also be running.

### Pass/Fail Criteria

A pass result entails a successful VPN connected status, and that form submission is completed with a successful notification. A failed test entails failed VPN connection, or that Dossier Submission Form submission has failed. If failed, verify that the VPN server is running. Also, check that required fields are include some values.

## Specifications

A key component in the web application is the Data Provider, which makes HTTP requests to the backend API in GraphQL. The screenshot below shows the London unit testing method to ensure that the Data Provider works for its purposes.

Text

Description automatically generated

Figure 7. London Method to Unit Testings using Jest

### Procedures

There are two approaches to unit testings; classical and London schools. The classical approach aims to verify a single behavior unit and isolate it from other tests. The London school, the method selected, aims to verify a small piece of code and do it in an isolated manner. The differences can be easily seen in codes; unit dependencies are replaced with duplicate mocks of the actual modules when using the London school. In Figure 7, pluralize and capitalize modules are mocked using the *mockImplementation* function provided by the testing framework, Jest.

The unit test structure is arranged in *setup, act, and asset* (SAA) pattern. The SAA pattern provides a uniform structure to all tests in the suite. The Setup section brings all dependencies to the desired state. The Act section capture any outcome, and the Assert section verify the outcome. In Figure 7, mocking is done in the setup section, and act & asset is done later in the test. The SAA pattern makes unit testing readable and understandable to the intended audiences, usually other developers.

### Results

Text

Description automatically generated

Figure 8. Successful unit testing results as provided by Jest

The screenshot above (Figure 8) shows extensive testing against the DataProvider as it is a critical component of the entire web application. The web application needs DataProvider to work thoroughly to process and display responses to requests to viewers.

# C4. Source Code

All source codes for all application subsystems are included with this documentation. It is also accessible via Github.com. Please see <https://github.com/thachp/c868>

# C5. Link to Live Version

The mobile application and a live website are accessible via the URL <https://www.dossa.network>.

Note that to install the mobile application on a mobile device, you must first install the Expo app on IOS or Android. Expo is an open-source platform for making universal native Android, iOS, and the web with JavaScript and React. It provides a convenient way for developers to publish mobile apps without going through app submission steps with the Apple App Store or Google Play.

# References

Fowler, M. (2006, June 19). *Event Collaboration*. Martinfowler.Com. Retrieved November 19, 2021, from https://martinfowler.com/eaaDev/EventCollaboration.html

Fowler, M. (2011, July 14). *bliki: CQRS*. Martinfowler.Com. Retrieved November 19, 2021, from https://martinfowler.com/bliki/CQRS.html

Khorikov, V. (2020). *Unit Testing Principles, Practices, and Patterns: Effective testing styles, patterns, and reliable automation for unit testing, mocking, and integration testing with examples in C#* (1st ed.). Manning.

Martin, R. C. (2000). *Design Principles and Design Patterns*. Universidad ORT Uruguay. Retrieved November 19, 2021, from https://fi.ort.edu.uy/innovaportal/file/2032/1/design\_principles.pdf

Millington, S. (2021, May 18). *A Solid Guide to SOLID Principles*. Baeldung. Retrieved November 19, 2021, from https://www.baeldung.com/solid-principles