

Dalhousie University

# Project Report

CSCI 5308 – Quality Assurance

**Submitted By:**

Samson Maconi	B00801169
Babatunde Adeniyi	B00792158
Yash Modi	B00799125
Ueli Haltner	B00526617

Group 2  
7-30-2019

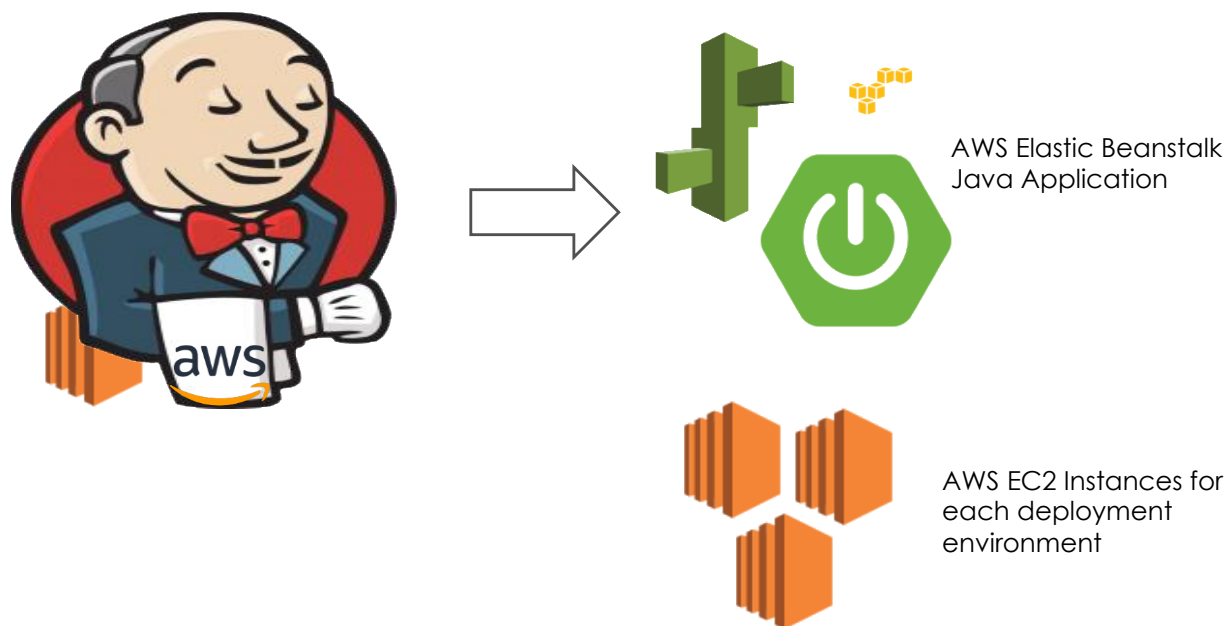
## Table of Contents

1. Continuous Integration .....	2
2. Design Patterns .....	3
Singleton .....	3
Strategy .....	3
Builder .....	4
Observer.....	5
3. Separation of Layers .....	5
4. Naming strategy and Coding conventions .....	5
5. Logging and Error Handling.....	6
6. Refactoring Performed.....	6
7. Technical Debt .....	7
8. Member Contributions .....	8
Ueli Haltner .....	8
Samson Maconi.....	9
Babatunde Adeniyi.....	10
Yash Modi.....	10

## 1. Continuous Integration

This section describes the composition of the Continuous Integration. The Continuous integration was setup using the following tools and services

- GitHub
- Amazon Elastic Compute Cloud (EC2)
- Jenkins
  - AWS SDK Plugin
  - AWS Elastic Beanstalk Deployment Plugin
  - AWS Elastic Beanstalk Publisher Plugin
- Amazon Elastic Beanstalk
  - Amazon Elastic Compute Cloud (EC2)
  - Amazon Simple Storage Service (S3)



*Figure 1: Continuous Integration Composition*

The Jenkins automation server was deployed on an EC2 instance for centralized access amongst team members. Jenkins polls the GitHub code repository at specified intervals and publishes changes to the Elastic Beanstalk service. The required environments for the application were deployed on EC2 instances through the AWS Elastic Beanstalk service. Which saved the source codes for each instance in S3 storages and deploys the application pushed from Jenkins. To achieve this, we had to install 3 AWS plugins on the Jenkins server. The AWS SDK plugin (for authentication and general AWS configurations including the preferred physical location of the AWS servers), the Elastic Beanstalk Deployment Plugin, and the Elastic Beanstalk Publisher plugin. The Deployment and Publisher plugins together specify the deployment environment and parameters for deploying Elastic Beanstalk applications.

Within each deployment environment, we configured environment properties specifying the database to be connected to and the name of the s3 bucket where the uploaded files will be stored. See Figure 2.

Environment properties

The following properties are passed in the application as environment properties. [Learn more](#)

Name	Value
FILESHARE_S3_BUCKET_NAME	csci5308-file-share-app-prod ✕
JDBC_CONNECTION_STRING	jdbc:mysql://db-5308.cs.dal.ca ✕

Figure 2: Deployment Environment Properties Configuration on AWS Elastic Beanstalk

## 2. Design Patterns

The following subsections describe the Design patterns that were used in the project. Each section will describe how the pattern was used and why.

### Singleton

#### Database Connection

The Singleton design pattern is used in FileShare Application to provide a database connection. It simply provides a global instance of the class to share all connections in the application. By initializing constructor of database connection class private, it restricts developers to create multiple instances. This only allows a single instance of a database connection class to get a connection and close connection to the database. The class uses method `getDbConnectionInstance` to get a single instance of `DatabaseConnection`.

#### Session Authentication Manager

The application used the Singleton design pattern to provide a global point to access sessions data and also provide one instance of the object which was responsible for the creation, initialization, and access. The session manager class retrieves `HttpSession` using `RequestContextHolder` and then manages the user session as required. The lazy initialization approach was used which initializes it when it's needed first.

#### S3StorageService

The `S3Storage` service, implementing the `IStorage` interface, is used to provide the CRUD operations for the uploaded files storage repository; using Amazon S3 storage service as the repository.

### Strategy

#### Document Sorting

Throughout the application the sorting feature is used for various dashboards. Sorting is performed through stored procedures which retrieves a list of documents in a specific order. As the application requires various types of sorting options, a strategy design pattern was implemented. A single dashboard controller is used for all various dashboards and instantiates a `DocumentSorter` object which encapsulates a single strategy. Based on the sorting option selected the sorting strategy can we

swapped. This allows the *DocumentSorter* object to execute whichever storing strategy is currently stored within the object without having to worry about special requirements. The sorting strategy is interchangeable as they all perform the same procedures with different results.

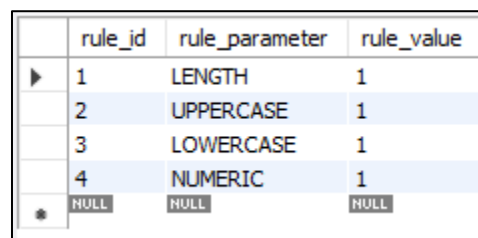
### Dashboard Type

A strategy design patterns were also used to select the various dashboard options themselves. The application three different dashboard options and the only different between them are the types of documents they show the user. Therefore, a strategy design pattern was implemented for the dashboard controller, where a *Dashboard* object contains a single dashboard strategy. As the user navigates through the different dashboards, the dashboard strategy is swapped out. These strategies contain different sets of defined Boolean values which describe the document list requirement for that dashboard. As the list is generated, the *Dashboard* object accesses its stored strategy and retrieves the set of Boolean values applicable to the current strategy. Lastly isolating the requirements for each dashboard through strategies allows for the use of a single dashboard controller to handle all dashboards and sorting options.

### Builder

#### Password Rules

In our project, configurable business logic is implemented and demonstrated through password rules. A set of rules is stored in a MySQL table as showing in **Figure 3**. Each rule contains an integer value which can be used to toggle the rule ON/OFF. The datatype was chosen to be an integer to consider a future adjustment in the interest of converting it to an adjustable value.



	rule_id	rule_parameter	rule_value
▶	1	LENGTH	1
	2	UPPERCASE	1
	3	LOWERCASE	1
	4	NUMERIC	1
✱	NULL	NULL	NULL

Figure 3: Password Rules table in MySQL database.

A builder design pattern was implemented to allow a password ruleset to be dynamically constructed based on the assigned value for each rule. Every time a password requires to be verified in either the signup page or profile page, the *PasswordValidator* object is created which instantiates a builder object. The builder object then passed to a director (*BuilderRuleSet* object). The director currently calls a stored procedure and adds the rule to the list if not equal to zero. The builder design pattern was selected as the password rule set contains complexity due to its dynamic nature. For example, if the previously described expansion were to be implemented of converting toggle values to range values the director could simply add this additional complexity as an additional step when creating the list of rules. Then the steps would include retrieving the table adding the enabled rules and lastly adjust the requirement based on a given range such as length of the password or number of specified characters.

### Document Deletion/Expiry

An observer design pattern is used to implement the expiry document feature. This feature provides the functionality to delete document permanently which stays in user's Trash bin for more than 30 days. The original idea is to implement a Polling mechanism; it provides timely functionality which checks the state of the subject and notifies the observers. However, there were many obstacles while implementing functionality based on time, non-polling method is used for document Expiry feature. In this implementation, DAO has been kept as an observer while the date of documents is kept as a subject. DAO observer is notified if any document is reaching past trashed date so DAO operation can be performed, and the document can be deleted permanently.

### 3. Separation of Layers

Layer separation was performed through the use of packages naming conventions and the framework itself. The Spring Boot framework was used for this project which included benefits of layer segregation. Due to the project structure requirement, all presentation layer code such as HTML CSS and JavaScript was contained within a *templates* folder under *resources*. Data was stored in three different MySQL databases which were available through the course to store all application data.

Lastly all business logic was stored in the project *java* folder and further divided into packages based on features for the project. These packages contained the logic required for that feature. To communicate between layers we used Controller classes and DAO classes. Controller classes contain all the mapping between business logic and front-end web pages while the DAO classes established the connection to the databases to call and retrieve data through stored procedures. All database data querying was done through stored procedures.

### 4. Naming strategy and Coding conventions

In order to ensure good readability and maintenance of the codebase, the following naming conventions were used:

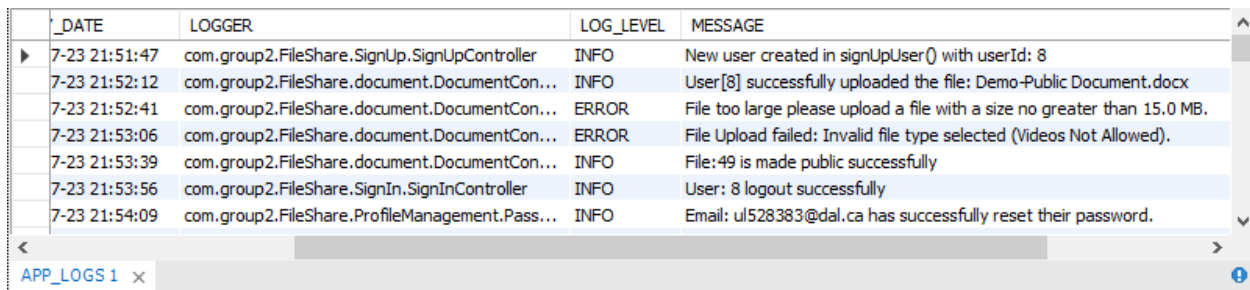
- Packages, interfaces and class names are all in Pascal case and ensure the words are nouns.
- Methods are all Camel case and should be verbs.
- Variables are also in Camel case and shouldn't start with an underscore or special character

Furthermore, all files under a package have a name which relates easily to the package. Since the application was built using MVC, all controllers have the 'Controller' suffix, all Data access object (DAO) classes—which provides an interface with the database— also have the 'DAO' suffix, and all validations were done in a separate class with 'Validation' suffix. These improve accessibility to aspects of the codebase.

In addition, the indentation style used was Tabs and ensure the configured tabs translate to 4 spaces.

## 5. Logging and Error Handling

For the FileShare application, the next generation logger Log4j 2 has been configured. Log4j tracks all the functionalities which are executed during runtime. Log4j is integrated with the application by adding Maven Dependency and configuration file. This logger is configured with the help XML file in the Project Folder. *Log4j2.xml* contains all the configurations done for the application related to logging. Here, three types of appenders are used which are Console, File and Database. These appenders are defined to save logs in different forms. In this Project, all the logs are saved in the database. File logging is used for only logs related to database connection. In case of a database connection failure, logs cannot be saved in the database. In this each type of logging, Pattern is defined to make sure we get all the relevant information such as timestamp, log level etc. Exceptions are used extensively in this project to get the detail exception stack trace for error handling. Overall, putting good usage of Logging and Exception has really helped us a lot in Error handling.



	DATE	LOGGER	LOG_LEVEL	MESSAGE
▶	7-23 21:51:47	com.group2.FileShare.SignUp.SignUpController	INFO	New user created in signUpUser() with userId: 8
	7-23 21:52:12	com.group2.FileShare.document.DocumentCon...	INFO	User[8] successfully uploaded the file: Demo-Public Document.docx
	7-23 21:52:41	com.group2.FileShare.document.DocumentCon...	ERROR	File too large please upload a file with a size no greater than 15.0 MB.
	7-23 21:53:06	com.group2.FileShare.document.DocumentCon...	ERROR	File Upload failed: Invalid file type selected (Videos Not Allowed).
	7-23 21:53:39	com.group2.FileShare.document.DocumentCon...	INFO	File:49 is made public successfully
	7-23 21:53:56	com.group2.FileShare.SignIn.SignInController	INFO	User: 8 logout successfully
	7-23 21:54:09	com.group2.FileShare.ProfileManagement.Pass...	INFO	Email: ul528383@dal.ca has successfully reset their password.

Figure 4. Application logs implemented using Log4j 2

## 6. Refactoring Performed

### Shotgun Surgery in Sort Strategy

Each sort strategy instantiated the document DAO class independently, which it uses to retrieve the document list. This made it difficult to apply a mock DAO class to alter where the data was being retrieved from for unit testing. To resolve this, I moved the instantiation to a higher abstract layer extends all strategies. This allowed a default DAO to be assigned in a single location and be applied to all strategies. Additionally, by creating an update DAO method in the abstract class, each strategy also has the flexibility to be assigned its own DAO class.

### Search Bar Data Clump

The dashboard controller stored and use local data related to the search bar feature in its own class, such as if a search is required and what the search phrase is. This data was clumped together into a little class called SearchBarHandler.

### Sign Up Controller and Password Recovery Controller Long Method Smell

The two controllers Sign Up, and Password Recovery contained a single method which performed notification generation, various business logic, and front-end navigation. This method was decomposed to allow the mapped method to instantiate the classes that are required for the work, provide front-end feedback, and to navigate the front-end based on the result. The business log was composed into a separate method with an identical name which performs the work while additional tasks were composed further into smaller private methods each with their own responsibility. An enum was chosen

as the returned parameter as it clearly describes the state upon return. An additional method then uses that enum to generate the proper string required for the front-end feedback. Feedback strings are now easily maintainable as they are in a single method, and the business logic has been isolated to allow unit testing with interface dependencies.

## 7. Technical Debt

### Stored Procedure Template Method

There is a large amount of duplicate code which occurs across all the DAO classes. Each function creates its own connection and callable statement object to populate a stored procedure with inputs followed by execution and parsing of the result set. To resolve this, all current DAO methods should be able to call a single template method which handles every stored procedure and list of varying inputs. This can be done by passing an array of enums indicating the datatype and an associated array of java object types which are the inputs. Based on the enum, the object can be unboxed into the required primitive datatype and inserted into the callable statement. After execution, the result set will then be returned from the template method and the invoker can parse the results set as required.

### Dashboard Strategy and Sorting Strategy Inappropriate Intimacy Smell

While some refactoring was applied to the sorting strategy (Sort Strategy Abstract class), the design pattern is still not perfect. As the Document Sorter class contains the strategy, it is internally dependent on an embedded DAO object. Additionally, the execute strategy method is dependent on the private parts of the Dashboard (Strategy) class. The DAO class and dependency on the Dashboard class should be extracted to a higher level. A new class should be created such as a dashboard manager which encapsulated the dashboard strategy class, sorting strategy class and DAO class. The sorting strategy would then simply return the assigned stored procedure string and the dashboard manager can execute the stored procedure through the DAO class. Ultimately there is a lot of coupling between these classes that can be avoided through the extraction of these objects into a higher-level dashboard manager class.

### Password Generator Violates Open/Close Principle

The password generator violated the second SOLID principle in that its implementation does not allow for extension but rather requires modification. There is no flexibility in the permitted characters used to generate a password as they are declared in the class itself. This could be resolved by passing the strings of available characters. Additionally, this would be an ideal opportunity to apply configurable business logic to allow modifications of approved characters outside the codebase.

### Password Validation Logic in Data Layer

Due to a misunderstanding early in the semester, it was realized during the final presentation that password rules should be stored in the database and not password rule validations. Currently, each password rule takes a string as input and calls a stored procedure to validate if it is valid. The logic in these stored procedures should be implemented in the appropriate rule object itself along with another input which is the range or count value from the password rule table (configurable business logic). Then the rules are more dynamic and can check for numerous numeric characters rather than just one.

### Password DOA Violates the Single Responsibility Principle

The PasswordDAO class violates the first SOLID principle in that it performs an additional task of encoding a raw password before it is stored in the database. This can be resolved using the stored



procedure template method described earlier. By removing the process of calling a stored procedure to a separate class, the DAO method would reduce its responsibilities to the preparation and extraction of parameters. Password encoding would then be part of input preparation before calling the stored procedure.

#### Inefficient/Improper method of disable logging in unit test

The approach used in disabling logging in SharedDocumentLinkValidator during unit testing isn't the best and maintainable approach. A better approach would be to set up log4j logger test config which disables logging to the database when running unit tests.

## 8. Member Contributions

Ueli Haltner

Java Files	Stored Procedures	Presentation Layer
DashboardController.java SearchBarHandler.java Dashboard.java IDashboard.java PrivateDashboard.java PublicDashboard.java TrashDashboard.java IPasswordDAO.java IPasswordEncoder.java IPasswordValidator.java PasswordDAO.java PasswordEncoder.java PasswordForm.java PasswordValidator.java ProfileController.java BuilderRuleSet.java IPasswordRuleBuilder.java PasswordRulesObject.java StandardPasswordRulesBuilder.java IPasswordRule.java IPasswordRuleDAO.java IPasswordRuleSet.java LengthRule.java LowercaseCharacterRule.java NumericCharacterRule.java PasswordRuleAbstract.java PasswordRuleDAO.java PasswordRuleSet.java UppercaseCharacterRule.java IMail.java IPasswordGenerator.java PasswordGenerator.java PasswordRecoveryController.java	create_profile get_documents_created_filter get_documents_filetype_filter get_documents_modified_filter get_documents_name_filter get_documents_size_filter get_profile password_length_rule password_lowercase_character_rule password_numeric_character_rule password_rules password_uppercase_character_rule update_password update_recovery_password user_exists	profile.html landing.html ▪ <body> only public_dashboard.html ▪ <body> only trash_dashboard.html ▪ <body> only

PasswordRecoveryMail.java ISignUpDAO.java SignUpController.java SignUpDAO.java SignUpForm.java IMailService.java MailService.java IUser.java User.java DocumentDAO.java <ul style="list-style-type: none"> <li>getDocumentList</li> </ul> DocumentController.java <ul style="list-style-type: none"> <li>getDocumentCollection</li> </ul> DefaultProperties.java <ul style="list-style-type: none"> <li>getMailHost</li> <li>getMailPort</li> <li>getMailUsername</li> <li>getMailPassword</li> <li>getMailSmtpSSLTrust</li> <li>getMailSmtpAuth</li> <li>getMailSmtpStartTlsEnable</li> <li>getMailTransportProtocol</li> <li>getMailDebug</li> </ul> DashboardStrategyTest.java DocumentSorterStrategyTest.java SearchBarHandlerTest.java SortTest.java BuilderRuleSetTest.java PasswordEncoderTest.java PasswordGeneratorTest.java PasswordRecoveryControllerTest.java PasswordRuleTest.java PasswordValidatorTest.java StandardPasswordRuleBuilderTest.java SignUpControllerTest.java		
---	--	--

## Samson Maconi

Java	Stored procedures	HTML & XML
Document.java DocumentController.java DocumentDAO.java <ul style="list-style-type: none"> <li>createPrivateShareLink</li> <li>getTotalFileSize</li> </ul> IDocumentDAO.java IStorage.java MockStorageService.java MockStorageServiceTest.java S3StorageService .java	create_private_shared_link get_config get_total_filesize	dashboard.html <ul style="list-style-type: none"> <li>File Sharing</li> <li>Modal UI</li> <li>Notifications</li> </ul> guest_dashboard.html <ul style="list-style-type: none"> <li>Notifications</li> </ul> public_dashboard.html <ul style="list-style-type: none"> <li>Notifications</li> </ul> trash_dashboard.html <ul style="list-style-type: none"> <li>Notifications</li> </ul>

IValidator.java FileValidator.java StorageLimitValidator.java IConfigDAO.java ConfigDAO.java DefaultProperties.java DefaultPropertiesTest.java		public_acc.xml
--	--	----------------

### Babatunde Adeniyi

Java	Stored procedures	HTML & XML
AuthenticationInterceptor.java AuthenticationSessionManager.java ICompression.java ZipCompression.java GuestDashboardController.java SharedDocumentLink.java SharedDocumentLinkValidator.java ISignInDAO.java SignInDAO.java SignInController.java SignInForm.java SignInValidator.java PublicAccess.java WebMvcConfig.java DocumentDAO.java <ul style="list-style-type: none"> <li>• getDocument method</li> <li>• getLinkedDocumentRefWith method</li> </ul> AuthenticationSessionManagerTest.java ZipCompressionTest.java SharedDocumentLinkValidatorTest.java SignInValidatorTest.test PublicAccessTest.test	get_document get_shared_document	guest_dashboard.html public_acc.xml

### Yash Modi

Java	Stored procedures	HTML & XML
DatabaseConnection.java DocumentDAO.java <ul style="list-style-type: none"> <li>• getDocuments</li> <li>• addDocument</li> <li>• updateDocument</li> <li>• DeleteDocument</li> </ul> DateObserver.java DeleteDocument.java	add_document delete_document get_documents update_document	log4j2.xml landing.html dashboard.html trash_dashboard.html

DocumentSubject.java Observer.java PinDocumentsLimit.java DefaultProperties.java <ul style="list-style-type: none"> <li>• getPinDocumentsLimit</li> <li>• getDeleteDocumentExpiry</li> </ul> DatabaseConnectionTest.java DeleteDocumentMock.java DeleteDocumentTest.java DatabaseMock.java DocumentDAOMock.java DocumentDAOTest.java PinDocumentMock.java PinDocumentsLimitTest.java DefaultPropertiesTest.java <ul style="list-style-type: none"> <li>• getPinDocumentsLimitTest</li> <li>• getDeleteDocumentExpiryTest</li> </ul>		
--	--	--