John Hancock - Project Coeus Architecture

|  |  |  |
| --- | --- | --- |
| **Author** | **Revision Number** | **Date** |
| Architect | 1.0 |  |
|  |  |  |
|  |  |  |

[John Hancock - Project Coeus Architecture 1](#_Toc448432764)

[1. Design 3](#_Toc448432765)

[1.1 Overview 3](#_Toc448432766)

[1.1.1 Work Flow 3](#_Toc448432767)

[1.1.2 Data Flow 3](#_Toc448432768)

[1.2 Component Requirements 4](#_Toc448432769)

[1.2.1 Assemblies 4](#_Toc448432770)

[1.2.2 Third Party Libraries 4](#_Toc448432771)

[1.3 Application Management 4](#_Toc448432772)

[1.3.1 Threading 4](#_Toc448432773)

[1.3.2 Transactions 4](#_Toc448432774)

[1.3.3 Configuration 4](#_Toc448432775)

[1.3.4 Persistence 5](#_Toc448432776)

[1.3.5 Logging 6](#_Toc448432777)

[1.3.6 Auditing 6](#_Toc448432778)

[1.3.7 Security 7](#_Toc448432779)

[1.3.8 Exception Handling 7](#_Toc448432780)

[1.3.9 Performance 8](#_Toc448432781)

[1.3.10 Scalability 8](#_Toc448432782)

[1.3.11 Print 8](#_Toc448432783)

[1.3.12 Archive assessments 8](#_Toc448432784)

[1.4 Deployment Constraints 8](#_Toc448432785)

[1.4.1 Technology overview 8](#_Toc448432786)

[1.5 Development Standards: 9](#_Toc448432787)

[1.6 Interfaces Classes Overview 9](#_Toc448432788)

[1.7 Changes to Existing System 9](#_Toc448432789)

[2. User Interface 9](#_Toc448432790)

[3. Included Documentation 9](#_Toc448432791)

[3.1 Architecture Documentation 9](#_Toc448432792)

# Design

Currently, various JH business units within the US Division have their own risk assessment process. Some business units are more advanced than others. The process involves assessing the risks of many functions and reporting on results. This siloed approach leads to a lack of consistency in regards to data input, templates, frequency of assessments and reporting at the US Division Operations level. Furthermore, there is no tool or means of capturing an overall inventory of controls.

This project will build a tool to capture and assess risks and controls data at the function level across various Business Units within US Division Operations

This architecture covers both backend and frontend implementations.

## Overview

### Work Flow

This application consists of REST API backend and frontend HTML5 pages.

*Backend*

This module is built with ASP.NET Web API framework and Entity Framework.

The basic workflow for serving a REST API is as follows:

1. An HTTP request is made to API endpoint.
2. This HTTP request is translated into a Controller action method
3. The ASP.NET Web API framework calls AuthorizationFilter to check if user is logged in, and respond with error if not.
4. The Controller method calls one or more of the back-end services to perform business operations.
5. The Controller method returns data model objects (if needed) as result.
6. If there is any exception experienced when running the Controller action method, the ExceptionFilter will be called to interpret exception as HTTP response status code.
7. The ASP.NET Web API framework automatically serializes data model objects as JSON response to the API caller.

The backend services will be implemented as POCO (Plain Old C# Object) services, [Entity Framework](http://msdn.microsoft.com/en-us/data/ef.aspx) will be used to access the SQL Server 2012 database.

The backend services will export data in Excel format.

The backend services will access Active Directory for authentication.

*Frontend*

Front end is implemented using AngularJS framework. The pages interact with AngularJS controllers, which call AngularJS services, which call back end REST API.

### Data Flow

At a high level the data flow of this application is as follows:

* All users will login with username and password.
* The owner can submit the assessment or save the assessment as draft.
* The BU Functional Approver can approve or reject the assessment which is submited by owner.
* The BU Risk Management Approver can approve or reject the assessment which is approved by BU Functional Approver.
* The Divisional Risk Management Approver can approve or reject the assessment which is approved by BU Risk Management Approver.

## *Expression Evaluation*

Currently, the formula/expressions for calculating risk scores is unknown. The business user should have ability to add/edit the formula/expressions.

To support evaluating formula/expressions in RiskScoreCalculator, the [Flee Expression Evaluator](http://flee.codeplex.com/) is used, it provides support for [variables](http://flee.codeplex.com/wikipage?title=ExpressionVariables&referringTitle=Examples). The [Flee Expression Language Reference](http://flee.codeplex.com/wikipage?title=LanguageReference&referringTitle=Home) provides detail format / grammar of expression.

## Component Requirements

### Assemblies

* John Hancock - Project Coeus Backend REST API Assembly

This assembly will implement the data models, exceptions, backend services, filters and controllers.

* John Hancock - Project Coeus Frontend Assembly

This assembly will implement Web Frontend app using HTML5 and AngularJS. It will be responsible for integrating backend REST API.

### Third Party Libraries

* Unity 3.5: <http://unity.codeplex.com/>

Unity will be used as the IoC container.

* Log4net 1.2.13: <http://logging.apache.org/log4net/>

Log4NET will be used to perform logging

* AngularJS 1.2.28 <https://angularjs.org/>

It is used as front end JavaScript framework.

* Microsoft OpenXML SDK 2.0: <http://msdn.microsoft.com/en-us/office/ee358824.aspx>

OpenXML is used to generate Excel.

* ClosedXML 0.76.0: <https://closedxml.codeplex.com>

ClosedXML is used overtop of the OpenXML. The usage of ClosedXML simplifies the usage.

* jqplot 1.0.9: <http://www.jqplot.com/>

jqplot is used to draw charts in frontend.

* Flee 0.9.26.0: <http://flee.codeplex.com/>

Flee is used to evaluate formular/expressions.

## Application Management

### Threading

Each controller will be created afresh for every request, hence they need not be thread safe, and it is not required to make ASP.NET Web API controller methods thread safe.

The backend services must be effectively thread-safe. It does not have to worry about the thread-safety of the input elements. The Unity IoC won’t be considered as a thread safety issue either.

Codes that use ADO.Net Entity Framework are thread-safe because DbContext is created and disposed in each service operation.

### Transactions

ASP.NET Web API controllers will use transactions in case where more a service’s write operation is being used, and it is safe because transactions in Entity Framework are across DbContext's. The old-school [TransactionScope](http://msdn.microsoft.com/en-us/library/system.transactions.transactionscope.aspx) class will be used for this purpose.

|  |
| --- |
| using(TransactionScope scope = new TransactionScope())  {  /\* Perform transactional work here \*/  //No errors-commit transaction  scope.Complete();  } |

### Configuration

*Backend*

The configuration will be done through Unity Injection. All the backend services, controllers and filters will have a CheckConfiguration() method which will throw ConfigurationException if any of the injected properties has invalid value, use the following xml fragment in app.config file to instruct Unity to invoke this method automatically:

|  |
| --- |
| <register type="ISecurityService" mapTo="SecurityService">  <method name="CheckConfiguration" />  </register> |

Please note that the IUnityContainer will pick up the application settings from the web.config’sappSettings section and will then register them with itself (see the sample GetUnityContainer method implementation). These settings will then automatically become available to the properties in the controllers.

The Unity will be used for Dependency Injection.

DB connection is configurable in the connectionStrings section of web.config, refer to [ADO.Net Entity Framework Connections and Models](http://msdn.microsoft.com/en-US/data/jj592674) for details.

Logging configuration is done in ILog configuration file; logger for services is configurable through BaseService/BaseController.Logger.

After this all properties marked with [Dependency] attribute will be injected by the container.

Note that the services and Log4NET ILog instances will be configured in a singleton manner i.e. the same instance will be injected into all the controllers/services.

The global JSON formatter should be configured to use camel case in the Application\_Start method of global.asax.cs, so that JSON property names won't be capitalized:

|  |
| --- |
| varjson = GlobalConfiguration.Configuration.Formatters.JsonFormatter;  json.SerializerSettings.ContractResolver = new CamelCasePropertyNamesContractResolver(); |

Filters will be registered as global filters. This can be done as follows in the Application\_Start method of global.asax.cs as follows:

|  |
| --- |
| GlobalFilters.Filters.Add(new AuthorizationFilter());  GlobalFilters.Filters.Add(new ExceptionFilter()); |

*Frontend*

The front end stores configuration parameters in a config.js file as “config” object like below:

var config = {

parameter1 : “value1”,

parameter2 : “value2”,

…

};

The following configurations are identified:

* REST\_SERVICE\_BASE\_URL
  + This configuration option specifies the base URL of the back end REST services. Required. String.

### Persistence

Backend services will use [Entity Framework](http://msdn.microsoft.com/en-us/data/ef.aspx) to access the SQL Server 2012 database.

The Entity Framework is a set of technologies in ADO.NET that support the development of data-oriented software applications. In this project we use the [Database First](https://msdn.microsoft.com/en-us/data/jj206878) approach (since Model First Approach will result in bad naming convention for database tables), following which the architect has provided the ERD (ERD.mwb), assemblers should take the ERD (database schema) and leverage DbContext Generator (example usage is [here](https://msdn.microsoft.com/en-us/data/jj206878)) to generate model classes and EDMX file. Any change to the model during the development should first result in changes to database schema and then the model classes and EDMX file can be regenerated.

Lazy Load

Since ADO.Net EF DbContext is created and disposed inside each service method, the service API callers won’t be able to access associations of the entity if they are [lazy loaded](http://msdn.microsoft.com/en-us/data/jj574232.aspx). To resolve this, either we need to explicitly load all the associations in service methods, or we can turn off the lazy loading feature for all the entities. Since entities in this module don’t have complicated associations (which means turning off lazy loading won’t affect performance) and we need all associations to be loaded, to ease the pain of adding codes for eagerly loading in every service method, the design decision is made to turn off lazy loading, this is implemented in the CreateDbContext() method of BasePersistenceService.

More discussions about this topic can be found [here](http://social.msdn.microsoft.com/Forums/en-US/adonetefx/thread/db1cd5d2-4bd0-4262-8899-f62b55397c40) and [here](http://gavindraper.com/2010/12/07/using-the-entity-framework-with-wcf/).

If not mentioned explicitly, following principles should be used to process the child entities when creating/updating/deleting/retrieving an entity:

* Create an entity:
  + The one-to-many child entities should also be created.
  + The many-to-one/many-to-many child entities should not be created, only the relationships should be created.
* Update an entity:
  + If there's an additional method to update the child entity, ignore this child entity in Update method. Otherwise:
    - The one-to-many child entities should also be updated. It means that the old child entities should be removed, and the new child entities should be persisted.
    - The many-to-one/many-to-many child entities should not be updated, only the relationships should be updated. It means that the old relationships should be removed, and the new relationships should be persisted.
* Delete an entity:
  + The one-to-many child entities should also be deleted.
  + The many-to-one/many-to-many child entities should not be deleted, only the relationships should be deleted.
* Retrieve an entity/Search Entities:

As mentioned above, all child entities should be loaded.

### Logging

Logging will be done using Log4NET.

The ILog instance will be injected into the services/controllers/filters using IoC.

The following situations will be logged:

* All exceptions originating from the services including cause and stack trace.
* The timestamp of the log
* The level (ERROR for exceptions and DEBUG otherwise)
* Any other informational message that may help application troubleshooting

Sensitive information like user credentials, database connection credentials should not be logged.

### Auditing

*Audit Properties*

API Controller is responsible to set the auditable properties (CreatedBy, CreatedTime, LastUpdatedBy, LastUpdatedTime) before modification method is called.

*Audit Record*

The application will maintain a history of user actions that modifies (including CUD) data. An audit record will look as follows:

| **Data Element** | **Description** |
| --- | --- |
| Username | The username of the user that performs the action |
| Action | The action corresponding to this audit record. |
| Item Type | The type of the entity being modified, in the case of using a service method that modifies data. Note that the fully qualified .NET type name should be used. |
| Item Id | A long value of the entity being modified, in the case of using a service method that modifies data. |
| Field | The changed field. |
| Previous value | The previous data of the audited value in JSON format.(if any) |
| New value | The new data of the audited value in JSON format.(if any) |
| Timestamp | The timestamp of the auditing moment. The same timestamp should be used for all audit records that result from the given action. |

Auditing will be done at API Controller tier, and the audit record will be persisted using IAuditService.

### Security

*Authentication*

This application uses username/password for authentication. The username/password authentication will be done via Active Directory.

An access token will be granted if authentication succeeded. The access token can be used in subsequent API calls for authentication purpose (in standard HTTP "Authorization" header as "Bearer" scheme) until the token expires. SecurityService.Authenticate will be used to check if the given token is valid and not expired.

AuthorizationFilter is responsible to verify the access token in "Authorization" header and reject requests without valid access token.

*Authorization*

Role-based authorization will be performed. AuthorizationService.IsAuthorized will be used to check if a role has permission to perform a specific action.

The REST APIs are supposed to be exposed via HTTPS.

### Exception Handling

ASP.NET Web APIControllers can throw regular .NET exceptions in case of known error cases. In case of exceptions received from calling the backend services, these exceptions will be allowed to bubble through.

ExceptionFilter#OnException method will be called in case any exceptions in the Controller method, and it will translate exceptions to HTTP response codes.

This module defines the following exceptions:

* ServiceException : base exception for this application
* PersistenceException : thrown if any error occurs during accessing database
* EntityNotFoundException : thrown for requests trying to update a non-existing data entity.
* AuthenticationException - thrown for authentication related errors.
* AuthorizationException - thrown for authorization related errors.
* ConfigurationException : thrown by business services and frontend controllers if any configuration is invalid

Please refer to the "Exceptions Class Diagram".

If any error occurs during JavaScript execution in front end, error message will be displayed.

### Performance

The application services will provide the means to get lists of data that are pageable and sortable. The paging will require the provision of a page number and page size. The page number should be 1-based, with page of 0 meaning there should be no paging and all data should be returned. Page size will be a positive number.

Sorting requires the provision of the name of the column to be sorted, and the sort order. In most cases, the service will provide some default sorting, so the methods can be called with a null column and sort order, which would be interpreted to mean the sorting is to be whatever the service deems to be as default.

### Scalability

There’s no explicit scalability requirement, and this solution does not prevent future scalability.

### Print

The browser build-in print functionality will be used for "Print Report" in frontend.

### Archive assessments

The prior year assessments can be archived by executing following sql in database:

|  |
| --- |
| CREATE TABLE Assessment\_2014 AS  SELECT \*  FROM Assessment  WHERE SubmiterTime BETWEEN TO\_DATE(20140101, 'yyyyMMdd') and TO\_DATE(20141231, 'yyyyMMdd')  DELETE  FROM Assessment  WHERE ID IN (SELECT ID FROM Assessment\_2014);  // Archive other tables(FunctionPerformedSite, FunctionChange, KPISLAAssessment, ProcessRiskAssessment, ControlAssessment) that associates the archived assessments similarly. |

## Deployment Constraints

The website will be deployed on the IIS server on Windows Server 2008+ / Azure.

In Visual Studio, all the C# code in the application gets compiled into a DLL in the bin folder. Other static content needs to be manually copied to the IIS server.

This application will access an SQL Server 2012 database.

This application will interact with Active Directory.

When deployed on Azure, it may be deploy with load-balancing, this architecture can work with load-balancing without code modification.

### Technology overview

* Windows Server 2008+ / Azure with .NET Framework 4.5 installed
* SQL Server 2012
* Microsoft IIS 8.5
* C# 5.0
* .NET Framework 4.5
* ASP.NET Web API 2.2
* HTML5
* Unity 3.5 <http://unity.codeplex.com/>
* Log4net 1.2.13 <http://logging.apache.org/log4net/>
* AngularJS 1.2.28 <https://angularjs.org>
* Microsoft OpenXML SDK 2.0 <http://msdn.microsoft.com/en-us/office/ee358824.aspx>
* ClosedXML 0.76.0 <https://closedxml.codeplex.com>
* jqplot 1.0.9: <http://www.jqplot.com/>
* Flee 0.9.26.0: <http://flee.codeplex.com/>

## Development Standards:

The assembly development must adhere to the guidelines as outlined in the [TopCoderAssembly Competition Tutorial](http://apps.topcoder.com/wiki/display/tc/Assembly+Competition+Tutorials).

## Interfaces Classes Overview

See TCUML

## Changes to Existing System

None

# User Interface

The user interface will be HTML5 and AngularJS technologies.

# Included Documentation

## Architecture Documentation

* Class Diagrams
* Sequence Diagrams
* Application Design Specification
* Assembly Specification
* REST API Specification
* ERD