

|  |  |  |  |
| --- | --- | --- | --- |
|  | ***AND WHOSOEVER PUTS HIS TRUST IN ALLAH,***  ***THEN HE WILL SUFFICE HIM.*** |  |  |



|  |  |  |  |
| --- | --- | --- | --- |
|  | **Date: 25 May 2018**  **Version: 1.0** |  |  |

**Conventions, Guidance**

**And**

**Lessons Learned**

Preface

بسم الله الرحمن الرحيم

**In the name of Allah, Most Gracious, Most Merciful**

This document was written with focus on letting readers understand the working architecture, concepts behind the scene and technology basis. We hope you enjoy reading this document and make much use of it.

*Conventions, Guidance and Lessons Learned*

Target Readers

This document mainly targets ARCHITECTURE, IT and/or JEE lovers.

Acknowledgment

Many Thanks for **Allah** for help over the whole life to achieve continuous success and we pray for **Allah**'s support in the following periods of life.

Thanks for every team member and every engineer who share constructing this document, enhancing its content, read it or put it on his reading list.

Thanks for families, friends and supporters.

Table of Contents

[Preface 2](#_Toc70524392)

[Target Readers 3](#_Toc70524393)

[Acknowledgment 4](#_Toc70524394)

[Table of Contents 5](#_Toc70524395)

[1 Tools 7](#_Toc70524396)

[2 Libraries 8](#_Toc70524397)

[2.1 Primefaces-Hibernate During 2018/2019 8](#_Toc70524398)

[2.2 Richfaces-Hibernate During 2014/2018 10](#_Toc70524399)

[3 Architecture 12](#_Toc70524400)

[3.1 Introduction 14](#_Toc70524401)

[3.2 Presentation Layer 15](#_Toc70524402)

[3.3 Backend Layer – Business Logic 18](#_Toc70524403)

[3.4 Backend Layer - Workflow 20](#_Toc70524404)

[3.5 Backend Layer – Data Access 21](#_Toc70524405)

[3.5.1 Data Access Management 21](#_Toc70524406)

[3.5.2 Hibernate Session Behavior 23](#_Toc70524407)

[3.5.3 ORM 24](#_Toc70524408)

[3.5.4 Reporting 25](#_Toc70524409)

[3.5.5 Auditing 27](#_Toc70524410)

[3.6 Backend Layer - Integration 28](#_Toc70524411)

[3.7 Data Repository Layer 29](#_Toc70524412)

[3.7.1 Data Types 30](#_Toc70524413)

[3.7.2 Table Creation 31](#_Toc70524414)

[3.7.3 Constraints 32](#_Toc70524415)

[3.7.4 DB2 Built-In Scalar Functions 33](#_Toc70524416)

[3.7.5 DB2 Indexes Creation (DBAs) 34](#_Toc70524417)

[3.8 Security 35](#_Toc70524418)

[4 Project Structure 36](#_Toc70524419)

[4.1 Projects and Packages design 36](#_Toc70524420)

[4.2 Configuring Projects in Eclipse 37](#_Toc70524421)

[5 Version Control Systems 38](#_Toc70524422)

[5.1 SVN 38](#_Toc70524423)

[5.2 Git 39](#_Toc70524424)

[Appendix A: Installation Guide JDK11 – WebLogic 14c 40](#_Toc70524425)

[A.1 JDK 11.0.10 LTS 40](#_Toc70524426)

[A.2 WLS 14.1.1.0.0 40](#_Toc70524427)

[A.3 Eclipse 2020-06 41](#_Toc70524428)

[Appendix B: WAS and DB2 42](#_Toc70524429)

[B.1 IBM Installation Manager 42](#_Toc70524430)

[B.2 IBM SDK and WAS 43](#_Toc70524431)

[B.2.1 Install IBM SDK and WAS 44](#_Toc70524432)

[B.2.2 Create a Profile 46](#_Toc70524433)

[B.2.3 Data Source 53](#_Toc70524434)

[B.2.4 Configuring Encoding 59](#_Toc70524435)

[B.3 Eclipse and plugins 59](#_Toc70524436)

[B.4 IBM Database Client for DB2 61](#_Toc70524437)

[B.5 Toad for DB2 62](#_Toc70524438)

[B.5.1 Installation 62](#_Toc70524439)

[B.5.2 Configuration 63](#_Toc70524440)

[B.6 IBM DS 65](#_Toc70524441)

[B.6.1 Installation 65](#_Toc70524442)

[B.7 iReport 66](#_Toc70524443)

[Appendix C: WebLogic 12.2.1.0.0 67](#_Toc70524444)

[Appendix D: Working on Multiple database 70](#_Toc70524445)

[Appendix E: Conversion from Richfaces to Primefaces 81](#_Toc70524446)

# Architecture





## Introduction

We will use N-tier Uni-direction architecture.

N-tier as we will utilize multi-tier in our architecture (Presentation – Integration – Business – Data Access – Data Repository).

Uni-direction (Anti of Bi-direction architecture) means that upper layers only have dependencies on the lower layers and lower layers have neither dependencies nor knowledge about the upper layers. (In that way we can change upper layers entirely with zero effect on the lower layers).

The architecture implementation should be fully segregated:

* The separation between the presentation layer and the business layer will be based on web-services through the integration layer.
* The presentation layer will never interact with the data access/repository layers.
* The separation between the data access layer and data repository layer will be based on data sources and entities/queries portability. Data access layer should be database independent.
* The external systems should interact with our system throw only the integration layer.
* To ensure that Uni-direction feature is achieved, examine the imports. No lower layer should contain any import statement from upper layer classes or technology library.
* Eliminate technology dependencies from which upper layers have on lower layers. For example: Business layer needs to control business transactions that will affect at the end database transactions. If these transactions will be controlled via the application (application managed), they will depend on the technology of the Data Access Layer (i.e., Hibernate Session Factory or Persistence Context or …). The whole idea is to isolate the business layer from these technology dependencies. We can achieve this goal via the Adapter Design Pattern or encapsulate them via repository utility.
* Integration layer providers are upper layer for business layer while the integration layer consumers are lower layer for business layer.

Of course, Standard Java Naming Conventions will be used (Interface – Class – Method – variables/attributes – Constants - …).

Usage of Enumerators and Resource Bundles will be empowered to avoid hard coding and match the Internationalization/Localization standards.

Try to register all static data for Internationalization/Localization, configuration or any coded value in database configuration tables and to be loaded in the application to make it easy to change these values and to avoid hard coding entirely.

The rest of this document is listings of conventions, guidance, best practices and lessons learned that we will follow.

***Of course, these guidelines could be broken at certain situations and that is the nature of our work BUT we will commit to it as we can***.

## Presentation Layer

### Commons

* Controller should not contain any business logic, but only call wrapper methods in the business layer or web-services clients (Managed bean in JSF is the controller part of the MVC design pattern).
* Strive for separate HTML/XHTML page for each screen in the application design and do not merge two or more screens in one file unless there is a very specific situation needs this. (For security – extendibility – maintainability).
* Empower the usage of composite/custome components and make use of the following tips for detection:
  + Whenever you found the same design for a portion of page shared between two or more pages.
  + Whenever you found a design for a portion of page shared between two or more pages with a rendering change behavior that could be parameterized.
  + Whenever you discuss with the team about one portion of a design that should be made as a composite component for future vision.
* Upload / Attachments component should be centralized for all pages.
* Hijri/Gregorian component should be centralized for all pages.
* For items selection and inquiry (DDLs – MSMS – Inquiry pages):
  + Use DDLs (drop down lists) for selection from short list values.
  + Use MSMS (mini search multi selection – a page that offers a simple search and ability to select single or multiple items from the result) pages for selection.
  + Use modes in MSMS pages to fit for more than one search perspective.
  + Lookups items should be loaded from the Database and mapped in the UI using (ID and Description).
  + Always design the search criteria to limit the number of returned items to a designed number suitable to the application (for example 100,000 items). Use client-side validations and server-side validations to make sure of that.
  + If the business requirements specifies that more than a designed number suitable to the application (for example 25,000 items) items to be returned, then database pagination methodology should be used.
  + Always use grid pagination unless there is a clue from the application design that avoiding the pagination is better for that case.
  + Define different page sizes:
    - Small page size 5.
    - Normal page size 10.
    - Large page size 50.
* Empower the usage of templates:
  + Main Template for the entire application (Header – Menu Tree – Body – Footer).
  + Plain Template for MSMS pages and pop-up pages.
* Buttons Locations:
  + Place form Actions at the bottom.
  + Place grid actions at the top.
  + MSMS buttons besides its fields.
  + Maybe we can use the search icons besides some fields for specific search behavior (like search above a tree of items).
  + Always use reset button with search button.
* Cross browsers compatibility should be considered (At least IE and Chrome).
* Use a red star on mandatory fields.
* Always use styles to differentiate mouse over and mouse selection for grid rows.
* Always change mouse cursor when it hovers an active button or link.
* Error page (Custom page for specific errors or exceptions and general error page).
* Validations:
  + No business validations should exist in the presentation layer.
  + Empower the usage of client-side validations to reduce the client-server communications.
  + Limit client-side validations for mandatory fields, data type and inquiry result limitation validations only. These validations should be repeated in the business layer.
  + Data length validations:
    - Input text field:
      * In JSF, use the “maxlength” attribute on “inputText” component.
    - Input text area field:
      * In JSF, use the custom JS function “limitMaxLength” in conjunction with “onchange” and “onkeyup” attributes on “inputTextArea” component.
    - In case that the input is mapped to be stored in the database, the size should be the exact database field size.
    - In case that Arabic letters stored in database in two units of the specified size (ex: DB2 or Oracle VARCHAR2 (BYTE)), then half the size should be specified.
    - If the input is not mapped to be stored in the database (ex: search parameter) we should set the max-length with MIN (100, length that should be specified in case of direct DB mapping).
    - 100 is a rule of thumb.
  + Data types validations:
    - Any input that should not accept non-numeric characters should be validated before form submission.
      * Use the custom utilities “isDigit” or “isFloat” in conjunction with “onkeypress” attributes for non-numberic validations.
      * To prevent errors from the copy/paste behavior, use the custom utility “isNumber” before form submission.
    - Any input (alpha-numberic or non-numeric) that accepts a certain group of characters only should be validated before form submission using regular expressions.
* There will be a general-purpose client-side utilities script file for common functionalities.
* Strive for avoiding writing any script code – other than linking to the scripting files – in HTML/XHTML files and use script files instead (Due to more organized code – cashing issues and performance). Organize these files around modules not around pages.
* There should be only one script function that will open the MSMS page. This function should contain all the parameters to open the screen. This function should have multiple wrappers with different signatures to be used by the caller pages to avoid the problem of editing in all the screens that open the same MSMS. Also, MSMS opener script function should be organized around modules.

### JSF

* We will have an abstract Base Managed Bean to handle common functionalities among managed beans (like messaging – Internationalization - …).
* We will have an abstract Base Workflow Managed Bean to handle common workflow functionalities.
* For each group of managed beans, if there are a range of common functionalities, we will make a base class for them.
* By default, use View Scope for managed beans unless there is a need for using other types of scopes. View scope beans must be serialized along with all contained attributes.
* Use only one managed bean per XHTML page and name the two files by the same name. Also name the managed bean annotation by the same name but in variable naming convention.
* Managed bean structure should be as following:
  + Attributes order, preferable in the same order like the XHTML page.
  + Constructor (contains the initialization code).
  + The method init() (mainly should not be used in request or view scops).
  + The method reset().
  + Action and listeners methods, in the same corresponding UI components order.
  + Utilities methods if needed.
  + All getters and setters in the same order as corresponding fields order (Use Eclipse generator).
* Context listener should be used for initializing and configuring the application.
* Phase Listener should be used for custom processing logic into the JSF lifecycle (like internationalization/localization) and debugging purposes.
* The target is to use the standard components, events … of JSF. Whenever JSF can fit, we will use it otherwise we will use the libraries features and components.
* Messages:
  + Empower the usage of the messages stack component in the templates.
  + Messages handling should be centralized in the base managed bean.
  + Differentiate messages in the stack by styles (Success – Warning – Error).
* Screen Lock:
  + Empower the usage of screen lock methodology in the templates to lock every page on every Ajax submit to stop handling the UI while a data manipulation is under processing to protect our system from unwanted changes.
  + All Navigation actions must be non-Ajax and show the screen lock.
* Empower the usage of passing parameters from XHTML pages to managed beans action and event methods (especially in grid operations).
* Executing and Rendering:
  + Try to avoid executing @form, execute your needed part only.
  + Do not re-Render @form, always re-Render your needed part only.
  + If a component Ajax event will re-Render the area in which the component is part of, then hide the reload component in the on complete of the Ajax event (Only when the component needs to catch the Ajax event for other purpose).
  + Do not re-Render a panel that has a rendered condition (You should render a parent container for it).
  + Do not re-Render a panel unless it has all of its children JSF components or it has an html table as a direct child.
  + Always use the form id as prefix while re-Render or execute.
  + Take care that not rendered / disabled components will not submit its value, but if there is a panel or column that will be hidden and has components, these components will submit its value unless it has the same render conditions.
* Use the following convention for naming in XHTML:
  + Do not name the element if you will not use that name in execute, re-Render…
  + Always name the main form as the same as the file name but in a variable naming convention and followed by “FormId”.
  + The value of the id attribute is the name of the field/component followed by ‘id’ suffix, example:
    - <h:inputText id="jobNameId" value="#{jobsInquiry.jobName}" />
  + In case the attribute is an ID, a foreign key as an example we captalize the ‘Id’ part of the field and follow it by ‘Id’ suffix, examples:
    - <h:inputText id="jobIDId" value="#{jobsInquiry.jobId}" />
  + In some cases, we need to map two components on the same field one is for view and the other to be hidden for submitting, we put ‘H’ before the ‘Id’ suffix in the hidden component:
    - <h:inputText id="jobNameId" value="#{jobsInquiry.jobName}" />
    - <h:inputHidden id="jobNameHId" value="#{jobsInquiry.jobName}" />
* Structure resources folder for (css – javascript – images – fonts – composites components).
* The file “faces-config.xml” should be used for java server faces configurations:
  + Supported locales.
  + Resource bundles.
  + Phase Listener.
  + Navigation rules.
* The file “web.xml” should be used for the application configurations:
  + Context Listener.
  + Context parameters (Libraries needed configuration – max upload size – max views in session).
  + Servlets and mapping (Faces servlet – web-services servlets).
  + Session Config:
    - Session timeout.
    - Cookie config.

### Angular

### Internationalization

* Internationalization will be handled via Resource Bundles.
* Resource Bundle keys should be classified:
  + For error messages: error\_
  + For labels: label\_
  + For notification and warning messages: notify\_
  + For questions: q\_
  + For titles: title\_
  + For font awesome: fa\_
* Strive for non-repeated values in resource bundles and make use of the following tips:
  + Adding new key: Always search in the resource bundle with the exact value and wanted meaning.
  + Removing key: Always search for usage before removing. Always strive for removing non-used keys by looking for the keys in any code subject for deletion.
  + Updating key value: Always search for usage before updating. If there is any usage inapplicable then we need another key so we should go to adding new key steps.
  + Empower the usage of parameterized messages.
* Resource Bundle keys naming convention:
  + After the only \_ key naming should follow Java convention in naming variables.
  + Key names should represent the message value not the business or technical design terms.
* The content of the resource bundles should be persisted in a configuration table in the data repository with specific column for each language.

## Integration Layer

* Services should not contain any business logic; it should wrap the business classes calls.
* Services should adopt the “Single Message Argument” pattern which means only one input and only one output message.
* Request Structure:  
  **AgenciesInquiryRequest**

**RequestMetaData**

SystemCode \*(Code represents the client system).

requestId \*(Client generated id – Recommended to be unique).

userId (Mandatory for your FE systems)

username (Preferable for external systems)

SecyirtyToken

firstIndex

fetchedRecordsSize

preferedLang

**RequsetDetails**

… (Basically, using dal/entities and/or integration/entities)

* Response Structure:

**AgenciesInquiryReponse**

**ResponseMetaData**

StatusCode (SUCCESS – ERROR – …)

errorCode

errorMessage

firstIndex

fetchedRecordsSize

recordsSize

requestId (From request)

providerId (System generated id for the request)

**ResponseDetails**

… (Basically, using dal/entities and/or integration/entities)

* All returned data from integration services should be always data objects (unless there is no data object).
* It is recommended to transfer all data either numbers or strings.
* Web service clients should have a proxy file to separate the application from the client structure with the naming convention “Client” at the end of the proxy’s name (i.e., EmployeesClient).
* Append “Service” in integration class name (i.e., EmployeesService, EmployeeDependentsService).
* Empower the usage of JAX WS or JAX RS as the standard technology for exposing services.
* Any other integration technology can be used AFTER discussion. Like when we should integrate with a message queue or spring rest controller.
* JAXB annotations will be used to adjust the XML messages and JSONB for JSON messages.
* Configure JAX WS context “services” in (web.xml) and JAX RS context “api” in (Application.java).
* Configure all of your clients’ providers WSDLs/URIs in your configuration entity (never hard code the domain, WSDL, URI, …
* For Security, SOAP Handler can be used in JAX WS and pre matching request FILTER in JAX RS.
* For Logging, SOAP Handler can be used in JAX WS and pre matching request FILTER and Writer Interceptor in JAX RS.  
  Pre matching request Filter gives access to all request data even URI info unlike Reader Interceptor. Writer Interceptor gives access to the output stream before and after serialization unlike response FILTER which does not have access to the serialized data.
* Recommended to use POSTMAN to test services. (We can use also SOAPUI).
* **JAX WS:**
  + Services will be published using servlet end point listeners.
  + WS Client Generation
    - Use META-INF/jax-ws-catalog.xml for generated clients to map a static WSDL to the target WSDL to avoid updating code in the generated files.
    - We should build the “jax-ws-catalog.xml” with logical URLs that will be replaced with physical URLs in runtime (maybe not possible in UI wars).
    - Use the following command wsimport -keep -verbose WSDL\_URL -p PACKAGE\_NAME -d TEMP\_DIRECTORY -s TEMP\_DIRECTORY –b FI
    - We always use -b option just to make binding customization like asynchronous methods and adaptors for data types customizations like date (If we will not use string).

|  |  |  |  |
| --- | --- | --- | --- |
| Option | Value | Example | Description |
| -keep |  |  | Keeps generated files |
| -verbose |  |  | Displays compiler messages |
|  | [WSDL\_URL](http://pns.ejada.com:8080/ejadaPns/PushClient?wsdl) | http://pns.ejada.com:8080/ejadaPns/PushClient?wsdl | The file that contains the machine-readable description of how the web service can be called, what parameters it expects, and what data structures it returns |
| -p | PACKAGE\_NAME | com.code.integration.webservicesclients.pushclient | Specifies a target package name to override the WSDL and schema binding customizations, and the default algorithm defined in the specification |
| -d | TEMP\_DIRECTORY | E:\Temp\ | Specifies where to place generated output files |
| -s | TEMP\_DIRECTORY | E:\Temp\ | Specifies where to place generated source files |
| -b | FILE\_PATH | E:\Temp\binding.xml | Specifies external JAX-WS or JAXB binding files |
| -Xnocompile |  |  | Skip compilation |

* **JAX RS:**
  + Unless discussed in exceptional scenarios, all HTTP methods will be POST with no path parameters or query parameters. Mainly to:
    - Satisfy the “Single Message Argument” pattern.
    - Avoid the GET URL size constraint.
    - Unify the development methodology.
  + A path URI segment will be designed for every service representing the business entity in plural form (e.g.: customers, agencies, employees, …) and another segment for every service method representing the behavior in noun form (e.g.: inquiry, management, statistics, …).
  + Use @jsonbProperty for customizing the field name.   
    (on field -> both serialization and deserialization / getter -> serialization only / setter -> deserialization only).
  + Register our services and providers (Filters and Interceptors) in singletons set.  
    Default in application class, if classes and singleton are empty sets, the runtime will scan and register all services using (per request behavior) and providers (once in app).
  + Clients Manager generic class should be used to separate client proxies from the rest technology classes.

## Business Layer

### Logic and Operations

* Business logic and operations should follow SOA (Service Oriented Architecture / Microservices) concepts.
* Utilities operations should be implemented in a separate package to be accessed by all other layers.
* Naming Conventions:
  + Class and Interface name should be a noun, and in mixed case with the first letter of each internal word capitalized (i.e., ItemManagement).
  + Use the compound nouns plural conventions in Classes (EmployeeVacations not EmployeesVacations).
  + Append “Business” in business class name (i.e., EmployeesBusiness, EmployeeDependentsBusiness).
  + Method name should start with a verb, in camel case (i.e., getItemData).
  + Variable name should be a noun, verb, letter as index for loop ONLY, in camel case (i.e., itemData).
  + Constant name should be all uppercase with words separated by underscores (“\_”) (i.e., MIN\_AGE).
* The business class is:
  + Singleton design pattern.
  + Stateless.
  + Static based methods or regular methods in singleton object initialized and injected using IOC.
* Business class methods are:
  + Public for use by other business classes or layers.
  + Protected for inheritance only.
  + Private for local use.
* Every business class is responsible for:
  + Group of related functionalities based on the single responsibility principle (Business Lead Responsibility).
  + Group of entities (every entity is managed by only one business class but could be used by many).
* If the business class manages operations on more than one entity, then it should organize the operations around the entities. The business class should be divided into regions, every region is dedicated for one entity and its structure is as follows:
  + Operations section (manage – add – modify – remove) and supporting methods (calculate – execute - …).
  + Validations section (validate).
  + Inquiries section (get – search).
  + Reports section (getXyzReportData).
* Business class inquiries:
  + Every query or queries will be handled only in one business method.
  + This method will be used by any other business methods that need the query results.
  + For example, if we have a query that get items information.
    - This query will be executed by one private method “**search**Items”.
    - This method will be used by wrapper methods in the same class called (**get**ItemById – **get**Items – **get**InvalidItems – …).
    - The wrapper methods will be used by any business method that needs the information.
    - If the query will be used in only one scenario, so the business method will be the owner and will be named **get** for example “**get**Items”.
  + This convention empowers code reusability. Wrappers are used for this method to execute it with different parameters.
  + If the method should return a List, it should return empty Array List in case of no data found.
  + If the method should return a single object, it should return null in case of no data found.
* For huge data inquiries, we will use database pagination.
* For all data inquiries, we will always return “Data Objects” unless there is no one.
* Business operations signatures will use the “Data Objects” but when passing to the DAL use the “Original Objects” that are calculated on the fly from the “Data Objects”. When something changes in the “Original Objects” by the DAL (Mostly the ID) the business operation is the responsible to reflect that change in the “Data Objects”.
* We should load the objects before updating them in case of partial update business operation. In the case of full update operation, we can use the same received entity as we will check full attributes.
* All business class transactional methods should receive the user for auditing and logging.
* Logging and Exception Handling:
  + All business methods should use the log method in the “Logging Util” for logging even if the current logging methodology is the default logging file of the AS. This will lead to an easy way to alter the logging behavior without any effect on the business classes.
  + Business class will return error codes via customized exceptions (any error or violation will be handled via exceptions otherwise; transactional methods will return void or the object in hand).
  + A general “Business Exception” should be designed for handling errors that doesn’t have another customized exception. All customized exceptions should inherit from the “Business Exception” which utilizes the usage of parameterized message and contains the error code that is mapped one to one with error message.
  + Log exception stack trace for general and unexpected exceptions only (any exception other than the customized exception). This behavior will be implemented via a parameter in the log method.
* Empower the usage of enumerators and constants to avoid using hard coded values (high priority for enumerators).
* We should minimize the comments as minimum as possible and maximize the code readability via selecting a clear describing name and a clear method structure.
* No nested transactions will be allowed.
* In some cases, we will need to add nested transactions behavior (after discussing with the team). In this case we will utilize threading to achieve the needed behavior (please refer to the section of hibernate session behaviors).
* A typical business class transactional method should unify its structure as the following example and any violation should be discussed with the team:
  + For application managed transactions:

|  |
| --- |
| public void/Entity/List<Entity> doSomeThingTransactional(…) throws BusinessException {  // All validations go here. (Validation methods throw business exception when detecting errors)  try {  // Begin Transaction  .  // logic and other calls external or internal (private or public)  .  // Commit Transaction  } catch (Exception e) {  // Rollback Transaction.  // Rollback Entities attributes changes if any. (In case of returned entities)  if (e instanceof BusinessException)  throw (BusinessException) e;  // Log with printing stack trace and user.  throw new BusinessException(ErrorCodesEnum.ERROR\_GENERAL);  }  }  private void/Entity/List<Entity> doPartOfSomeThingTransactional(…) throws BusinessException {  .  // logic and other calls external or internal (private or public)  .  } |

* + For container managed transactions:

|  |
| --- |
| @Supports Transaction // to enable controlling exception after rollback  public void/Entity/List<Entity> doSomeThingTransactionalWrapper(…) throws BusinessException {  // All validations go here. (Validation methods throw business exception when detecting errors)  try {  .  // call to doSomeThingTransactional().  .  } catch (Exception e) {  // Rollback Entities attributes changes if any. (In case of returned entities)  if (e instanceof BusinessException)  throw (BusinessException) e;  // Log with printing stack trace and user.  throw new BusinessException(ErrorCodesEnum.ERROR\_GENERAL);  }  }  @Required Transaction  private void/Entity/List<Entity> doSomeThingTransactional(…) throws BusinessException {  .  // Actual transactional logic management  // logic and other calls external or internal (private or public)  .  }  @Mandatory Transaction  private void/Entity/List<Entity> doPartOfSomeThingTransactional(…) throws BusinessException {  .  // logic and other calls external or internal (private or public)  .  } |

* In spring AOP; transactional annotation properties only take place if the method call done through the service object (method call another method in the same service won't work). We can overcome this issue by using AspectJ which will inject a proxy around every method call in case we decided to use CMT. Also, we need to use transactional annotations at all methods to overcome the problem of private methods calling.  
  <https://www.geekyhacker.com/2020/03/28/how-to-configure-aspectj-in-spring-boot/>
* For CMT, use public orchestration methods that handle the exceptions with support transaction and private methods for transaction execution with required transaction.
* We can inject spring managed services into non-managed java classes (i.e., JAX-WS, JAX-RS or any static java class)  
  WebApplicationContextLocator.getCurrentWebApplicationContext().getAutowireCapableBeanFactory().autowireBean(endPoint);

### Workflow

* The workflow is based on the popular idea of processes, instances and tasks.
* Our workflow is a property to our team and based on:
  + Entities design to manage the workflow needs:
    - Process Group (Business grouping for each collection of processes).
    - Process (Represent a business process or a workflow of human interactions).
    - Process Step / Position (Used for determining the workflow interactions).
    - Instance (Concrete object of the process).
    - Task (concrete step that belong to an instance).
    - Delegation (To manage the total and partial delegations).
    - Business WF entities (to handle the data managed by the workflow).
  + Base Workflow class that manages all the above entities except the business WF entities.
  + General Steps Workflow class that manages the workflow interactions in a configured and general methodology (steps could be defines and configured and this class run these steps).
  + Business Workflow classes that manage complex workflow interactions.
* Strive for the separation between Business Workflow classes and Business Classes (One way to always control by work flow and prevent calling the workflow from business classes).
* Generic Inbox, Outbox, Delegation and Steps Management pages should be built to enable the WF users to view and manage their instances and tasks.
* Composite components should be used:
  + Current task (info – notes - refuse reasons).
  + Previous tasks (history for previous tasks along with their actions).
* Version control should be there on the task level to avoid concurrent access on the same task.
* A status for the instance should differentiate between (Running – Done – Completed) instances.

## Data Access Layer

### Data Access Management

* The target is to follow the standard of JPA “Java Persistence API”, so we will limit the use of hibernate to the minimum (i.e., only session manipulations).
* Data access methodologies in our preference order:
  + Session factory along with application managed transactions.
  + Entity manager factory along with application managed transactions.
  + Session factory along with container managed transactions (declarative).
  + Entity manager factory along with container managed transactions (declarative).
  + Single repository (crud repository based) along with container managed transactions (declarative). Single repository that uses a generic spring repository based on “Common Entity” for entity operations and a generic method for inquiries.
  + Recommended spring methodology (single repository for each entity).
* The “Repository Manager” class is the core of the data access layer. It will handle:
  + All the ways that will lead to the database.
  + The configuration of the Session Factory / Entity Manager Factory:
    - Native bootstrapping by building the factory inside the “Repository Manager” class.
    - JPA bootstrapping by injecting the factory inside the “Repository Manager” class.
  + Application Managed Transaction Operations (begin – commit – rollback – flush).
  + Entity Operations (insert – update – delete).
  + Execute Queries (Named Query – Dynamic Query – Native Query).
  + Database pagination and locking.
  + Database Objects reading/calling (Avoid – No or Min usage):
    - Reading Sequences.
    - Calling function or procedure.
  + Auditing.
* Entities operations:
  + Generic and based on Base Entity interface (The marker interface for all entities).
  + Knows nothing about the entity type or content.
  + Used only from business services.
* Custom Session / Custom Entity Manager: The “Decorator” that will assign the transaction owner (the business operation that owns the transaction) and store the transaction into the current thread. It will be used only by the “Repository Manager” class and will help in application managed transaction management. This will introduce the transaction control needs for the business layer and in the same time will isolate it from the actual used technology.
* Prevent nested transactions. This will be achieved via using the same transaction for all sub methods to always have one start and one commit per any request. This convention eliminates entirely nested transactions problems. In the situations that will need multiple transactions, we will achieve this by utilizing threading.
* Data Access Layer will throw a custom exception called “Repository Exception”. This exception will be thrown when something goes wrong in the data access layer such as constraint violation. This exception should not be thrown from any other layer but should be caught in the business layer, processed and wrapped in another exception “Business Exception”.
* If needed, JDBC Connections will be used in the application only via the sessions.
* For default schema configuration, add default schema name to:
  + Main configuration table to be used where needed (i.e., for reporting).
  + “hibernate.cfg.xml” for "hibernate.default\_schema" property or “application.properties” for "spring.datasource.schema" property.
* Entity operations in hibernate session vs entity manager:

|  |  |  |
| --- | --- | --- |
| **Operation** | **Hibernate Session** | **Entity Manager** |
| Insert Entity | * session.save(entity) * Generate insert statement. | * em.presist(entity) * Generate insert statement. |
| Update Entity | * session.update(entity) * Generate one update statement even it was called multiple times. * Updating non-existing entity will fail. | * em.merge(entity) * Generate one update statement even it was called multiple times. * For non-managed entity, a select statement is issued before the update and a new managed entity with a different reference will be returned. * Updating non-existing entity will insert new entity with new identifier. |
| Delete Entity | * Native bootstrapping: session.delete(entity) * JPA bootstrapping: session.delete(session.merge(entity)) * Generate delete statement. * For non-managed entity, a select statement is issued before the delete (either implicit in case of native bootstrapping or explicit by calling merge in the JPA bootstrapping). | * em.remove(em.merge(entity)) * Generate delete statement. * For non-managed entity, a select statement is issued before the delete. |

* It is not recommended to update an entity by calling the update entity after inserting it within the same transaction as it will generate update statement after the insert. We will always try to insert the new entities after finalizing its data.
* It is not recommended to update a managed entity (either after inserting it, updating it or selecting it using the same transaction) by modifying its content without calling the update operation as it will generate update statement after the insert, harm the business clarification / code readability and make it harder to change the data access technology. We will always try to insert the new entities after finalizing its data.
* It is not recommended to perform multiple entity operations on the same object with the same transaction – although we can – instead try to perform the entity operation after finalizing its data.
* Don’t perform entity operations on two objects with the same identifier in the same transaction.
* Don’t perform insert entity operation for an existing entity as it will fail.
* Don’t perform update entity operation for non-existing entity (refer to the above table).
* Don’t perform delete entity operation for non-existing entity as it will not fail because it will insert and delete the same entity. Make sure that the record exists before delete.
* Merge doesn't do anything for managed entities, otherwise a select statement is issued for every non-managed entity, then the values of the non-managed entity will override the values of the newly selected managed entity which will have a different reference.
* We can eliminate the need for select statement in case of non-managed entity operations: update (Entity Manager only) / delete (Entity Manager and Session) by selecting it using the same entity manager / session. But this will lead to flush operation with every select statement, also this may lead to unwanted business behavior as the retrieved managed object will be saved to the DB if any change happened to it even without calling save or update that's why we don't use the same entity manager / session in reading.
* Batching:
  + The default batching configuration is disabled.
  + Enable batching when needed per transaction based on batching configuration. (It could be implemented via sending the batch configuration as a parameter to “beginTransaction”).
  + Before using order\_insert and order\_update always compare the performance against not using them with the correct order implemented in business.
  + We can use bulk update and delete statements on business need.
* There is a problem in using hibernate 5 native bootstrapping with spring devtools (don’t use devtools if we decided to use native bootstrapping).

### Queries

* JPA standard defines three ways to handle queries: Named Queries, Dynamic Queries and Native Queries.
  + Avoid the usage of native queries to stay with the database independence.
  + Avoid the usage of dynamic queries and strive for the named queries for the following reasons:
    - Improve the code organization by separating the query structure from the business logic.
    - Empower the encapsulation concept. It let us handle the query as a black box, only I gave it inputs to get from it outputs and never concern about the query internal structure.
    - Enforce the use of query parameters instead of embedding them dynamically that result in more secure queries.
    - More maintainable while refactoring than maintaining scattered queries in the business layer code.
    - The queries are validated and compiled once at the Context initialization. That will definitely enhance the overall performance by avoiding validating and compiling the queries every time we execute them. Also, this will prevent the execution failure as If there were any errors it will be caught at startup.
    - We can replace them or restructure them to enhance their performance without any change in the business layer. They help to have a good analysis of any entity usages in one shot of the associated named queries.
* Query names:
  + Should be defined in a class that holds only query name constants. These constants should be used in the definition of queries and while referring to them.
  + Should have prefixes with module and entity name “OwnerSystem\_EntityName\_functionality”.  
    **public** **static** **final** String ***MODULE\_ITEM\_GET\_ACTIVE\_ITEMS*** = "MODULE\_ITEM\_GET\_ACTIVE\_ITEMS".
* Queries and Parameters:
  + Always analyze the query to get the most optimized performance (look for joins, inner queries …).
  + All named queries that retrieve more than one record should have order by the business need; otherwise order by its id.
  + The dates should be passed as strings and converted in the query with the same used format.
  + When passing an array to a query, take care to use the toArray(new T[original List.size()]) instead of toArray().  
    For example: qParams.put("P\_TASKS\_IDS", tasksIds.toArray(new Long[tasksIds.size()]));  
    Instead of qParams.put("P\_TASKS\_IDS", tasksIds.toArray());
  + When passing an array as a parameter to a query and this array contains many values (1000 or more), you should use the “getEntitiesWithLargeArray” operation from the “Repository Manager” class. Order is your responsibility in that case.
  + Do not use count (\*), always count with ids.
  + Do not use (NOT IN) other than with static values. Instead use count or not exists.  
    For example:  
    Don’t use: select \* from child where child.parent\_id not in (select id from parent where …)  
    But use : select \* from child where (select count(id) from parent where …) = 0  
     or select \* from child where not exist (select \* from parent where …)
  + Always use alias before referring to attributes even if there is only one entity in the query.
  + For highly used queries, design them as separate queries. But for queries that share the same returned entities and have different parameters, design them as a single query and use (P\_ESC\_SEARCH\_STR, P\_ESC\_SEARCH\_INT, P\_ESC\_SEARCH\_LONG and P\_ESC\_SEARCH\_DOUBLE) flags to escape “WHERE” clauses.
  + Use P\_DATE\_FLAG and P\_ARRAY\_FLAG with dates and arrays.
  + When comparing dates:
    - If you are comparing dates from the database versus others in the database so you can use the date attribute directly.
    - If you are comparing dates from the database versus application parameters so you must use to\_date while comparing.  
      (:P\_DATE\_FLAG = :P\_ESC\_SEARCH\_LONG or item.issueDate <= to\_date(:P\_DATE,'MI/MM/YYYY')).
  + The parameter data type should be the same type as the entity attribute.

|  |
| --- |
| qParams.put("P\_DECISION\_NUMBER", (decisionNumber == **null** || decisionNumber.length() == 0) ? FlagsEnum.ALL.getCode(): decisionNumber); // At the service, this will give an exception because getCode() returns integer value, so an empty string should be concatenated with it.  qParams.put("P\_DECISION\_NUMBER", (decisionNumber == **null** || decisionNumber.length() == 0) ? FlagsEnum.ALL.getCode() + "": decisionNumber); // correct |

* + For constants in queries (-1, ‘-1’, ‘ABC’, etc.), make sure that the constant represents the same data type as the field being compared to. Also, make sure not to use a constant and use a query parameter instead in case of the query parameter in comparison with the constant is used in another comparison with other field.

|  |
| --- |
| (:P\_NAME = ‘-1’ or b.name like :P\_ NAME)  // Raise an exception in case that P\_NAME value size more than 2 in some databases like DB2.  (:P\_ NAME = :P\_ESC\_SEARCH\_STR or b.name like :P\_ NAME)  (:P\_ AMOUNT = -1 or b.totalAmount >= :P\_ AMOUNT)  // Raise an exception in case that P\_Amount value than INTEGER maximum value in some databases like DB2.  (:P\_ AMOUNT = :P\_ESC\_SEARCH\_DOUBLE or b.totalAmount >= :P\_ AMOUNT) |

* In case of multiple database vendors in the same project, do not use the ready-made database functions otherwise use custom functions designed in a unified fashion and written against every database.

### Hibernate Session Behavior

* Hibernate sessions and JTA transactions by default - and this is our methodology also – tracks the database transactions against the threads which initiate the transactions. Always, one and only one DB transaction is active within the thread at any point of time (All fail or all success and also one time commit whatever how many commit statements are called on different JTA transactions).
* Therefore, when there is a need for two independent DB transactions, we need to handle each one at a separate thread of execution. If we need these two independent DB transactions to be run sequentially, we can use the “thread.join()” API as shown at the following example:

|  |
| --- |
| **public** **static** **void** save() **throws** Exception {  CustomSession session1 = DataAccess.*getSession*();  session1.beginTransaction();  Country c1 = **new** Country();  c1.setName("ar name 1");  DataAccess.*addEntity*(c1, session1);  Runnable myRunnable = **new** Runnable() {  **public** **void** run() {  **try** {  CustomSession session2 = DataAccess.*getSession*();  session2.beginTransaction();  Country c2 = **new** Country();  c2.setName("ar name 2");  DataAccess.*addEntity*(c2, session2);  session2.commitTransaction();  session2.close();  } **catch** (Exception e) {  e.printStackTrace();  }  }  };  Thread thread = **new** Thread(myRunnable);  thread.start();  thread.join();  session1.commitTransaction();  session1.close();  } |

### ORM

* Entities Structure:
  + “BaseEntity” is a marker interface for all of the ORM entities to enable the “Repository Manager” class to deal with them seamlessly.
  + “CommonEntity” is a base class (@MappedSuperclass) for all our system entities. It implements the “BaseEnitity” interface and contains the common attributes (insertionDate and lastUpdateDate) and behavior (@PrePersistand @PreUpdate).
  + “AuditeeEntity” is a base class for all our auditable system entities. It extends the “CommonEntity” and contains the attributes and behavior related to auditing.
  + Our system entities should extend either “AuditeeEntity” or “CommonEntity” based on either the entity is subject to auditing process or not. External entities to our system which mainly will represent integration entities should implement “BaseEnitity” interface.
  + “BaseEntity”, “CommonEntity” and “AuditeeEntity” should be in the “dal.entities.base” package and other entities should be organized in packages that represent the domain structure.
* Entity name should:
  + Reflect the table name but in singular (i.e., Item, ItemComponent, …).
  + Reflect the view name but in singular with appending “Data” to the name (i.e., ItemData, ItemComponentData).
* Entity Structure:
  + Package and import statements.
  + Named queries:
    - Only queries that select/count/… the same entity or a group of entities in which our entity is the main entity relevant to the query logic should be placed in the entity file.
    - Queries should be organized according to their common logic (look first if the needed query is already existed, if not exist check if you can modify an already existing one to handle your case, if not think where to place your new query).
    - Name the query via the constants mechanism described before.
    - @NamedQueries({

@NamedQuery(name = QueryNames.***MODULE\_ITEM\_GET\_ITEM\_BY\_ID***,

query = "select i from Item i where i.id= :P\_ID"),

@NamedQuery(name = QueryNames.***MODULE\_ITEM\_\_GET\_ITEMS***,

query = "select i from Item i Order by i.name")

})

* + Use “Lombok” annotation:
    - @Data to generate the getters, setters, constructor, toString, hashCode and equals methods. We can override the generated methods by just implementing them by ourselves.
    - @EqualsAndHashCode(callSuper = false) if your entity extends from another entity.
  + Use JPA annotation @Entity and @Table annotations to mark the entity as a JPA entity and to specify the table or view name in the database.
  + Attributes:
    - Make sure that the order is the same as the columns in the database table.
    - Data types should be wrapper classes instead of primitive types as they may be null.
    - Use filed access annotation.  
      <https://thorben-janssen.com/access-strategies-in-jpa-and-hibernate/>
    - Use @Column annotation to specify the column name or use @Transiant annotation if the field cannot be persisted.
    - Use @Temporal (TemporalType.TIMESTAMP) annotation with dates.
    - Separate attributes by empty lines.
    - Id Attribute:
      * Use @Id annotation to specify the primary key. (All our system entities should follow single PK)
      * Use @IdClass annotation for mapping of complex PK. (Only allowed for external entities)
    - Id Generation (Any strategy should allow multiple application servers to generate non conflicting ids):
      * Sequence generation: Id values will be based on database sequences designed around related table groups. Set the allocation size to 1 as the default value is 50 in some databases.  
        @SequenceGenerator(name="ModItemsSeq", sequenceName="MOD\_ITEMS\_SEQ", allocationSize=1)

@GeneratedValue(strategy=GenerationType.SEQUENCE, generator=" ModItemsSeq")

* + - * UUID.
      * Custom Hibernate Strategy.
  + Methods:
    - If needed override any of Lombok generated methods.
    - Empower the usage of calculated values in data entities. (i.e., date string representation, number calculations, flags, …) through introducing additional methods in the entity.
    - For data entities, add another method to get the wrapped representation of the original entity by calculating it on the fly upon any request for it.
    - For auditee entities, override “calculateContent” if needed and implement “calculateContentId”.
* Entities should be developed via JPA annotations not hibernate annotations.
* No mapping for relationships. This will simplify the generated database queries and efficiently reduce the size of un-needed data retrieval and finally will enhance the overall performance.
* Empower the usage of Data Objects (mainly based on a database view and designed to cover all the needed data from other entities for one entity).

### Reporting

* Reports names along with its relative path should be defined in “ReportNamesEnum” enumerator with the following naming convention:
  + Many conventions could be adopted:
    - [Group Name as in DAL package name]\_[DECISION\_] {0..1}[CATEGORY\_]{0..1} [COLLECTIVE\_]{0..1}[DESCRIPTION]{0..1}
    - [Group Name as in DAL package name]\_ [DESCRIPTION]
  + Files names should be the same as the enumerator.
  + The file extension should be in upper case (.JRXML) not in lower case (.jrxml) as the Linux OS is case sensitive regarding the files names including the extension.
  + For **sub reports**, file extension should be in lower case (.jrxml) not in upper case (.JRXML) as jasper engine always search for sub reports with lower case extension.
* Report Manager:
  + Generic class used for exporting all of our reports in all supported reporting types.
  + Should be accessible only from the business layer.
  + Should contain absolute path for the reports folder in the application server (Read from Configuration Utility).
  + Should contain the default schema (Read from Configuration Utility).
  + Should contain method for each supported reporting type.
  + Every method should accept the parameters:
    - Report relative path including the report name.
    - Report parameters map.
  + This class auto compiles sub reports.
* Reports Root:
  + Reports will be in an outside folder (Reports Root) for better maintenance.
  + There will be a configuration available via the configuration utility to specify the reports root path.
  + All reports will have P\_REPORTS\_ROOT parameter to use it for referencing its resources (images, sub-reports, fonts …) and passed to the reports from the Report Manager.
* Schema name:
  + All reports and sub reports will have P\_SCHEMA\_NAME parameter and passed to the reports from the Report Manager.
  + Use the schema name before any table, view, or function name. Write the name using $P!{P\_SCHEMA\_NAME} so the query will use the parameter value without interpreting it.
  + Pass the Schema Name parameter to the subreports like the DECISION\_SIGNATURE subreport.
  + Set a default value to the schema name parameter so the compiler will be able to compile the query and read the fields while developing the report.
* Sub reports:
  + When passing return value from a sub report to another sub report, the two sub reports should be separated in two bands for the order of execution.
  + Common fields within the same report and distributed in sub reports should be passed between the sub reports.
* Dates:
  + For All dates (including print dates), the reports will receive it as string parameter from the application.
  + Use TO\_CHAR (date\_to\_be\_formatted, 'fmMI/MM/fmYYYY') to select a well and standard formatted date. Note that ‘fm’ (fill mode) is used to remove the leading zeros from the date (e.g, 01/03/1434 will be 1/3/1434).
* Export the used fonts as jars from IReport and include it in the class path.
* Add the connectivity file (like sqljdbc4-2.0.jar) to the iReport classpath.
* If you have a report with table that will take several pages and you don’t want the rows to split on pages If the value of some column is too big to fit in at the end of the page, you can set the splitType property of the band which the table in to splitType = Prevent (You can find it in the Properties panel if you choose the band from Report Inspector panel).
* Tips for using while reports developing:
  + Use String.indexOf(“”) > -1 instead of String.contains(“”) due to java runtime changes.
  + If you have a field contains an inline if expression which return different datatypes, you should cast anyone of them to make both return the same data type.  
    Example: ($P{P\_CATEGORY\_ID} == 1) ? $F{MILITARY\_NUMBER} : $F{JOB\_CODE} WRONG  
    ($P{P\_CATEGORY\_ID} == 1) ? $F{MILITARY\_NUMBER}.toString() : $F{JOB\_CODE} CORRECT
  + Don’t use String.replace(“”,””) at the report expression, instead try to do all the replacements at the query.
  + Don’t define any string at single quotes ‘’, instead use double quotes.  
    Example: ($F{PH\_UNIT\_FULL\_NAME} != null ? $F{PH\_UNIT\_FULL\_NAME} : '-') WRONG  
     ($F{PH\_UNIT\_FULL\_NAME} != null ? $F{PH\_UNIT\_FULL\_NAME} : "-") CORRECT
  + Always write queries, tables, names, columns and aliases in upper case. Take care of case sensitivity in some databases.
  + Use “CASE WHEN” instead of “DECODE”.
  + The tables’ columns to be compared should be with the same data type.
  + Use OUTER JOIN convention instead of (+) join operator.
  + In case of multiple database vendors in the same project, do not use the ready-made database functions otherwise use custom functions designed in a unified fashion and written against every database.
* The returned report byte array to the presentation layer should be handled into the response in a centralized methodology (i.e., BaseManagedBean.jsf).

### Auditing

* Solution Description:
  + A table called “UM\_AUDIT\_LOGS” which will contain:
    - ID.
    - Module ID for logging against many projects and/or modules.
    - User ID.
    - System name in case the transaction was made automatically via a system.
    - Operation (Insert – Update – Delete).
    - Operation Gregorian date.
    - Content entity for the entity manipulated by the operation.
    - Content ID for the row ID in the corresponding content entity table.
    - Content, JSON object contains the entity data.
  + An entity called “AuditLog” which is the ORM entity corresponding to the audit table.
  + An entity called “AuditeeEntity” which is used as a base class for any entity that should be audited. One abstract method in this class to enforce child entities to calculate its content ID. One default method to calculate the content. Also, one flag used to prevent specific transaction from being audited.
  + The entity operations at the “Repository Manager” class always call the audit method which audits the entity in case that the entity is an auditee and there is no prevention for this transaction to be audited.
* Solution Usage:
  + Make the entity that you are interested in extends the “AuditeeEntity” entity.
  + Implement “calculateContentId” and override “calculateContent” if needed. The former one used to return the identifier of the entity and the later one used to return the values that you are interested in (the auditee entity will audit all the fields by converting it to JSON by default).
  + Always pass the user / system as string to the “Repository Manager” class entity operations and set the entity prevent audit flag in case you do not want to audit a specific transaction.

## Data Repository Layer

### General

* Database should be business free. It should not contain any functions, triggers or procedures. This rule may be violated in certain situations AFTER discussions.
* In case of multiple databases in the same project, do not use the ready-made database functions otherwise use a custom function designed in a unified fashion and written against every database.
* Always one serial id in each table will be better than using compound primary keys.
* Use the following keywords for joining multiple tables:
  + INNER JOIN
  + LEFT OUTER JOIN
  + RIGHT OUTER JOIN
  + FULL OUTER JOIN
* Naming conventions in tables, views, sequences and constraints should be as the following:
  + Table name should follow “OwnerSystem\_TableName”.
  + View name should follow “OwnerSystem\_VW\_ViewName”.
  + For integration views follow “OwnerSystem\_INTEG\_ViewName”.
  + Table name and view name should be in plural (i.e., ITEMS and ITEM\_COMPONENTS).
  + Table name and view name should not represent functionality or behavior otherwise it should reflect a repository description for the data it contains.
  + Sequence name should follow " OwnerSystem\_TableNameORTableGroupName\_SEQ".
  + Upper letters should be used in all objects unless in postgreSQL lower case letters is used.
* Hijri dates in database:
  + Hijri Dates stored in database in Dates/Timestamp columns for the simplicity and performance in manipulation and comparison operations.
  + In the Date/Timestamp data type there is no such 30/02/YYYY, so we store the field of Hijri days in the field of minutes. Days, seconds and hours in the date are don’t care and should be the same for comparison reasons.
  + So the format in database should be “MI/MM/YYYY” while in Java “mm/MM/yyyy”.
  + In SQL Developer, adjust the formatting of dates to be able to read the date correctly.  
    Tools -> Preferences -> Database -> NLS -> Date Format and Timestamp Format,  
    DD/MM/YYYY HH24:MI:SS.
  + Always save the Gregorian date with the Hijri date.

### Data Types

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Java | DB2 | Oracle | MS SQL Server | PostgreSQL |
| String | VARCHAR(n)  Use the double needed size for Arabic content  Store empty strings | VARCHR2(n)  Use CHAR not BYTE.  Converts empty strings to null | VARCHAR(n)  Store empty strings | character varying(n)  Store empty strings |
| Integer | INTEGER | NUMBER | Numeric(x,0) | Numeric |
| Long | BIGINT | NUMBER | Numeric(x,0) | Numeric |
| Double | DOUBLE | Number | Numeric(x,y) | Numeric |
| Boolean | DECIMAL(1) | NUMBER(1,0) | Numeric (1,0) | Numeric |
| Date | TIMESTAMP | DATE | Datetime2 | timestamp without time zone |
| Byte[] | BLOB | BLOB | Varbinary(MAX) | bytea |

* In case we are targeting multiple databases, while creating/replacing a view script, we should make sure the script valid for all the databases, also we should pay attention not to use PostgreSQL resultant view script for unifying other databases script as PostgreSQL generates additional syntax for data types in the view script.
* Scale and Precision:
  + You should refer to business for specifying scale and precision exact values.
  + The following are approximate values:
    - X: use 18 for default cases, especially for IDs.
    - Y: use 3 for default cases.
    - N: use 200 for default cases.
* Use setter methods to overcome the problem of storing empty strings in database:

**public** **void** setShieldMobileNumber(String shieldMobileNumber) {

**this**.shieldMobileNumber = "".equals(shieldMobileNumber) ? **null** : shieldMobileNumber;

}

### Table Creation

|  |  |  |
| --- | --- | --- |
| Database | Syntax | Example |
| DB2 | **CREATE TABLE [SCHEMA\_NAME].[TABLE\_NAME] ( [COLUMN\_NAME] [DATA\_TYPE] NOT NULL,  [COLUMN\_NAME] [DATA\_TYPE(SIZE)],  PRIMARY KEY ([COLUMN\_NAME])**  **)** | CREATE TABLE TAHSEELPOC.HCM\_ORG\_UNITS  ( ID **BIGINT** NOT NULL,  NAME **VARCHAR**(**200**),  PARENT\_UNIT\_ID **BIGINT**,  HKEY **VARCHAR**(**20**),  REGION\_ID **BIGINT**,  REMARKS **VARCHAR**(**1000**),  ORDER\_UNDER\_PARENT **INTEGER**,  ACTIVE\_FLAG **DECIMAL**(**1**),  CONSTRAINT PK\_UNITS PRIMARY KEY (ID)  ); |

### Constraints

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Type | Naming Convention | DB2 | Oracle | | MS SQL Server | PostgreSQL |
| Primary Key | **PK\_**[Abbreviated Table Name] | ALTER TABLE [SCHEMA\_NAME].[TABLE\_NAME] ADD CONSTRAINT [CONSTRAINT\_NAME] PRIMARY KEY ([COLUMN\_NAME])  ENFORCED; -- DB2 only. | | | | |
| Unique | **U\_**[Abbreviated Table Name]**\_**[ Abbreviated Column Name] | ALTER TABLE [SCHEMA\_NAME].[TABLE\_NAME]  ADD CONSTRAINT [CONSTRAINT\_NAME] UNIQUE ([COLUMN\_NAME])  ENFORCED; -- DB2 only  For DB2 and nullable columns:  CREATE UNIQUE INDEX [SCHEMA\_NAME].[INDEX\_NAME]  ON [SCHEMA\_NAME].[TABLE\_NAME]  ([COLUMN\_NAME] ASC )  **EXCLUDE NULL KEYS**; | | CREATE UNIQUE NONCLUSTERED INDEX constraint\_name ON schema\_name.table\_name(column\_name) WHERE column\_name IS NOT NULL;  Note that: In earlier versions of MS SQL server, when defining a unique constraint, a single NULL is allowed but multiple NULLs are not. Fortunately, In SQL Server 2008, we can overcome this by defining a unique filtered index based on a predicate that excludes NULLs. | | ALTER TABLE table\_name ADD CONSTRAINT constraint\_name UNIQUE (column\_name); |
| Foreign Key | **FK\_**[Abbreviated Table Name]**\_**[Abbreviated referenced Table Name]**\_**[ Abbreviated Column Name] | ALTER TABLE [SCHEMA\_NAME].[TABLE\_NAME]  ADD CONSTRAINT [CONSTRAINT\_NAME] FOREIGN KEY ([COLUMN\_NAME])  REFERENCES [SCHEMA\_NAME].[TABLE\_NAME] ([COLUMN\_NAME])  -- The following for DB2 only.  ON DELETE RESTRICT ON UPDATE RESTRICT ENFORCED  ENABLE QUERY OPTIMIZATION; | | | | |
| Check | **C\_**[Abbreviated Table Name]**\_**[Abbreviated Column Name]**\_**[ Abbreviated Action] | ALTER TABLE [SCHEMA\_NAME].[TABLE\_NAME]  ADD CONSTRAINT [CONSTRAINT\_NAME] CHECK ([CHECK CONSTRAINT CONDITION])  ENFORCED ENABLE QUERY OPTIMIZATION; -- DB2 only. | | | | |
| Not Null |  | ALTER TABLE [SCHEMA\_NAME].[TABLE\_NAME] ALTER COUMN [COLUMN\_NAME] SET NOT NULL; | ALTER TABLE [TABLE\_NAME] (COLUMN\_NAME NOT NULL); | ALTER TABLE [SCHEMA\_NAME].[TABLE\_NAME] ALTER COUMN [COLUMN\_NAME DATA TYPE] NOT NULL; | | ALTER TABLE [SCHEMA\_NAME].[TABLE\_NAME] ALTER COUMN [COLUMN\_NAME] SET NOT NULL; |

### DB2 Installation Structure

* The following diagram represents DB2 database installation structure.

**DB2 Product**

Instances….

**Instance**

DBs…

**Database**

**Schema**

Schemas…

* In DB2, you will need to commit or rollback manually after any DDL statement.

### DB2 Built-In Scalar Functions

|  |  |  |
| --- | --- | --- |
| Function | Example | Notes |
| CONCAT | SELECT **CONCAT**(str1, str2) FROM SYSIBM.SYSDUMMY1; |  |
| LENGTH | SELECT **LENGTH**(str) FROM SYSIBM.SYSDUMMY1; |  |
| NVL | SELECT **NVL**(param1, param2) FROM SYSIBM.SYSDUMMY1; |  |
| REPLACE | SELECT **REPLACE**(str, pattern, replacement) FROM SYSIBM.SYSDUMMY1; | If replacement is null, it returns null |
| SUBSTR | SELECT **SUBSTR**(str, startPosition, subStringLength) FROM SYSIBM.SYSDUMMY1; | stratPosition starts with 1 |
| TO\_CHAR | SELECT **TO\_CHAR**(param, pattern) FROM SYSIBM.SYSDUMMY1; |  |
| TO\_NUMBER | SELECT **TO\_NUMBER**(str) FROM SYSIBM.SYSDUMMY1; |  |
| TO\_DATE | SELECT **TO\_DATE**(str, pattern) FROM SYSIBM.SYSDUMMY1; |  |
| MOD | SELECT **MOD**(param1, param2) FROM SYSIBM.SYSDUMMY1; |  |
| TRIM | SELECT **TRIM**(**L** trimConstant FROM str) FROM SYSIBM.SYSDUMMY1; | B: Both L: LEADING T: TRAILING The default is: B |
| LPAD | SELECT **LPAD**(str, num, pad) FROM SYSIBM.SYSDUMMY1; |  |
| RPAD | SELECT **RPAD**(str, num, pad) FROM SYSIBM.SYSDUMMY1; |  |
| ADD\_MONTHS | SELECT **ADD\_MONTHS**(date, monthesCount) FROM SYSIBM.SYSDUMMY1; |  |
| MONTHS\_BETWEEN | SELECT **MONTHS\_BETWEEN**(date1, date2) FROM SYSIBM.SYSDUMMY1; | date1 – date2 |
| NEXTVAL | VALUES NEXTVAL FOR HCM\_SETUP\_SEQ; |  |

### DB2 Indexes Creation (DBAs)

When you create a table that contains a primary key or a unique constraint, you must create a unique index for the primary key and for each unique constraint. DB2 marks the table definition as incomplete until the explicit creation of the required enforcing indexes, which can be created implicitly depending on whether the table space was created implicitly, the schema processor, or the CURRENT RULES special register. If the required indexes are created implicitly, the table definition is not marked as incomplete.

## Security

* Authentication and Authorization validations should be handled by the presentation layer and the business layer will deal with the presentation layer as a trusted zone.
* Web Filter should be used for securing the application (implementing authorization).
* A generic Security Manager will handle:
  + Authentication: mainly against an identity management product throw LDAP through Security Utility Class.
  + Authorization:
    - User access from the UI channels: User menus and menu actions should be loaded upon login and saved in the user session then used to verify authorization in the security web filter and managed beans.
    - User access from external channels (like mobile): Session management solution should be implemented and maintained.
    - System access from the same organization network:
      * Empower trusted zones and allow the clients for the target services for certain IPs only.
      * Use a security token saved in database and granted for read to the external system. The token should change with every access.
    - System access from external environment: Security certificates should be used.
* Use SSL for data integrity and confidentiality.
* Make sure of proper usage of URL parameters (always check for authorization [specially in managed beans] if the page data depends on an URL parameter).
* Invalidate the user session after log out and do not enable caching for the data.
* Watch for the following security vulnerabilities:
  + Turn off password autocomplete.
  + ClickJacking attack: Set the response header: (“X-Frame-Options”, “DENY”).
  + MIME-sniffing attack: Set the response header: (“X-Content-Type-OPTIONS”, “nosniff”).
  + Cross Site Scripting:
    - Set the response header: (“X-XSS-Protection”, “1;mode=block”).
    - Set cookie HttpOnly flag to true. This will not allow the cookie to be accessed via a client side script such as JavaScript.
  + Web Content Caching: Set the response headers:  
    (“Cache-Control”,”no-cache, no-store, must-revalidate”).  
    (“Pragma”, “no-cache”).  
    (“Expires”, 0).  
    We can exclude JavaScript resources or all JSF resources from this point.
  + Secure Cookies: Set cookie secure flag to true. Only sent the cookie if the request is being sent over a secure channel such as HTTPS.
  + Always develop general error page to prevent application error disclosure.

# Project Structure

## Projects and Packages design

* When the layers separation realized by concepts in one project, you should take the union of the following packages. In case real segregation was adopted then follow the structure below.
* One project for each presentation component:
  + appconfig (for application Contextlistener and any other app configuration).
  + business (for security).
  + integration (config – entities – entities/workflow – requests – responses – clients).
  + ui (components – converters – filters – util – managedbeans [base – home]).
  + WEBContent (resources [code – css – fonts – images – javascript] – WEB-INF).
* One project for each business component (may be one shared project for common services):
  + appconfig (for application Contextlistener and any other app configuration).
  + dal (entities – entities/workflow).
  + business (workflow).
  + integration (config – entities – entities/workflow – requests – responses – services – clients).
* One project for core non business functionality (all projects will depend on it):
  + resources (if any).
  + enums (all enumerators).
  + exceptions.
  + dal (Custom Session – Data Access – Report Executer).
  + dal/entities (audit – config – hijri – workflow).
  + business (config – workflow).
  + business/util (Resource Bundle Util – File Util – Logging Util – Content Util [XML and JSON utilities] – Report Util – Basic Util [Java Basic operations and collections] – Multi Chronology Calendar Util – Security Util – Spreadsheet Util).
  + integration (requests/Request Metadata – responses/Response Metadata – clients/Clients Manager).

\* All packages prefixed with com/code.

\* integration/config for applications, filters, interceptors, providers, handlers … classes.

\* Integration clients in case of SOAP, each client has a separate package for the generated files and a wrapper client class outside this package.

## Configuring Projects in Eclipse

* For core project:
  + Make a new **Java Project** (name: core).
  + Make sure from the execution JRE (i.e., JRE/JDK 11) and Java Project Facets (i.e., 11).
  + Uncheck “create module-info.java file”.
* For business component project:
  + Make a new **Dynamic Web Project** (i.e., Billing).
  + Make sure from the target runtime (i.e., WebLogic).
  + Make sure from the execution JRE (i.e., JRE/JDK 11) and Java Project Facets (i.e., 11).
  + Uncheck “Add project to an EAR”.
  + Name the source folder on the build path “src”.
  + Name the default output folder “WebContent\WEB-INF\classes”.
  + Name the context root as the project name and the content directory “WebContent” then check “generate web.xml”.
* For presentation layer project:
  + Follow the same business component project steps.
  + Add JSF and Primefaces JARs to (\WebContent\WEB-INF\lib) and add them to the project class path libraries (If only one UI war used is used otherwise follow the EAR methodology).
* For the Development EAR:
  + Make a new **EAR Application Project** (i.e., tahseelDev).
  + Make sure from the target runtime (i.e., WebLogic).
  + Choose EAR modules (projects).
  + Write in “Content directory” (EarContent).
  + Check “generate application.xml”.
  + Add your common JARs in (\EarContent\APP-INF\lib). Make sure in EAR project that these jars appear in (Java Build Path 🡪 Libraries 🡪 classpath 🡪 EAR).
* Dynamic Web Project Dependencies:
  + Project Properties 🡪 Deployment Assembly 🡪 Add 🡪 Project 🡪 Select core.
  + Project Properties 🡪 Deployment Assembly 🡪 Add 🡪 Archives from Workspace 🡪Add 🡪 tahseelDev 🡪 EarContent 🡪 lib 🡪 Select your JARs.

# Version Control Systems

## Common

* It is not allowed at any circumstance to COMMIT any snippet of code contains errors or could cause errors at other parts of the code base.
* COMMIT only complete tasks (try to make each commit a logically separate changeset).
* COMMIT only with quality commit messages:
  + Start with a single line commit description summary that’s no more than about 150 characters.
  + If there are more details about the commit add first blank line after the summary line followed by a more detailed explanation message.
* Don’t COMMIT any class before “Auto-Formatting” it based on the agreed formatting rules.
* Eclipse plug-ins which affect the code and agreed to be used must be unified within the team.
* Eclipse plug-ins which do not affect any sources or the final war can be used freely without committing its settings.
* Workspace metadata / generated classes and any team member preference data should be ignored from source control.
* It is highly recommended to compare and UPDATE/PULL every day.
* A review strategy should be settled within each project.

## SVN

* Manipulate renaming and moving files through SVN tools.
* Shared project settings, class path and lib should be committed and locked.
* Always UPDATE before COMMIT unless in very special cases related to different files.
* Recommended review process: Always review with the team leader before commit.

## Git

* Use different repository for every workspace. Also recommended to use Git for code only.
* Shared project settings, class path and lib should be pushed and never changed unless agreed.
* Always align with the commit guidelines above. When we need history adjustment, we can make use of rewriting history capabilities (amend - edit comments - delete - reorder - split - squash).
* Do not REBASE commits that you have pushed already unless it is in your branch that you only use and before merging with the master branch by the team leader.
* During rebasing resolve the conflicts if happened and use mark as merged to continue rebase. If there are many local commits use interactive rebase to rebase every commit.
* It is recommended to PULL everyday using:
  + Compare with the remote master.
  + Stash any uncommitted work.
  + PULL if there is no local commits or FETCH if there are any local commits.
  + In case of FETCH, REBASE your work (only MERGE in very rare situations in case we need to preserve the written code that will be changed according to the rebase or merge. MERGE need team leader approval).
  + Apply stash changes and continue the work (Always remove the stash after usage).
* Tagging:
  + Always Tag commits that represent production version: “PRD-DD-MM-YYYY{-DESC}”.
  + Always Tag commits that represent quality control version: “QC-DD-MM-YYYY{-DESC}”
  + Always Tag commits that represent demo, training, UAT, SIT or any implementation version: “IMPL-DD-MM-YYYY{-DESC}”
  + In case of tags maintenance, remove all and pull from remote.
  + Tags is the responsibility of the configuration and deployment leader in the project.
* Branching.
  + Always use branches according to the working and review methodology.
  + Teams always collaborate on core branches (like master, production, …).
  + If noncore branch is needed to handle task, qc-fix, prd-fix, … then it should be named “username-task id or desc” or “team/group name- task id or desc”.
  + Noncore branches should be used only by the same developer/team/group and his team leader for review. When team uses noncore branches, they should use REBASE strategy.
  + Noncore branches should be removed after merge.
* Working and Review Methodology:
  + Only one core branch: “master” branch (for dev, qc, prd, …).
  + None core branches should be for:
    - Every task not planned for the next production deployment. (REBASE Strategy).
    - Fixes in case that master branch includes work that should not be shipped with the fix. (Merge Strategy to preserve the deployment tag).
  + Regular tasks that planned for next production deployment should be done and committed on the master, then pushed only during the review process (always make sure to compare and rebase first before PUSH).
  + Emergency tasks that planned for next production deployment:
    - If no local commits: stash your work, do the emergency task (as a regular task) then apply the stash changes.
    - If local commits exist: Make a noncore branch from the origin/master, do the emergency task (as a regular task but on the new branch) then rebase your local master commits.

Appendix A: Tools and Libraries

## A.1 Tools

|  |  |  |  |
| --- | --- | --- | --- |
| Type | Tool | Version | Notes |
| OS | RedHat Enterprise Linux (RHEL) | 7.5 | Development Environment: Windows |
| DBMS | IBM DB2 | 11.1.3.3 |  |
| DB Browser | Toad for DB2 |  |  |
| Application Server | IBM WebSphere Application Server (WAS) | 9.0.0.7 |  |
| JDK | IBM SDK | 8.0.5.11 |  |
| Reports IDE | iReport | 5.0.1 |  |
| Development IDE | Eclipse | Oxygen 3a |  |
| Repository | SVN and Tortoise |  |  |

WAS/DB2/JEE/JSF Tools during 2018/2019

WebLogic/Oracle DB/JEE/JSF Tools during 2014/2018

|  |  |  |  |
| --- | --- | --- | --- |
| Type | Tool | Version | Notes |
| OS | RedHat Enterprise Linux (RHEL) | 7 | Development Environment: Windows |
| DBMS | Oracle DB | 12c |  |
| DB Browser | SQL Developer | 4 |  |
| Application Server | WebLogic | 12.2.1.0.0 |  |
| JDK | Oracle JDK | 8 |  |
| Reports IDE | iReport | 5.0.1 |  |
| Development IDE | Eclipse | Neon SR1 |  |
| Repository | SVN and Tortoise |  |  |

## A.2 Libraries

### A.2.1 Primefaces-Hibernate During 2018/2019

|  |  |
| --- | --- |
| Library | Jars |
| JSF 2.2  Primefaces 6.1 | * myfaces-api-2.2.12.jar * myfaces-bundle-2.2.12.jar * myfaces-impl-2.2.12.jar * primefaces-6.1.jar |
| Hibernate 5 | * antlr-2.7.7.jar * classmate-1.3.0.jar * dom4j-1.6.1.jar * hibernate-commons-annotations-5.0.1.Final.jar * hibernate-core-5.2.10.Final.jar * hibernate-jpa-2.1-api-1.0.0.Final.jar * jandex-2.0.3.Final.jar * javassist-3.20.0-GA.jar * jboss-logging-3.3.0.Final.jar * jboss-transaction-api\_1.2\_spec-1.0.1.Final.jar * ojdbc7.jar (for Oracle) * mssql-jdbc-6.2.1.jre8.jar (for MS SQL Server) * db2jcc4.jar (for DB2) * db2jcc\_license\_cu.jar (for DB2) |
| Jasper Reports 5.0.1 | * jasperreports-5.0.1.jar * jdt-compiler-3.1.1.jar * itext-2.1.7.jar * fonts.jar * barcode4j-2.0.jar * xml-apis-ext.jar * batik-all-1.7.jar |
| Common | * commons-beanutils-1.9.2.jar * commons-codec-1.6.jar * commons-collections-3.2.2.jar * commons-digester-1.8.jar * commons-logging-1.1.1.jar * commons-discovery-0.4.jar * commons-el-1.0.jar * commons-fileupload-1.0.jar * commons-lang-2.1.jar * commons-validator-1.3.1.jar |
| Time Operations | * joda-time-1.6.2.jar |
| Scheduler | * c3p0-0.9.1.1.jar * log4j-1.2.16.jar * quartz-2.2.1.jar * quartz-jobs-2.2.1.jar * slf4j-api-1.6.6.jar * slf4j-log4j12-1.6.6.jar |
| Excel | * poi-3.17.jar * poi-ooxml-3.17.jar * ooxml-schemas-1.3.jar * commons-collections4-4.1.jar * xmlbeans-2.6.0.jar |

### A.2.2 Richfaces-Hibernate During 2014/2018

|  |  |
| --- | --- |
| Library | Jars |
| JSF 2.1.12  Richfaces 4.3.2 | * myfaces-api-2.1.12.jar * myfaces-bundle-2.1.12.jar * myfaces-impl-2.1.12.jar * richfaces-components-api-4.3.2.Final.jar * richfaces-components-ui-4.3.2.Final.jar * richfaces-core-api-4.3.2.Final.jar * richfaces-core-impl-4.3.2.Final.jar * annotations-4.0.0.Final.jar * cssparser-0.9.5.jar * guava-14.0-rc1.jar * sac-1.3.jar |
| Hibernate 4.1.10 | * antlr-2.7.7.jar * dom4j-1.6.1.jar * hibernate-commons-annotations-4.0.1.Final.jar * hibernate-core-4.1.10.Final.jar * hibernate-jpa-2.0-api-1.0.1.Final.jar * javassist-3.15.0-GA.jar * jboss-logging-3.1.0.GA.jar * jboss-transaction-api\_1.1\_spec-1.0.0.Final.jar * ojdbc7.jar |
| Jasper Reports 5.0.1 | * jasperreports-5.0.1.jar * jdt-compiler-3.1.1.jar * itext-2.1.7.jar |
| Common | * commons-beanutils-1.8.3.jar * commons-codec-1.6.jar * commons-collections-3.2.1.jar * commons-digester-1.8.jar * commons-logging-1.1.1.jar * commons-discovery-0.4.jar * commons-el-1.0.jar * commons-fileupload-1.0.jar * commons-lang-2.1.jar * commons-validator-1.3.1.jar |
| Handle passing parameters from XHTML to Java with old servlets version | * jboss-el-2.0.0.GA.jar |
| Time Operations | * joda-time-1.6.2.jar |
| Scheduler | * c3p0-0.9.1.1.jar * log4j-1.2.16.jar * quartz-2.2.1.jar * quartz-jobs-2.2.1.jar * slf4j-api-1.6.6.jar * slf4j-log4j12-1.6.6.jar |

Appendix B: Installations

## B.1 Env: JDK11 – WebLogic 14c – Eclipse 2020-06

|  |  |
| --- | --- |
| JDK 11.0.10 (64 bit) | <https://www.oracle.com/sa/java/technologies/javase-jdk11-downloads.html> |
| WLS 14.1.1.0.0 | <https://www.oracle.com/middleware/technologies/weblogic-server-installers-downloads.html> |
| Eclipse 2020-06 (64 bit) | <https://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/2020-06/R/eclipse-jee-2020-06-R-win32-x86_64.zip&mirror_id=518> |

### B.1.1 JDK 11.0.10 LTS

1. Uninstall any version of JDK 11 installed at your machine.
2. Install the downloaded JDK 11 update 10.
3. Set environment variable **JAVA\_HOME** in the “system variables” to point to the new JDK installation. For example, “**C:\Program Files\Java\jdk-11.0.10**”.
4. Update the path environment variable by removing any java path and add **“%JAVA\_HOME%\bin;**” at the beginning of the list.

### B.1.2 WLS 14.1.1.0.0

1. Run your command prompt (cmd) as **administrator**.
2. Change directory to the path that contains the downloaded Weblogic source jar file
3. Execute the command **java -jar fmw\_14.1.1.0.0\_wls\_lite\_generic.jar** and press enter then follow the normal process to install Weblogic.
4. Create a domain in development mode with default configurations (weblogic/weblogic1).
5. Adjusting memory parameters:
   1. Open the file “**commBaseEnv.cmd**” which should be located in your Weblogic installation folder which is by default “**C:\Oracle\Middleware\Oracle\_Home\oracle\_common\common\bin**” and change all the **MEM\_ARGS** and **UTIL\_MEM\_ARGS** to be as the following  
      *set MEM\_ARGS=-Xms1024m -Xmx1024m  
      set UTILS\_MEM\_ARGS=-Xms1024m -Xmx1024m*
   2. Open the file “**setDomainEnv.cmd**” which should be located in your domain folder which is by default “**C:\Oracle\Middleware\Oracle\_Home\user\_projects\domains\base\_domain\bin**” and change all the **WLS\_MEM\_ARGS\_64BIT** and **WLS\_MEM\_ARGS\_32BIT** to be as the following  
      *set WLS\_MEM\_ARGS\_64BIT=-Xms1024m -Xmx1024m  
      set WLS\_MEM\_ARGS\_32BIT=-Xms1024m -Xmx1024m*
6. Define your data sources.
7. Change JTA timeout from 30 seconds to **1200** seconds.  
   (from the console web app -> select the domain “base\_domain” -> select “JTA” tab -> update the field “Timeout Seconds”).

### B.1.3 Eclipse 2020-06

1. Extract eclipse at any directory you prefer.
2. Add eclipse shortcut to desktop.
3. Make the workspace folder is [/Skills/SkillsDevelopment/](https://ejadasvn.ejada.com/svn/MOFTAHSEEL/E%20-%20Development/Application/Code)BuildingSkillsWorkspace and ignore.metadata folder.
4. Install all the required plugins:
   1. **Oracle Weblogic Server Tools**

Help menu -> Install new software.

Add -> Copy the link “<http://download.oracle.com/otn_software/oepe/12.2.1.10/photon/repository/>” in the location section and type “Oracle” into name.

Select “Tools” and press “Next”.

1. Add new server from “Servers” tab:
   1. Expand “Oracle”, and select “Oracle WebLogic Server”.
   2. Select “Weblogic Home” to be “C:\Oracle\Middleware\Oracle\_Home\wlserver”.
   3. Select “base\_domain”.
2. Don’t import **unused imports**. You can fulfill this requirement by adjusting the following configuration and organize the imports of any class that you will create or edit by using the shortcut “CTRL + SHIFT + O”.
   1. Open Window-> Preferences and search for compiler (you will find it under Java).
   2. Select Errors/Warnings.
   3. Select Unnecessary Code region you will see Unused imports option.
   4. Change the select menu of unused import from Warning to Error.
   5. Click Apply and Close Then choose yes.
3. To **avoid the warning for serialization** in any class that you are sure you don’t need it to be serialized, adjust the following configurations
   1. Open Window-> Preferences and search for compiler (you will find it under Java).
   2. Select Errors/Warnings.
   3. Search for Serializable class without serialVersionUID.
   4. Change the select menu from Warning to Ignore.
   5. Click Apply and Close Then choose yes.
4. To guarantee the code formatting and organizing the imports, we should make these tasks as automatic tasks after modification at any file. We do that by **enabling the Save actions** according to the following the steps.
   1. Open Window-> Preferences and search for Save Actions.
   2. Select the save actions under Java->Editor.
   3. Check the option “Perform the selected actions on save”.
   4. Check the option “Format source code”.
   5. Select the “Format all lines” toggle.
   6. Check the option “Organize imports”.
   7. Click Apply and Close.
5. To enable eclipse auto format using “CTRL + SHIFT + F” and keep the code readable:
   1. Make a new profile “Custom Java Conventions” based on “Java Conventions [built-in]”:

Open Window->Preferences, type formatter in the search box.

Select the formatter under Java->Code Style.

Make the new profile.

Change “Line Wrapping / Maximum line width” to “500”.

Check “Line Wrapping / Never join already wrapped lines”.

Change “Comments / Maximum width for comments” to “500”.

Check “Comments / Never join lines”.

* 1. For xhtml tags formatting:

Open Window->Preferences, type html in the search box.

Select the Editor under Web->HTML Files.

Change line width to 500.

In the Inline Elements list delete all tags in it.

Apply and OK

## B.2 Env: IBM SDK – WAS – Eclipse Oxygen 3a – DB2

This installation guide is intended for the team members to understand the tools that we are going to use during the project and be able to setup them easily. All the tools will be provided to you and no need to download them.

### B.2.1 IBM Installation Manager

1. Download the installation manager from the [link](https://delivery04.dhe.ibm.com/sdfdl/v2/sar/CM/RA/07hnz/0/Xa.2/Xb.jusyLTSp44S0eY6gDODD8PqJAeHtzbXNIHWOBoWlzF1l2JQIMER2AX8LIKw/Xc.CM/RA/07hnz/0/agent.installer.win32.win32.x86_64_1.8.9000.20180313_1417.zip/Xd./Xf.LPR.D1VC/Xg.9642766/Xi.habanero/XY.habanero/XZ.d9rXWb3pC6xFExSiLucIdzIkHCo/agent.installer.win32.win32.x86_64_1.8.9000.20180313_1417.zip).
2. Unzip it.
3. Run the installer as administrator and follow the wizard.
4. If you have a problem with the language (i.e. UI is in Arabic instead of English) follow these steps
   1. Before installation:  
      Open \Installation Manager Source Folder\configuration\config.ini  
      After installation  
      Open \Installation Manager Installation Folder\eclipse\configuration\config.ini
   2. At the end of file add   
      osgi.nl=en
   3. Restart installing manager.

### B.2.2 IBM SDK and WAS

**CELL**

**DMGR**

Deployment Manager

**…………**

**Agent**

**Server 01**

**Server 02**

**Server n**

**…**

**Node 01**

**Agent**

**Server 01**

**Server 02**

**Server n**

**…**

**Node 02**

**Agent**

**Server 01**

**Server 02**

**Server n**

**…**

**Node n**

Installation at PROD environment

**Cell**

**Node**

**Server**

Installation at DEV environment

#### B.2.2.1 Install IBM SDK and WAS

1. Run “IBM Installation Manager” as administrator.
2. Add the following repositories from File -> Preferences

|  |  |  |
| --- | --- | --- |
| Tool | Location | Notes |
| WAS 9.0.0.0 | E:\IBM-Tools\WAS\WAS\_9.0\_Windows\WAS\_ND\_V9.0\_MP\_ML.zip |  |
| IBM SDK 8.0.3.0 | E:\IBM-Tools\WAS\WAS\_9.0\_Windows\sdk.repo.8030.java8.win.zip |  |
| JAVA 8 |  | Extract all the parted zip files at one directory called “was.repo.9000.java8” |
| WAS 9.0.0.7 FP |  |  |
| JAVA 8.0.5.11 UPDATE |  | Extract all the parted zip files at one directory called “ibm-java-sdk-8.0-5.11-all” |



1. Press “Install” and select “IBM WebSphere Application Server Network Deployment” and follow the wizard according to the below screenshots.  
     
     
     
     
     
     
     
   

#### B.2.2.2 Create a Profile

1. Run the “Profile Management Tool” as administrator.  
     
   
2. Select “Application Server”.  
     
   
3. Select “Advanced profile creation”.  
     
   
4. Select “Deploy the administrative console” and “Deploy the Installation Verification Tool application” and deselect “Deploy the default application”  
     
   
5. Let the profile name “AppSrv01” and location as default and select Server runtime performance tuning setting “Development”.  
     
   
6. Set the Node name “**Node01**” and Server name “**server1**” and let the Host name as is but be sure it is identical to your full computer name.  
     
   
7. Select “Enable administrative security” and set the user name “**admin**” and the password “**admin**”  
     
   
8. Select “Create a new default personal certificate” and “Create a new root signing certificate”.  
     
   
9. Set the “expiration period” to the maximum “**15**” years.   
     
   
10. Let the ports as default. We are interested only at the highlighted ports.   
      
    
11. Set the windows service startup type to “manual”  
      
    
12. From Windows -> Open Services and then Start the service named “IBM WebSphere Application Server V9.0 – Node01”.  
      
    
13. Open the URL <http://localhost:9060/admin> and let chrome trust the certificate, it will redirect you the secure URL <https://localhost:9043/ibm/console/logon.jsp> enter the credentials and press “Log in”.  
      
    
14. If the server is not running, you can use the command line to start it manually.
    1. Open the “cmd” as administrator.
    2. Change the directory to the path “C:\Program Files\IBM\WebSphere\AppServer\bin”
    3. Run the command “startServer.bat server1”.
    4. If an error occurred saying that an instance of server is currently running, stop the server first using the command “stopServer.bat server1”. Then start it as previous step by command “startServer.bat server1”.
15. You can find the logs at the following directory  
    “C:\Program Files\IBM\WebSphere\AppServer\profiles\AppSrv01\logs\server1”.
16. Adjust the heap size and enable the debug mode:
    1. Click on “WebSphere application server” and then click on “server1”.  
         
       
    2. Expand “Java and process management” and click on “process definition”.  
         
       
    3. Click on “Java Virtual Machine”.  
         
       
    4. Set the “Initial heap size” and the “Maximum heap size” to be **1024** and check the “Debug Mode” and press “OK” then “Save”.  
         
       

#### B.2.2.3 Data Source

1. Create a JDBC Provider
   1. Expand “Resources”, expand “JDBC”, click on “JDBC providers”, select the scope to be the cell scope and press “New”.  
        
      
   2. Select Database type “**DB2**”, Provider type “**DB2 Using IBM JCC Driver**”, Implementation type “**Connection pool data source**” and set an appropriate name like “**TAHSEEL DB2 Using IBM JCC Driver**” and press Next.  
        
      
   3. Copy the folder “**DB2DRV**” to the location “**C:\Program Files\IBM**”.
   4. Set the three environment variables “${DB2\_JCC\_DRIVER\_PATH}”, “${PUREQUERY\_PATH}” and “${DB2\_JCC\_DRIVER\_NATIVEPATH}” to be “**C:/Program Files/IBM/DB2DRV**” then press Next and Finish.  
        
      
2. Create Data Source:
   1. Expand “Resources”, expand “JDBC”, click on “Data Sources”, select the scope to be the cell scope and press “New”.  
        
      
   2. Enter the data source name and the JNDI name. for example, “TAHSEEL\_POC” and “jdbc/TAHSEEL\_POC”.  
        
      
   3. Enter the Database name and the Server name and press Next.  
        
      
   4. Press Next  
        
      
   5. Press Finish and Save.
   6. Click on your data source.  
        
      
   7. Click on the link “[Global J2C authentication alias](https://localhost:9043/ibm/console/com.ibm.ws.console.security.forwardCmd.do?csrfid=-2049596727&forwardName=JAASAuthData.content.main&contextId=cells:lb055Node01Cell) [Security domains](https://localhost:9043/ibm/console/com.ibm.ws.console.security.forwardCmd.do?csrfid=-2049596727&forwardName=SecDomain.content.main&resourceUri=security.xml&perspective=tab.configuration&contextId=cells:lb055Node01Cell)” to define the credentials.   
        
      
   8. Press New.  
        
      
   9. Enter the credentials for the database and an alias for the user, press OK and then Save.  
        
      
   10. Click on your data source.
   11. Select the created user at the security settings section as shown below and press OK then press Save.  
         
       
   12. Test the connection. If the test failed you may need to restart the server using the following commands; stopServer.bat server1 and startServer.bat server1 (Run the cmd as administrator and change the directory to the path “C:\Program Files\IBM\WebSphere\AppServer\bin”).

#### B.2.2.4 Configuring Encoding

1. On the Application Server page, click on the name of the server you want enabled for UTF-8.
2. Expand “Java and process management” and click on “process definition”.
3. On the Process Definition page, click Java Virtual Machine.
4. On the Java Virtual Machine page, specify -Dfile.encoding=UTF-8 for Generic JVM Arguments and click OK.
5. Click Save on the console taskbar.
6. Restart the application server.

### B.2.3 Eclipse and plugins

1. Download Eclipse Oxygen 3a from the [URL](https://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/oxygen/3a/eclipse-jee-oxygen-3a-win32-x86_64.zip&mirror_id=1093).
2. Extract eclipse at any directory you prefer.
3. Add eclipse shortcut to desktop.
4. Run as administrator, to make this automatic apply these steps:
   1. right-click on the eclipse shortcut and click properties
   2. From Compatibility tab, in privilege section check 'run this program as an administrator'
5. Make the workspace folder is <https://ejadasvn.ejada.com/svn/MOFTAHSEEL/E%20-%20Development/Application/Code> and do NOT commit .metadata folder.
6. Install all the required plugins:
   1. **Jinto** for resource editing <http://www.guh-software.de/eclipse>
      1. Help menu -> Install new software
      2. Add -> Copy the previous Jinto link in the location section and type any name
      3. Select “Open Source Tools” only.
      4. Prevent Jinto from adding date to messages.  
         Window -> preferences -> uncheck include jinto header in resource files on saving.
   2. **Subclipse** for SVN management (Use Eclipse Marketplace)
      1. Help menu -> Eclipse Marketplace
      2. Search for “Subclipse” and install it.
      3. Select all items.
   3. **JBoss** Tools (Use Eclipse Marketplace)
      1. Help menu -> Eclipse Marketplace
      2. Search for “JBoss Tools”.
      3. Select “JBoss Tools 4.5.3.Final” and install it.
      4. Select the following items:
         1. JBoss Tools Usage Reporting.
         2. JBoss Tools Foundation.
         3. JBoss Tools Java Standard Tools.
         4. JBoss Tools JSF.
         5. JBoss Tools RichFaces.
         6. JBoss Tools Visual Page Editor.
   4. **IBM WebSphere Application Server** V9.x Developer Tools For Oxygen (Use Eclipse Marketplace)
      1. Help menu -> Eclipse Marketplace
      2. Search for “IBM WebSphere Application Server V9.x Developer Tools for Oxygen”.
      3. Select “IBM WebSphere Application Server V9.x Developer Tools for Oxygen” and install it.
      4. Don’t change the default selected items.
      5. In Servers panel -> you can click on pause automatic updates.
7. Open eclipse.ini from eclipse directory and add these 2 lines just before *–vmargs*  
   *-vm  
   C:\Program Files\IBM\WebSphere\AppServer\java\8.0\jre\bin\javaw.exe*
8. Add WebSphere Application Server
   1. From Window -> Perspective -> open perspective -> Java EE
   2. From Servers Panel -> Click add new Server
   3. Choose IBM folder -> WebSphere Application Server traditional V9.0
   4. Enter Installation Directory of WebSphere which is *C:\Program Files\IBM\WebSphere\AppServer* then next.
   5. Type username and password and click Finish.
   6. Right-click the server -> open and change Timeout to be 6000 milliseconds.
9. Don’t import **unused imports**. You can fulfill this requirement by adjusting the following configuration and organize the imports of any class that you will create or edit by using the shortcut “CTRL + SHIFT + O”.
   1. Open Window-> Preferences and search for compiler (you will find it under Java).
   2. Select Errors/Warnings.
   3. Select Unnecessary Code region you will see Unused imports option.
   4. Change the select menu of unused import from Warning to Error.
   5. Click Apply and Close Then choose yes.
10. To **avoid the warning for serialization** in any class that you are sure you don’t need it to be serialized, adjust the following configurations
    1. Open Window-> Preferences and search for compiler (you will find it under Java).
    2. Select Errors/Warnings.
    3. Search for Serializable class without serialVersionUID.
    4. Change the select menu from Warning to Ignore.
    5. Click Apply and Close Then choose yes.
11. To guarantee the code formatting and organizing the imports, we should make these tasks as automatic tasks after modification at any file. We do that by **enabling the Save actions** according to the following the steps.
    1. Open Window-> Preferences and search for Save Actions.
    2. Select the save actions under Java->Editor.
    3. Check the option “Perform the selected actions on save”.
    4. Check the option “Format source code”.
    5. Select the “Format all lines” toggle.
    6. Check the option “Organize imports”.
    7. Click Apply and Close.
12. To enable eclipse auto format using “CTRL + SHIFT + F” and keep the code readable (suitable line length in java and separated tags in xhtml), change the default settings in eclipse as follows:
    1. For java line length:
       * + 1. Open Window->Preferences, type formatter in the search box.
           2. Select the formatter under Java->Code Style.
           3. Click import and select the [file](https://ejadasvn.ejada.com/svn/MOFTAHSEEL/D%20-%20Technical%20Design/Application/Lessons/HCM_Java_Conventions_Oxygen.xml)
    2. For xhtml tags formatting:
       * + 1. Open Window->Preferences, type html in the search box.
           2. Select the Editor under Web->HTML Files.
           3. Change line width to 500.
           4. In the Inline Elements list delete all tags in it.
           5. Apply and OK

### B.2.4 IBM Database Client for DB2

1. Download “ibm\_data\_server\_driver\_package\_win64\_v11.1.exe” from the [URL](https://www-01.ibm.com/marketing/iwm/iwm/web/preLogin.do?source=swg-idsdpds).
2. Install as administrator “ibm\_data\_server\_driver\_package\_win64\_v11.1.exe” and let everything as default at the installation wizard.
3. Verify the installation of the data server driver by executing the command “*db2cli validate*” at the path “*C:\Program Files\IBM\IBM DATA SERVER DRIVER\bin*”. You should get a result similar to the following screenshot.  
     
   

### B.2.5 Toad for DB2

#### B.2.5.1 Installation

1. Download “ToadforDB2\_6.5.0.125.x64.msi” freeware version from the [URL](https://www.toadworld.com/download/toad-for-ibm-db2/freeware).
2. Install “ToadforDB2\_6.5.0.125.x64.msi” and let everything as default at the installation wizard. You will need to open the command prompt at administrator and run the file inside it.

#### B.2.5.2 Configuration

1. Disable the auto commit: Options -> Database -> General  
     
   
2. Change the editor encoding to UTF-8: Options -> Editor -> General  
     
   
3. Change the date format to be shown at the table viewer (grid) to be “*dd/MM/yyyy HH:mm:ss*”: Options -> Environment -> Grid  
     
   

### B.2.6 IBM DS

1. Run “IBM Installation Manager” as administrator.
2. Add the following repositories from File -> Preferences

|  |  |  |
| --- | --- | --- |
| Tool | Location | Notes |
| ibm\_ds4130\_win | E:\IBM-Tools\DB2\\ibm\_ds4130\_win\disk1\diskTag.inf |  |

1. Press “Install” and select “IBM Data Studio client” and follow the wizard without changing anything.

### B.2.7 iReport

1. Creating new Connection:
   1. Add db2 JDBC drivers from iReport -> tools -> options -> iReport -> classPath
   2. add jars (db2jcc4.jar, db2jcc\_license\_cu.jar) that are in the path 'C:\Program Files\IBM\DB2DRV'
   3. do NOT check reloadable.
   4. Add new connection and enter these information:
      1. JDBC Driver: com.ibm.db2.jcc.DB2Driver
      2. URL: jdbc:db2://192.168.8.116:50000/TAHSEEL
      3. username: db2inst1
   5. Make sure to test connection

## B.3 Env: JDK8 – WebLogic 12.2.1.0.0 – Eclipse Neon

1. Prerequisites:

|  |  |
| --- | --- |
| JDK 8 update 51 (64 bit) | <http://download.oracle.com/otn/java/jdk/8u51-b16/jdk-8u51-windows-x64.exe> |
| WLS 12.2.1.0.0 | <http://download.oracle.com/otn/nt/middleware/12c/1221/fmw_12.2.1.0.0_wls_Disk1_1of1.zip> |
| Eclipse Neon (64 bit) SR1 | <https://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/neon/1a/eclipse-jee-neon-1a-win32-x86_64.zip&mirror_id=1068> |

1. JDK Installation:
   1. Uninstall any version of JDK 8 installed at your machine.
   2. Install the downloaded JDK 8 update 51.
   3. Set environment variable **JAVA\_HOME** to point to the new JDK installation. For example, “C:\Program Files\Java\jdk1.8.0\_51”.
   4. Update the **path** environment variable by removing “C:\ProgramData\Oracle\Java\javapath;” and add “**%JAVA\_HOME%\bin;**” if it doesn’t exist.
2. WLS 12.2.1.0.0 Installation:
   1. Run your command prompt (cmd) as **administrator**.
   2. Change directory to the path that contains the downloaded Weblogic source jar file
   3. Execute the command **java –jar fmw\_12.2.1.0.0\_wls.jar** and press **enter** then follow the normal process to install Weblogic and create a domain in **development** mode.
   4. Adjusting memory parameters:
      1. Open the file “**commBaseEnv.cmd**” which should be located in your Weblogic installation folder which is by default “C:\Oracle\Middleware\Oracle\_Home\oracle\_common\common\bin” and change all the MEM\_ARGS and UTIL\_MEM\_ARGS to be as the following  
         set MEM\_ARGS=-Xms1024m -Xmx1024m -XX:PermSize=512m -XX:MaxPermSize=512m  
         set UTILS\_MEM\_ARGS=-Xms1024m -Xmx1024m -XX:PermSize=512m -XX:MaxPermSize=512m
      2. Open the file “**setDomainEnv.cmd**” which should be located in your domain folder which is by default “C:\Oracle\Middleware\Oracle\_Home\user\_projects\domains\base\_domain\bin” and change all the WLS\_MEM\_ARGS\_64BIT and WLS\_MEM\_ARGS\_32BIT to be as the following  
         set WLS\_MEM\_ARGS\_64BIT=-Xms1024m -Xmx1024m -XX:PermSize=512m -XX:MaxPermSize=512m  
         set WLS\_MEM\_ARGS\_32BIT=-Xms1024m -Xmx1024m -XX:PermSize=512m -XX:MaxPermSize=512m
   5. Define the “BGETR” and “BGETR\_TEST” data sources.
   6. Change JTA timeout from 30 seconds to 1200 seconds.  
      (from the console web app -> select the domain “base\_domain” -> select “JTA” tab -> update the field “Timeout Seconds”)
3. Eclipse Configurations:
   1. Extract eclipse at any directory you prefer.
   2. Open eclipse at the same workspace to preserve all the configurations that has been used at Kepler.
   3. Install updates for eclipse if exist
      1. Select “Check for updates” from the “Help” menu and install all the updates.
   4. Install all the required plugins:
      1. Jinto for resource editing <http://www.guh-software.de/eclipse>
         1. Select “Open Source Tools” only.
      2. Subclipse for SVN management (Use Eclipse Marketplace)
         1. Search for “Subclipse” and install it.
         2. Select all items.
      3. JBoss Tools (Use Eclipse Marketplace)
         1. Search for “JBoss Tools”.
         2. Select “JBoss Tools 4.4.1.Final” and install it.
         3. Select the following items:
            1. JBoss Tools Usage Reporting.
            2. JBoss Tools Foundation.
            3. JBoss Tools Java Standard Tools.
            4. JBoss Tools JSF.
            5. JBoss Tools LiveReload.
            6. JBoss Tools RichFaces.
            7. JBoss Tools Visual Page Editor.
      4. Oracle Weblogic Server Tools (Use Eclipse Marketplace)
         1. Search for “Oracle Weblogic Server Tools”.
         2. Select “Oracle Weblogic Server Tools” and install it.
         3. Don’t change the default selected items.
   5. Update the **formatter** file [https://ejadasvn.ejada.com/svn/BGETR/BG ETR Docs/Lessons Learned/Helper Files/HCM Java Conventions \_neon.xml](https://ejadasvn.ejada.com/svn/BGETR/BG%20ETR%20Docs/Lessons%20Learned/Helper%20Files/HCM%20Java%20Conventions%20_neon.xml) and **reimport** it again to eclipse.
   6. Delete the old Weblogic server instance (10.3.6).
   7. Add a new Weblogic server instance (12.2.1.0.0).
4. Project:
   1. Update your project to get all the new configuration.
   2. Right click on the project and select “Migrate Project Runtime”.
   3. Change the compiler level of the project from “1.6” to “1.8”.
   4. Change the java facet of the project from “1.6” to “1.8”.
   5. You may need to clean and rebuild the project again if you find any build errors.
   6. Sometimes the debug on server doesn’t lunch automatically so when the publishing finish, you can use the application normally.
5. IReport Configurations:
   1. IReport 5.0.1 is not compatible with JDK 8. Open the file “**ireport.conf**” at the path “**C:\Program Files (x86)\Jaspersoft\iReport-5.0.1\etc**” as **administrator** and uncomment this line “#jdkhome="/path/to/jdk"” and let it point to your JDK 6 installation for example “**jdkhome="C:\Program Files\Java\jdk1.6.0\_45"**”.
6. Reports:
   1. Some of the expressions that has been used at our reports are not working now due to some compatibility issues. So we will need to test **ALL** the reports.
   2. Here is a listing for some issues and its suggested workaround
      1. If you are using String.contains(“”) -> replace it with String.indexOf(“”) > -1
      2. If you have a field contains an inline if expression which return different datatypes, you should cast anyone of them to make both return the same data type.  
         Example:  
         If you have an expression like:  
         ($P{P\_CATEGORY\_ID} == 1) ? $F{MILITARY\_NUMBER} : $F{JOB\_CODE}  
         you should change it to  
         ($P{P\_CATEGORY\_ID} == 1) ? $F{MILITARY\_NUMBER}.**toString()** : $F{JOB\_CODE}  
         as “$F{JOB\_CODE}” return a String object and “$F{MILITARY\_NUMBER}” return a BigDecimal object.
      3. Don’t use String.replace(“”,””) at the report expression, instead try to do all the replacements at the query.
      4. Don’t define any string at single quotes ‘’, instead use double quotes.  
         Example:  
         if you have expression like:  
         ($F{PH\_UNIT\_FULL\_NAME} != null ? $F{PH\_UNIT\_FULL\_NAME} : **'-'**)  
         you should change it to  
         ($F{PH\_UNIT\_FULL\_NAME} != null ? $F{PH\_UNIT\_FULL\_NAME} : **"-"**)

Appendix C: Working on Multiple database

1. **Database Layer**

* **Functions:** Implementing the following **U**ser **D**efined **F**unctions (**UDF**) at all databases to be used instead of the oracle built-in functions:
  + PKG\_CONCAT
  + PKG\_LENGTH
  + PKG\_NVL
  + PKG\_REPLACE
  + PKG\_SUBSTRING
  + PKG\_DATE\_TO\_CHAR
  + PKG\_CHAR\_TO\_DATE
  + PKG\_NUMBER\_TO\_CHAR
  + PKG\_CHAR\_TO\_NUMBER
  + PKG\_MOD
  + PKG\_RPAD
  + PKG\_LPAD
  + PKG\_ADD\_MONTHS
  + PKG\_MONTHS\_BETWEEN
  + PKG\_RTRIM

The scripts for these functions for each database exist at Helper Files\Packaging

At PKG\_DATE\_TO\_CHAR function, add the keyword “(r0)” before the date format pattern to remove the leading zeros from the date so the pattern will be like “(r0)MI/MM/YYYY” or “(r0)YYYY-MM-MI” instead of using the fill mode “fm”.

1. **Application Servers**
   * **WebLogic:**
   * **SQL Server – Data Source:**
     + Add **mssql-jdbc-6.2.1.jre8.jar** at: C:\Oracle\Middleware\Oracle\_Home\wlserver\server\lib   
       (You can find this jar at the WEB-INF\lib at the project workspace)
     + Modify the file **startWebLogic.cmd** at C:\Oracle\Middleware\Oracle\_Home\user\_projects\domains\*domain\_name*\bin by adding these two lines after the line containing “set CLASSPATH=%SAVE\_CLASSPATH%”:

|  |
| --- |
| set MSSQL\_PATH=%WL\_HOME%\server\lib\mssql-jdbc-6.2.1.jre8.jar  set CLASSPATH=%MSSQL\_PATH%;%CLASSPATH% |

* + - When creating the data source, select the Microsoft driver not the oracle driver. The driver name is: Microsoft's MS SQL Server Driver (Type 4) versions: 2005 and later
  + **PostgreSQL – Data Source:** 
    - Add **postgresql-42.1.4.jar** at: C:\Oracle\Middleware\Oracle\_Home\wlserver\server\lib  
      (You can find this jar at the WEB-INF\lib at the project workspace)
    - Modify the file **startWebLogic.cmd** at C:\Oracle\Middleware\Oracle\_Home\user\_projects\domains\*domain\_name*\bin by adding these two lines after the line containing “set CLASSPATH=%SAVE\_CLASSPATH%”:

|  |
| --- |
| set PG\_PATH=%WL\_HOME%\server\lib\postgresql-42.1.4.jar  set CLASSPATH=%PG\_PATH%;%CLASSPATH% |

1. **Application JARs:**
   * + Upgrade to **hibernate 5.2.10** by including the jars:

\*Note: The upgrade step was required because the current used hibernate version (4.1.10) supports till SQL Server 2005 dialect but we needed a support for SQL Server 2012 so we can use the Sequence to generate the primary key and this feature is supported starting from SQL Server 2008 dialect.

|  |
| --- |
| - antlr-2.7.7.jar  - classmate-1.3.0.jar  - dom4j-1.6.1.jar  - hibernate-commons-annotations-5.0.1.Final.jar  - hibernate-core-5.2.10.Final.jar  - hibernate-jpa-2.1-api-1.0.0.Final.jar  - jandex-2.0.3.Final.jar  - javassist-3.20.0-GA.jar  - jboss-logging-3.3.0.Final.jar  - jboss-transaction-api\_1.2\_spec-1.0.1.Final.jar  - ojdbc7.jar  - mssql-jdbc-6.2.1.jre8.jar |

* + - Modify the **weblogic.xml** to prefer the upgraded version of JBoss because weblogic is using an old version of it:

|  |
| --- |
| <package-name>org.jboss.logging.\*</package-name> |

1. **Data Access Layer:**

* **hibernate.cfg.xml:**
* **Oracle:**

|  |
| --- |
| <property name=*"connection.datasource"*>jdbc/*ETR\_HCM\_PKG*</property>  <property name=*"hibernate.transaction.coordinator\_class"*>org.hibernate.resource.transaction.backend.jta.internal.JtaTransactionCoordinatorBuilderImpl</property>  <property name=*"hibernate.transaction.jta.platform"*>org.hibernate.engine.transaction.jta.platform.internal.WeblogicJtaPlatform</property>  <property name=*"dialect"*>com.code.dal.dialects.HcmOracle12cDialect</property> |

* **SQL Server:**

|  |
| --- |
| <property name=*"connection.datasource"*>jdbc/*ETR\_HCM\_PKG\_SQL\_MS*</property>  <property name=*"hibernate.transaction.coordinator\_class"*>org.hibernate.resource.transaction.backend.jta.internal.JtaTransactionCoordinatorBuilderImpl</property>  <property name=*"hibernate.transaction.jta.platform"*>org.hibernate.engine.transaction.jta.platform.internal.WeblogicJtaPlatform</property>  <property name=*"dialect"*>com.code.dal.dialects.HcmSQLServer2012Dialect</property>  <property name=*"hibernate.default\_schema"*>dbo</property> |

* **PostgreSQL:**

|  |
| --- |
| <property name=*"connection.datasource"*>jdbc/*ETR\_HCM\_PKG\_PG*</property>  <property name=*"hibernate.transaction.coordinator\_class"*>org.hibernate.resource.transaction.backend.jta.internal.JtaTransactionCoordinatorBuilderImpl</property>  <property name=*"hibernate.transaction.jta.platform"*>org.hibernate.engine.transaction.jta.platform.internal.WeblogicJtaPlatform</property>  <property name=*"dialect"*>com.code.dal.dialects.HcmPostgreSQL95Dialect</property>  <property name=*"hibernate.default\_schema"*>ETR\_HCM\_PKG</property> |

* **DataAccess.java:**
* At **init**(): Build the session factory using:

|  |
| --- |
| StandardServiceRegistry standardRegistry = **new** StandardServiceRegistryBuilder().configure("com/code/dal/hibernate.cfg.xml").build();  Metadata metadata = **new** MetadataSources(standardRegistry).getMetadataBuilder().build();  *sessionFactory* = metadata.getSessionFactoryBuilder().build(); |

* At **executeQuery** method:
  + - 1. Use parameterized type query. The type should be the returned result set type Query<T>.
      2. Use createNativeQuery instead of createSQLQuery.
      3. Use q.setParameter and specify the parameter type manually. For example:

|  |
| --- |
| **if** (value **instanceof** Integer)  q.setParameter(paramName, value, org.hibernate.type.IntegerType.***INSTANCE***);  **else** **if** (value **instanceof** String)  q.setParameter(paramName, value, org.hibernate.type.StringType.***INSTANCE***);  **else** **if** (value **instanceof** Long)  q.setParameter(paramName, value, org.hibernate.type.LongType.***INSTANCE***); |

* At **getNextValFromSequence** method:

1. Generate the sequence next value using a database stored procedure implemented at all databases which takes the sequence name as a parameter and returns the next value as an output. The schema name is required for the stored procedure for both SQL Server and PostgreSQL, and it’s required for the sequence name for PostgreSQL.
2. The hibernate source code to access the stored procedure is:

|  |
| --- |
| Session session = *sessionFactory*.openSession();  **try** {  ResourceBundle config = ResourceBundle.*getBundle*("com.code.resources.config");  StoredProcedureQuery query = session.createStoredProcedureQuery(config.getString("defaultSchema") + ".**PKG\_GET\_SEQUENCE\_NEXT\_VALUE**")  .registerStoredProcedureParameter(1, String.**class**, ParameterMode.***IN***)  .registerStoredProcedureParameter(2, Long.**class**, ParameterMode.***OUT***)  .setParameter(1, config.getString("defaultSchema") +"."+ sequenceName);    query.execute();  **return** (Long) query.getOutputParameterValue(2);  } **catch** (Exception e) {  **throw** **new** DatabaseException(e.getMessage());  } **finally** {  session.close();  } |

* **config.properties:**
  + - Add the **defaultSchema** configuration: defaultSchema=*schema\_name* which is the same value configured at **hibernate.cfg.xml** for "hibernate.default\_schema" property.
* **Dialects:**
  + - Use custom dialects to be able to parse the JPQL with the modified functions’ calls.
      * Oracle:

|  |
| --- |
| **class** HcmOracle12cDialect **extends** Oracle12cDialect |

* + - * SQL Server:

|  |
| --- |
| **class** HcmSQLServer2012Dialect **extends** SQLServer2012Dialect |

* + - * PostgreSQL:

|  |
| --- |
| **class** HcmPostgreSQL95Dialect **extends** PostgreSQL95Dialect |

* + - Each dialect contains the registrations of each **UDF** as the following sample:

|  |
| --- |
| registerFunction("PKG\_NVL", **new** StandardSQLFunction(*config*.getString("defaultSchema") + ".PKG\_NVL"));  registerFunction("PKG\_LENGTH", **new** StandardSQLFunction(*config*.getString("defaultSchema") + ".PKG\_LENGTH"));  registerFunction("PKG\_CONCAT", **new** StandardSQLFunction(*config*.getString("defaultSchema") + ".PKG\_CONCAT"));  registerFunction("PKG\_SUBSTRING", **new** StandardSQLFunction(*config*.getString("defaultSchema") + ".PKG\_SUBSTRING"));  registerFunction("PKG\_REPLACE", **new** StandardSQLFunction(*config*.getString("defaultSchema") + ".PKG\_REPLACE")); |

* **ORM:**
  + - Modify all entities to make the JPQL uses the database **U**ser **D**efined **F**unctions (**UDF**) instead of Oracle built-in functions. We only introduce (**UDF**) for any operation other than the aggregate functions and comparison operators.
    - For the primary key generation at all entities:
* At **@SequenceGenerator**: add **allocationSize = 1** because the default value is 50 at the SQL Server and PostgreSQL databases.
* At **@GeneratedValue**: add **strategy = GenerationType.SEQUENCE** because the default strategy at SQL Server is GenerationType.IDENTITY which use the identity column strategy to generate the next value.
  + - The fields to be compared with any database column or the columns to be compared with each other should be with the same data type. i.e. both values should be numbers or strings because PostgreSQL database cannot handle the datatypes differences like what Oracle and SQL server handle. This will enforce us to change the parameter type sent to the JPQL or change the column type. We need to revisit all the queries and the methods sent parameters. For example:

|  |
| --- |
| qParams.put("P\_DECISION\_NUMBER", (decisionNumber == **null** || decisionNumber.length() == 0) ? FlagsEnum.ALL.getCode(): decisionNumber); // At the service, this will give an exception because getCode() returns integer value, so an empty string should be concatenated with it.  qParams.put("P\_DECISION\_NUMBER", (decisionNumber == **null** || decisionNumber.length() == 0) ? FlagsEnum.ALL.getCode() + "": decisionNumber); // correct |
| and (:P\_DEC\_NO = -1 OR v.decisionNumber = :P\_DEC\_NO) // At JPQL, this will give an exception because the column type is varchar but the sent parameter is an integer, so the sent parameter should be changed at the service to be String instead of int |

* + - When passing an array to a query, take care to use the toArray(new T[original List.size()]) instead of toArray().  
      For example:   
      qParams.put("P\_TASKS\_IDS", tasksIds.toArray(**new** Long[tasksIds.size()]));  
      Instead of  
      qParams.put("P\_TASKS\_IDS", tasksIds.toArray());
* **Manipulating Views:**
* At **AttachmentsService.java**, it’s required to insert the security key to the view instead of a table but PostgreSQL needs a trigger to do that.

|  |
| --- |
| CREATE OR REPLACE FUNCTION etr\_hcm\_pkg.cust\_sec\_web\_show\_vw\_dml()  RETURNS TRIGGER AS $function$  BEGIN  IF TG\_OP = 'INSERT' THEN  INSERT INTO etr\_hcm\_pkg.CUST\_SEC\_WEB\_SHOW VALUES(NEW.SEC\_KEY);  RETURN NEW;  END IF;  RETURN NEW;  END;  $function$ LANGUAGE plpgsql; |
| CREATE TRIGGER cust\_sec\_web\_show\_vw\_dml\_trig INSTEAD OF INSERT ON  etr\_hcm\_pkg.CUST\_VW\_SEC\_WEB\_SHOW FOR EACH ROW EXECUTE PROCEDURE etr\_hcm\_pkg. cust\_sec\_web\_show\_vw\_dml(); |

* **Reports:**
  + - At **ReportService.java**, send the **P\_SCHEMA\_NAME** parameter at **getReportData** method to be sent for all jasper files.

|  |
| --- |
| parameters.put("P\_SCHEMA\_NAME", BaseService.getConfig("defaultSchema")); |

* Add the file **sqljdbc4-2.0.jar** to the iReport classpath for SQL Server database connectivity. The file is located at: <https://ejadasvn.ejada.com/svn/BGETR/BG%20ETR%20Docs/Lessons%20Learned/Helper%20Files/Packaging/sqljdbc4-2.0.jar>.
* Change the query to use the database **U**ser **D**efined **F**unctions (**UDF**) for any operation other than the aggregate functions and comparison operators.
* Add a String parameter “**P\_SCHEMA\_NAME**” to all reports and subreports (This parameter is sent from the ReportService.java to all jasper files like the **P\_REPORTS\_ROOT** parameter).
* Uncheck the automatic field retrieval option in Report query window while changing query in case the Database type of the connection is PostgreSQL or Always develop the report query using Oracle connection as reports fields names are generated in lowercase while automatic reading from PostgreSQL.
* Use the schema name before any table, view, or function name. But access it using **$P!{P\_SCHEMA\_NAME}** so the query will use the parameter value without interpreting it.
* Send the Schema Name parameter to the subreports like the DECISION\_SIGNATURE subreport.
* Set a default value to the schema name parameter so the compiler will be able to compile the query and read the fields.
* The tables and views’ aliases and columns’ names are case sensitive. So, make sure that all reports are using capital letters.
* Convert “DECODE” to “CASE WHEN” statement.
* At **PKG\_DATE\_TO\_CHAR** function, add the keyword “**(r0)**” before the date format pattern to remove the leading zeros from the date so the pattern will be like “(r0)MI/MM/YYYY” or “(r0)YYYY-MM-MI” instead of using the fill mode “fm”.
* The tables’ columns to be compared should be with the same data type (as mentioned at previous section).
* Convert the (+) join operator to LEFT JOIN.

Appendix D: Conversion from Richfaces to Primefaces

1. **Components Conversion:**

|  |  |  |  |
| --- | --- | --- | --- |
| **RichFaces Tag** | **PrimeFaces Tag** | **Sample Screen** | **Notes** |
| a4j:ajax | p:ajax | FileUpload.xhtml |  |
| a4j:status | p:ajaxStatus | Template.xhtml | Should be exist at **Template** only |
| a4j:commandButton | p:commandButton |  | Shouldn’t be used at the application |
| rich:calendar | p:calendar |  | Should be exist at **AuditInquiry** only |
| rich:fileUpload | p:fileUpload | ImageUpload.xhtml |  |
| rich:dataTable | p:dataTable | VacationsCollectiveApproval.xhtml |  |
| rich:dataScroller | paginator="true" | VacationsCollectiveApproval.xhtml |  |
| rich:column | p:column | VacationsCollectiveApproval.xhtml |  |
| rich:columnGroup | p:columnGroup | DecisionsPrivileges.xhtml | use <p:row> for each line |
| rich:placeholder | p:watermark | Login.xhtml |  |
| rich:popupPanel | p:dialog | Template.xhtml |  |
| rich:component('name') | PF('name') | Template.xhtml | The ‘name’ used at primefaces is the "widgetVar" not the "Id" |
| rich:notifyStack | p:growl | Template.xhtml |  |
| rich:notifyMessages | p:growl | Template.xhtml |  |
| rich:tree | p:tree | UnitCreation.xhtml |  |
| rich:treeNode | p:treeNode | UnitCreation.xhtml |  |
| rich:collapsiblePanel | p:fieldset | OrganizationHierarchy.xhtml | RegularVacation.xhtml has issue of disabling the toggle |

1. **datatable Conversion:**

|  |  |
| --- | --- |
| **RichFaces datatable Attributes** | **PrimeFaces datatable Attributes** |
| rowKeyVar | rowIndexVar |
| onrowclick="changeSelectedRowStyle(this);" | selectionMode="single" rowKey="#{index}" |
| rich:dataScroller component | paginator="true" paginatorPosition="bottom" |
| If there is “execute” on the dataScroller | Add <p:ajax event="page" skipChildren="false" /> inside the datatable |
| onrowmouseover="changeRowOverStyle(this);"  onrowmouseout="resetRowOverStyle(this);"  rowClasses="gridRowOdd,gridRowEven" styleClass="gridMaster" | Replaced with CSS classes overriding the default primefaces styles |

1. **collapsiblePanel Conversion:**

|  |  |
| --- | --- |
| **RichFaces collapsiblePanel Attributes** | **PrimeFaces fieldset Attributes** |
| Header | legend |
| Expanded | collapsed |
|  | Add toggleable="true" because the default is false |

1. **fileUpload Conversion:**

|  |  |
| --- | --- |
| **RichFaces fileUpload Attributes** | **PrimeFaces fileUpload Attributes** |
| immediateUpload | auto |
| context parameter “org.richfaces.fileUpload.maxRequestSize” | sizeLimit |
| sizeExceededLabel | invalidSizeMessage |
| maxFilesQuantity | fileLimit |
| acceptedTypes (comma separated string) | allowTypes (regular expression) |
| ontyperejected | invalidFileMessage |
| doneLabel | handled manually (refer to ImageUpload.xhtml) |

1. **tree Conversion:**

|  |  |
| --- | --- |
| **RichFaces tree Attributes** | **PrimeFaces tree Attributes** |
|  | dynamic="true" to send ajax requests when expanding the tree nodes (not static data) |
|  | cache="false" to get the node children from the DB even if the node was expanded before |
| toggleListener | <p:ajax event="expand" global="true" ... /> |
| treeNode h:commandLink action | <p:ajax event="select" global="true" ... /> |
| treeNode h:commandLink | h:outputText |
| treeNode title | p:tooltip for the treeNode |
| treeNode iconExpanded, iconCollapsed, iconLeaf | treeNode icon (which is a style class not image path) |

1. **Notes:**
   1. Remove all unmask statements. No need to unmask after any ajax request even if the component firing the event will be re-rendered. Unless the UI component that fire the ajax request catches itself the ajax event.
   2. Make sure that all the datatables columns have widths because primefaces will set equal widths for all columns by default if no widths set.
   3. For p:ajax requests, the keyword “execute” is replaced with “process”, and “render” with “update”.
   4. Add the PF namespace: xmlns:p=<http://primefaces.org/ui> and remove the richfaces and a4j namespaces.