|  |  |
| --- | --- |
| **Date:** | 28th September 2015 |
| **Time:** | 0930H – 1130H |
| **Venue:** | SMU X Meeting Room 2-2 |
|  |  |
| **Attendees:** | All |
|  |  |
| **Agenda:** | 1. Code review |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

The following was decided after much discussion:

|  |  |
| --- | --- |
| Login | 1. **ViewController**  * Calls executeLogin method based on mapped URL * Check whether email/password is null, store in request attribute   + Null 🡺 redirect to login.jsp   + Not null 🡺 retrieve Demographic object through LoginController * If admin, redirect to bootstrap.jsp, else redirect to home page  1. **LoginController**  * Does validation   + Else, retrieve from database by email through DemographicDAO * Null 🡺 password incorrect or user does not exist * Admin 🡺 Instantiate Demographic object, name=Admin * Student 🡺 check against database record by email using DemographicDAO, create Demographic object  1. **login.jsp**  * footer.jsp contains all JavaScript library imports – put at the bottom of login.jsp to let the page load faster (can try the same for bootstrap)  1. **Tags**  * c: 🡺 javascript tag library * redirect: 🡺 does response.sendRedirect |
| Logout | 1. **ViewController**  * Click logout * Invalidate session * Redirect to login page |
| Login (JSON) | 1. **JSONController**  * Method executeJSONLogin: same method as LoginController, but returns Authenticate object with parameters status and token according to wiki specifications * Method JWTGenerator: generate tokens, shared secret for verification purposes  1. **JSONController vs LoginController**  * JSON doesn’t store objects in session, only reuses functionalities |
| Navbar | 1. **navbar.jsp**  * boolean \_isAdmin == false 🡺 signed in as “name" from getName() method, else signed in as Admin * \_currentUser = Demographic object retrieved from session * If \_currentUser==null 🡺 no session created 🡺 trying to access protected page 🡺 redirect to login page |
| Bootstrap – not done for location & locationlookup (bootstrap) | 1. **DatabaseConfig.java**  * Configuration for hibernate  1. **ViewController**  * Dependencies 🡺 allow the use of Spring stereotype annotations * @Autowire 🡺 instantiates annotated attributes without actually instantiating them * ui.Model 🡺 methods take in Model object, add attribute into object and return object, attributes can be accessed directly from UI using request.getAttribute method * Multipart file 🡺 part of spring enables user to retrieve zip and rar files that are used during bootstrap  1. **Algorithm (method = bootstrap)**  * String[] of files required * 3 HashMap, 1 per csv * LinkedHashMap for fileSuccessList and errorMap – in order to iterate in insertion order * LinkedHashMap for errorMap takes String fileName as key and String[] with line number and errorMsg as value * Method deleteData 🡺 wipe table – done using SQL, executeUpdate method, which wipes table for object calling the method (App, AppLookup, Demographic all have their own deleteData method) * Iterate through files in zip file to find file with required name * ZipInputStream reads the incoming zip file * Use try with resources so that resources will be closed automatically * Put zip file into InputStream followed by ZipInputStream to iterate through entire zip * ZipEntry entry holds each individual file within the zip file * boolean fileFound = true when name of file equals to file name the method looks for by checking through the file array in order to add the files in a certain order, preventing failing foreign key constraint 🡺 file within the iteration is found and the if condition will not be true * boolean filesUploaded checks if file to upload exists (added after finding bootstrap error of displaying success message even though no file was uploaded) * errorMap.put method 🡺 stores fileName as key and LinkedHashMap (key = line number, value = String[] of error messages) as value, which is returned when validate methods are called  1. **Algorithm (method = validateDemographic)**  * New InputStreamReader opened, passed into CSVReader * CSVReader has delimiter \” as some fields in csv file contain “ * Start iterating from line count 1, skip line 1 because it is header row * Iterate through required fields to look for missing field and add to errorList before passing to errorMap * If common file validation fail, no need carry out file specific validation and goes to if(errorList.isEmpty()) condition * Trim strings before assigning to variables * File specifics: check for invalid macAddress, password, email, gender using regex/other conditions * if(errorList.isEmpty()) 🡺 entire line is error free 🡺 put inside demographicMap, which will be reflected in the main loop because it is passed as a parameter, therefore no need to return demographicMap * if(!errorList.isEmpty()) 🡺 put line count and array of error messages into errorMap * fileSuccessList 🡺 put fileName and demographicMap.size(), which is the size of the map containing successfully loaded entries * call batchCommit method using DemographicDAO  1. **Algorithm (method = batchCommit)**  * Although we are using SessionFactory from Hibernate, we open a new session in order to do batch commit * Transaction 🡺 allow committing to database * Batch commit saves 30 by 30 Demographic object to session * Flush after every 30 is committed, but the committed objects stay in Transaction object * Each batch commit is 1 session, but complete data is committed all at once * Batch size 30 was selected based on recommendations during research and it is working well * In order to use batch statements, need to reflect in java code (in batchCommit method), db.url under application.properties requires &rewriteBatchedStatements=true and hibernate.jdbc.batch\_size must be set 🡺 enables batching to the database we are accessing * hibernate.hbm2ddl.auto under application.properties is set to update. Auto is done upon deployment, update 🡺 create only if table didn’t exist before  1. **Algorithm (method = validateAppLookup)**  * Same as the validation done on demographic.csv with same algorithm used for checking Demographic but checking different fields * regex (typeA | typeB) 🡺 match as long as String being checked is equals to either of the expressions in the regex provided * put in rows with no errors into appLookupMap  1. **Algorithm (method = validateApp)**  * Requires appLookupMap, appMap and demographicMap * Have to compare with keys in appLookupMap and demographicMap to so that foreign key constraint will not fail * Check for missing field * Check for file specific validations:   + compare with appData: concatenate timestamp with macAddress as key to check for duplicate   + only replace duplicates with valid appID   + appMap.containsKey(appID) and errorList.isEmpty()   + parse String timestamp to Date, catch ParseException for SimpleDateFormat   + AppPK 🡺 composite primary key class   + Put in required maps & return errorMap |
| Bootstrap (Add Data – demographic and app) | * Check if App exist in database * Calls retrieveAll() method in AppDAO and returns a List, iterate through to store in a HashMap * daoMap!=null 🡺 check if daoMap already contains entry * comparison to daoMap is the extra step for addData from bootstrap |
| BootstrapError object | Common object for JSON and view  String status = success  errorMap not empty 🡺 iterate through and add to error list  return status, file success list and error list |
| ViewController | Add attribute to model and display it on jsp pages  Null check to prevent NullPointerException |
| JSONController | Return ObjectNode (part of JacksonAPI)  ObjectNode works similar to HashMap  [] 🡺 array  Verify method returns username or null if username doesn’t exist  NumLoadedList reconstructs JSON object  Use node.putPOJO method  Already created entity (bootError) in the form that matches specifications so not required to format further before returning |

The meeting was adjourned at 11.30 am.

Prepared by,

Yap Swee Lian, Jennifer

Vetted and edited by,

Koh Chu Qian