## Task 1) Database tables

Professor Phillips has already done most of the work for this task[1]. Someone on your team will need to run the database scripts to create the employee and activeuser tables against your team's database.

- Define a table to store employee information
  - Record ID
    - This is the primary key of the table
  - First name
  - Last name
  - Employee ID
    - Should support up to 5 numerical characters in length
  - Active
    - Indicates whether an employee is active or inactive
  - Classification or role
    - Must maintain 3 or more values starting with: general manager, shift manager, and cashier
  - Manager
    - This is a reference, via the Record ID, to another record in the Employee table
    - May be empty
  - Password
  - Created On
    - Timestamp

- Define a table to store active user information
  - Record ID
    - This is the primary key of the table
  - Employee ID
    - This will be a reference to a Record ID in the employee table
  - Name
    - This will store the complete name of the user
  - Classification or role
    - Must maintain 3 or more values starting with: general manager, shift manager, and cashier
    - Should be modeled off of the same field in the employee table
  - Session key
    - A string
  - Created On
    - Timestamp

- Define server-side code to model the database within the server-side code
  - Model/entity objects for working with individual records
  - Repository objects for interacting with the tables in the database
    - CRUD operations

[1] See Employee.sql in the Sprint2Starter branch of the RegisterApp-DataDefinition repository that you cloned to your team's organization in Github

## Task 2) Sign In - View

- Allow for displaying an error message when one is available
  - See similar constructs in the existing productListing and productDetail views

- Use a form element[1,2] to submit user credentials via a POST[3,4] request
  - Form should contain a few elements
    - Input element[5,6] of type number[7] for the employee ID
    - Input element[5,6] of type password[7] for the password
    - Input element[5,6] of type submit[7] with text “Sign In”
  - Form should perform validation on submission[8,9]
    - Invoke the method to be defined in the “Sign In – Server-side functionality” task

When an HTML form is submitted it performs an HTTP request of the type specified in the “method”[10] attribute of the element to the route, or endpoint, specified in the “action”[11] attribute of the element. It sends the contents of the form’s child elements as the payload of the HTTP request, using each element’s “name” attribute as the associated property’s name. For example, if your form contained the following input element – <input name=”xyz” type=”text” value=”abc” /> – then the HTTP request payload would contain – xzy <= “abc” – along with any other inputs that were part of the form.

- Views are defined in HTML, but use a templating engine to allow for dynamic content from the server
  - The Java templating engine is Thymeleaf[12]
  - The NodeJS templating engine is EJS[13]
  - There are other templating engines for either environment, but these are the ones that I used
    - If you prefer another templating engine feel free to make the necessary adjustments

[1] https://www.w3schools.com/tags/tag_form.asp
[2] https://www.w3schools.com/html/html_forms.asp
[3] https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html
[4] https://www.w3schools.com/tags/ref_httpmethods.asp
[5] https://www.w3schools.com/tags/tag_input.asp
[6] https://www.w3schools.com/html/html_form_elements.asp
[7] https://www.w3schools.com/html/html_form_input_types.asp
[8] https://www.w3schools.com/jsref/event_onsubmit.asp
[9] https://www.formget.com/javascript-onsubmit/
[10] See the heading “The Method Attribute” at https://www.w3schools.com/html/html_forms.asp
[11] See the heading “The Action Attribute” at https://www.w3schools.com/html/html_forms.asp
[12] https://www.thymeleaf.org/doc/tutorials/3.0/usingthymeleaf.html
[13] https://ejs.co/#docs

## Task 3) Sign In - Client-side functionality

- Build a method to validate input on form submission[1,2]
  - Validate that the employee ID is numeric and not blank
  - Validate that the password is not blank

[1] https://www.w3schools.com/jsref/event_onsubmit.asp
[2] https://www.formget.com/javascript-onsubmit/

## Task 4) Sign In - Server-side routing (Java)

- "API" objects ("models/api" package)
  - Define an EmployeeSignIn.java class
    - This object's contents will represent the payload from the client for signing in a user
    - Properties
      - employeeId of type String
      - password of type String

- Controllers ("controllers" package)
  - Define a SignInRouteController.java class
    - To get started you can use the existing ProductDetailRouteController as an example
      - Routes, method names, etc will be different, but it is useful as a pattern
      - This is used for handling web site navigation/document serving
    - Define a route handler for requesting the view/document
      - GET request with "/" route/endpoint path
      - Parameters
        - Map<String, String> object
          - Use the @RequestParam[1] annotation
          - This parameter will contain any query string[2,3] parameters passed in the URL
      - Functionality
        - Make use of functionality built in Task #5
        - If employees exist
          - Should serve up the Sign In view/document
        - If no employees exist
          - Should immediately redirect[4] to the Employee Detail view/document
    - Define a route handler for signing in
      - POST request with "/" route and `consumes` a value of type `MediaType.APPLICATION_FORM_URLENCODED_VALUE`
      - Parameters
        - EmployeeSignIn object (defined above in "API" objects)
        - HttpServletRequest[5] object
          - Provides access to the request session ID
      - Functionality
        - Should verify employee credentials: employee ID and password
        - If correct should
          - Create an record in the activeuser table using
            - The current session ID
            - The employee details associated with the provided credentials
            - Make use of functionality built in Task #5
          - Redirect[4] to the Main Menu view/document
        - If not correct should serve up the Sign In view/document again
          - Should provide an error message indicating that the sign in was not successful
  - Define a SignInRestController.java class
    - To get started you can use the existing ProductRestController as an example
      - Routes, method names, etc will be different, but it is useful as a pattern
      - This is used for handling AJAX requests

- Define a route handler to remove an active user
  - DELETE request with a route to be coordinated with the developer working on Task #3
    - Could be “/api/signOut”
  - Functionality
    - Should remove any record in the activeuser table associated with the current session ID
      - Make use of functionality built in Task #5
    - Should redirect to the Sign In view/document
      - Return an ApiResponse object with the “redirectUrl” set appropriately

[1] https://www.baeldung.com/spring-request-param
[2] https://en.wikipedia.org/wiki/Query_string
[3] See the heading “The GET Method” at https://www.w3schools.com/tags/ref_httpmethods.asp
[4] https://www.baeldung.com/spring-redirect-and-forward#redirect-with-the-prefix-redirect
[5] https://tomcat.apache.org/tomcat-5.5-doc/servletapi/javax/servlet/http/HttpServletRequest.html

## Task 4) Sign In - Server-side routing (NodeJS)

- Controllers (“controllers” directory)
  - Define routes/signInRoutes.ts and controllers/signInRouteController.ts objects
    - To get started you can use the existing routes/productDetailRoutes.ts and controllers/productDetailRouteController.ts as an example
      - Routes, method names, etc will be different, but it is useful as a pattern
    - Define a route handler[1] for requesting the view/document
      - routes/signInRoutes.ts
        - GET request with “/” route/endpoint path
      - controllers/signInRouteController.ts
        - Parameters
          - Request[2]
          - Response[3]
        - Functionality
          - Make use of functionality built in Task #5
          - If employees exist
            - Should serve up the Sign In view/document
          - If no employees exist
            - Should immediately redirect[4] to the Employee Detail view/document
    - Define a route handler[1] for signing in
      - routes/signInRoutes.ts
        - POST request with “/” route
      - controllers/signInRouteController.ts
        - Parameters
          - Request[2]
          - Response[3]
        - Functionality
          - Should verify employee credentials: employee ID and password
          - If correct should
            - Create an record in the activeuser table using
              - The current session ID
                - Use the Request[2] session[5] property
              - The employee details associated with the provided credentials
                - Use the Request[2] body[6] property
              - Make use of functionality built in Task #5
            - Redirect[4] to the Main Menu view/document
          - If not correct should serve up the Sign In view/document again
            - Should provide an error message indicating that the sign in was not successful
    - Define a route handler to remove an active user
      - routes/signInRoutes.ts
        - DELETE request with a route to be coordinated with the developer working on Task #3
          - Could be “/api/signOut”
      - controllers/signInRouteController.ts
        - Functionality
          - Should remove any record in the activeuser table associated with the current session ID

        - Make use of functionality built in Task #5
    - Should redirect to the Sign In view/document
    - Return an ApiResponse object with the “redirectUrl” set appropriately

[1] https://expressjs.com/en/guide/routing.html
[2] https://expressjs.com/en/api.html#req
[3] https://expressjs.com/en/api.html#res
[4] https://expressjs.com/en/api.html#res.redirect
[5] https://expressjs.com/en/resources/middleware/session.html
[6] https://expressjs.com/en/api.html#req.body

## Task 5) Sign In - Server-side functionality (Java)

- "Command" objects
  - "commands/employees" package
    - Define an ActiveEmployeeExistsQuery.java class
      - Functionality
        - Query if any active employees exist in the database
          - Use the existing EmployeeRepository.existsByIsActive() method
        - If there are no active employee records in the database then
          - Throw a NotFoundException
    - Define an EmployeeSignInCommand.java class
      - Properties
        - EmployeeSignIn object
          - Defined in Task #4
        - Current session key
      - Functionality
        - Validate the incoming Employee request object
          - Employee ID should not be blank and should be a number
          - Password should not be blank
        - Query the employee by their employee ID
          - Use the existing EmployeeRepository.queryByEmployeeId() method
          - Verify that the employee exists
          - Verify that the passwords from the request and the database match
            - Use Arrays.equals()[1]
        - In a transaction - use the @Transactional[2] annotation for your method
          - Query the activeuser table for a record with the employee ID
          - If an activeuser record already exists
            - Update the entity's sessionKey property with the current session key
            - Use the existing ActiveUserRepository.save() method
          - Else
            - Create a new activeuser record in the database
              - Set the session key
              - Set the necessary employee details
              - Use the existing ActiveUserRepository.save() method
  - "commands/activeUsers" package
    - Define an ActiveUserDeleteCommand.java class
      - Properties
        - Current session key
      - Functionality
        - Validate the incoming Employee request object
          - First name should not be blank
          - Last name should not be blank
        - In a transaction - use the @Transactional[1] annotation for your method
          - Query the activeuser table for a record with the provided session key
            - Use the existing ActiveUserRepository.findBySessionKey() method
          - Delete the activeuser record
            - Use the existing ActiveUserRepository.delete() method

[1] https://docs.oracle.com/javase/8/docs/api/java/util/Arrays.html#equals-byte:A-byte:A-

[2] https://www.baeldung.com/transaction-configuration-with-jpa-and-spring#the-transactional-annotation

## Task 5) Sign In - Server-side functionality (NodeJS)

- "Command" objects
  - "controllers/commands/employees" directory
    - Define an activeEmployeeExistsQuery.ts object
      - Functionality
        - Query if any active employees exist in the database
          - Use the existing EmployeeRepository.queryActiveExists() method
        - Return an appropriate response depending on whether or not an active employee exists in the database
    - Define an employeeSignInCommand.ts object
      - Functionality
        - Parameters
          - Define a sign in request interface
            - Properties
              - employeeId of type string
              - password of type string
          - Express.Session[1]
        - Validate the incoming request object
          - Employee ID should not be blank and should be a number
          - Password should not be blank
        - Query the employee by their employee ID
          - Use the existing EmployeeRepository.queryByEmployeeId() method
          - Verify that the employee exists
          - Verify that the passwords from the request and the database match
        - In a transaction - use the existing DatabaseConnection.createTransaction() method
          - Query the activeuser table for a record with the employee ID
          - If an activeuser record already exists
            - Update the entity's sessionKey property with the current session key
            - Invoke the update() method on the queried object to persist the change to the database
          - Else
            - Create a new activeuser record in the database
              - Set the session key
              - Set the necessary employee details
              - Use the existing ActiveUserModel.create() method
  - "controllers/commands/activeUsers" directory
    - Define an clearActiveUserCommand.ts object
      - Functionality
        - Parameters
          - Express.Session[1] id
        - In a transaction - use the existing DatabaseConnection.createTransaction() method
          - Query the activeuser table for a record with the provided session key
            - Use the existing ActiveUserRepository.queryBySessionKey() method
          - Invoke destroy() on the queried activeuser entity object to delete the record from the database

[1] https://expressjs.com/en/resources/middleware/session.html

## Task 6) Main Menu - View

- Allow for displaying an error message when one is available
  - See similar constructs in the existing productListing and productDetail views

- Define several button elements[1] that will perform navigation when clicked[2]
  - “Start Transaction”
  - “View Products”
  - “Create Employee”
    - Should only be visible to elevated users, managers
  - “Sales Report”
    - Should only be visible to elevated users, managers
  - “Cashier Report”
    - Should only be visible to elevated users, managers

- Allow for a user to sign out
  - Use an image element[3] with a material design icon[4]
  - Will sign out the user and navigate to the sign in view when clicked[2]
  - Coordinate with the developers working on the other view tasks so that this control looks the same on each page

- Views are defined in HTML, but use a templating engine to allow for dynamic content from the server
  - The Java templating engine is Thymeleaf[5]
  - The NodeJS templating engine is EJS[6]
  - There are other templating engines for either environment, but these are the ones that I used
    - If you prefer another templating engine feel free to make the necessary adjustments

[1] https://www.w3schools.com/tags/tag_button.asp
[2] Click event handlers will be defined in the “Main Menu - Client-side functionality” task
[3] https://www.w3schools.com/tags/tag_img.asp
[4] Pick an appropriate image like “input”, or similar. https://material.io/resources/icons/?icon=input&style=baseline
[5] https://www.thymeleaf.org/doc/tutorials/3.0/usingthymeleaf.html
[6] https://ejs.co/#docs

## Task 7) Main Menu - Client-side functionality

- On page load[1], define “click” event handlers[2] for the various navigation buttons defined in the view
  - Start transaction
    - Should display error “Functionality has not yet been implemented.” on the view when clicked[3]
  - View products
    - Should navigate[4] to the existing productListing view when clicked
  - Create employee
    - When visible, should navigate[4] to the new employeeDetail view when clicked
  - Sales report
    - Should display error “Functionality has not yet been implemented.” on the view when clicked[3]
  - Cashier report
    - Should display error “Functionality has not yet been implemented.” on the view when clicked[3]

- On page load[1], define a “click” event handler[2] for the sign out image when defined
  - Should be defined in a common module that can be accessed from each of the client-side modules (a view/script(s)/style(s) triplet)
    - Probably master.js
  - Should make an HTTP request, via AJAX[5], to the server to clear the active user
    - Perform a DELETE request[6,7]
  - Redirect[8] to the Sign In view upon completion of the request

[1] https://developer.mozilla.org/en-US/docs/Web/API/Window/DOMContentLoaded_event
[2] https://www.w3schools.com/js/js_htmldom_eventlistener.asp
[3] You can make use of the existing *displayError(errorString)* method defined in master.js
[4] https://www.w3schools.com/jsref/met_loc_assign.asp
[5] https://www.w3schools.com/xml/ajax_intro.asp
[6] You can make use of the existing *ajaxDelete()* method defined in master.js
[7] https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html
[8] https://www.w3schools.com/jsref/met_loc_replace.asp

## Task 8) Main Menu - Server-side routing (Java)

- Controllers (“controllers” package)
  - Define a MainMenuRouteController.java class
    - To get started you can use the existing ProductDetailRouteController as an example
      - Routes, method names, etc will be different, but it is useful as a pattern
      - This is used for handling web site navigation/document serving
    - Define a route handler for requesting the view/document
      - GET request with “/mainMenu” route/endpoint path
      - Parameters
        - Map<String, String> object
          - Use the @RequestParam[1] annotation
          - This parameter will contain any query string[2,3] parameters passed in the URL
        - HttpServletRequest[4]
          - Use this to access the current session and the associated user
      - Functionality
        - If there is an active user for the current session then
          - Should add any error messages received in the query string parameters to the view
          - Should serve up the Main Menu view/document
        - Else
          - Should immediately redirect[5] to the Sign In view/document with an appropriate error message

[1] https://www.baeldung.com/spring-request-param
[2] https://en.wikipedia.org/wiki/Query_string
[3] See the heading “The GET Method” at https://www.w3schools.com/tags/ref_httpmethods.asp
[4] https://tomcat.apache.org/tomcat-5.5-doc/servletapi/javax/servlet/http/HttpServletRequest.html
[5] https://www.baeldung.com/spring-redirect-and-forward#redirect-with-the-prefix-redirect

## Task 8) Main Menu - Server-side routing (NodeJS)

- Controllers (“controllers” directory)
  - Define routes/mainMenuRoutes.ts and controllers/mainMenuRouteController.ts objects
    - To get started you can use the existing routes/productDetailRoutes.ts and controllers/productDetailRouteController.ts as an example
      - Routes, method names, etc will be different, but it is useful as a pattern
    - Define a route handler[1] for requesting the view/document
      - routes/mainMenuRoutes.ts
        - GET request with “/mainMenu” route/endpoint path
      - controllers/mainMenuRouteController.ts
        - Parameters
          - Request[2]
          - Response[3]
        - Functionality
          - If there is an active user for the current session then
            - Should add any error messages received in the query string parameters to the view
            - Should serve up the Main Menu view/document
          - Else
            - Should immediately redirect[4] to the Sign In view/document with an appropriate error message

[1] https://expressjs.com/en/guide/routing.html
[2] https://expressjs.com/en/api.html#req
[3] https://expressjs.com/en/api.html#res
[4] https://expressjs.com/en/api.html#res.redirect