Easysim

Technical Requirement Document

Project Overview:

EasySIM is a competency-driven real job scenario-based Simulated Learning and Assessment Platform designed to accelerate workforce performance and provide tangible insights into human capital growth against an organization's L&D investments.

At its core, EasySIM is not an ordinary eLearning platform, can help define, design, create, and manage business-specific competency-driven Simulated Learning and Assessment events with an ability to gather, process, and analyze diverse data points to provide deep insights into the present competency and performance state of the workforce as well as design a clear path for the organization's growth.

EasySIM is one of the world's top competency-driven simulation-based learning platforms that can help organizations easily create decision tree-based design that helps train their employees through branching scenario-based simulations and assessments.

Key features

- **Competency Assessment And Development**
- **Simulation Library Building**
- **Performance Development**
- **Assessment**
- **Learning Experience**
- **AI Based Proctoring Software**
- **Google Meet / MS Team Meeting Synch**

Technical Requirements

Server Requirements

- ***Operating System:*** Linux (recommended) / Windows / Mac
- ***Current OS:*** Centos 7 / 8
- ***Web Server:*** Apache 2.4 / IIS / Nginx (latest stable versions recommended)

- Ensure that mod_rewrite and Port 80/443 (for Apache) or equivalent (for Nginx) is enabled for clean URL support.
- ***Database:*** MySQL 5.7+, restrict mode disable
- ***Server coding language:*** HyperText Preprocessor (PHP) - details below
- **Scripting Language:** HTML5, CSS, Javascript, Jquery, Ajax
- Google Calendar / MS Team
- Proctor API's
- Emotions analysis emotion_analysis_report
- Sentiment Analysis
- Chatgtp API

PHP Extension and Setting

- Pdo_mysql
- Openssl
- Mbstring
- Xml
- Json
- Ctype
- Zip
- gd (optional, if image processing is required)
- Composer
- CURL
- Intl memory_limit: 512MB
- post_max_size: 80MB
- upload_max_filesize: 1024MB
- max_execution_time: 60 seconds

Videojs Library / Package

- VideoJs Packages
- Videojs-vimeo
- Videojs youtube

Third Party JS Package

- Ckeditor
- Datatable
- Fullcalender.io
- Datetime picker
- Canvas js (graph)
- Owl carousel
- Sweet alert

- fancybox

AI Based Package / Proctor API's

AI based proctor API's developed in python, flask, pm2 service, language, emotion api through *ai practo video analysis*, below is the configuration steps

- **Server Port check config file** - Verify purpose that API running port
  etc/ngnix/
  emotionapi.easysim.app: 8881
- Go to Project Path inside Project =>
  /home/PractoAdmin/ai-practo-video-analysis
- Package confirmation (optional) run this command inside project
  pip3 install -r requirements.txt
- FLASK_APP env variable set (optional)
  FLASK_APP=app.py
- flask server run (optional) -Test to confirm any code issues
  flask run -h 0.0.0.0 -p 8881 for (emotionapi)
- Step 7 pm2 service start in inside the API project directory
  /home/PractoAdmin/ai-practo-video-analysis/emotionapi....
- Check PM2 Status
- pm2 service
  Pm2 start app.py --name flask-app-emotionapi --interpreter=python3
  Pm2 status // if service stop then need to start as below
  pm2 restart "flask run" // check pm2 service running status as below
  Pm2 status

Generative AI ChatGpt

Integrated Generative AI in easysim Text-DaVinci-003 model within OpenAI's GPT-3 (Generative Pre-trained Transformer 3) family of language models. It is one of the most advanced models in the GPT-3 lineup, designed to understand and generate human-like text based on the input it receives. Below is the working process and characteristics of this model

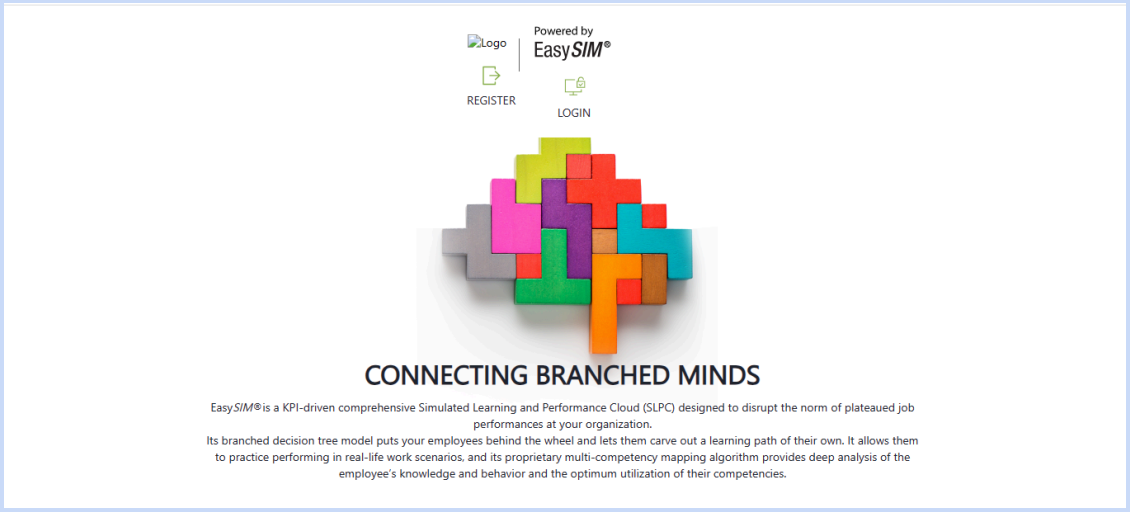
- Text-DaVinci-003, a highly advanced model in the GPT-3 family.
- High-Quality Text Generation module synch
- It can handle various tasks such as text completion, summarization, translation, question-answering, dialogue systems, and more, making it highly versatile.
- Efficiency and Productivity of the systems and Users
- Advanced Understanding and Contextual Awareness - Understands and follows complex instructions, allowing it to perform detailed and nuanced tasks. Also, maintains context over long pieces of text, ensuring responses are relevant and consistent

Integration Steps:

- Step 1: Set Up Your Node.js
- Step 2: Get Your OpenAI API Key. Login in Openai platform and generate API key
- Step 3: Create a .env File and putted APIkey and PORT
  PORT=3002
  OPENAI_API_KEY=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
- Step 4: Create the Integration Script Create index.js
- Step 5: Define Proxy Path
- Run API

Landing Page

Template Name: index.php



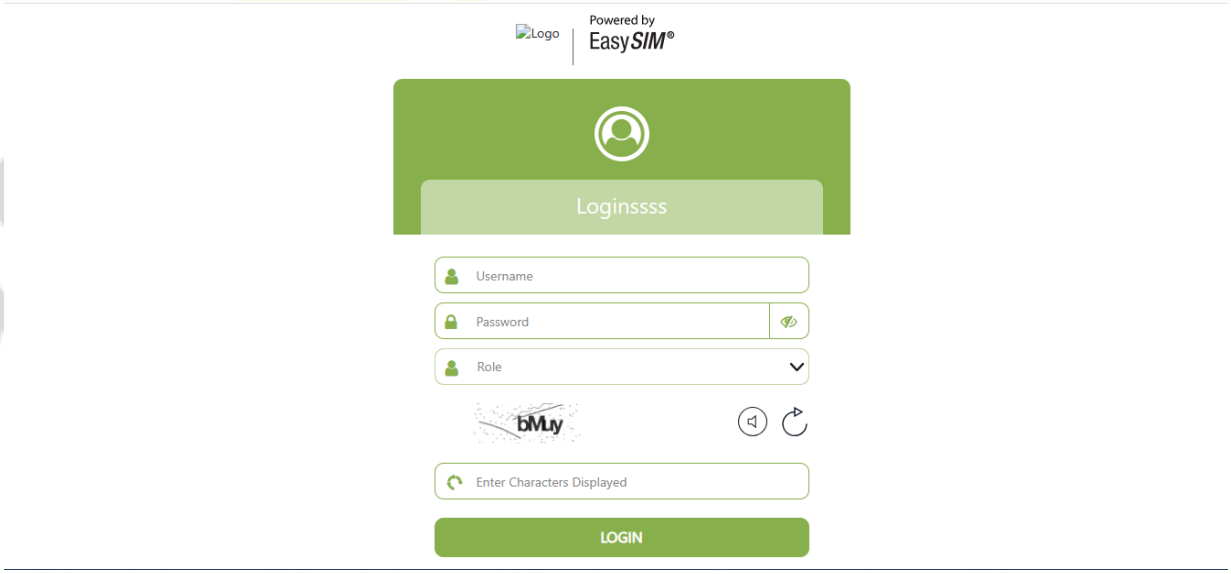
1. Configure database
2. Configure Calendar API
3. Used **webBranding()**:- Global function from "db.class.php". This is used for branding. Like Color, logo, font family, font size, etc.
4. Used **getCMSPage()**:- Global function from "db.class.php". It fetches the data from the database with the condition of page-id and shows the data.
5. Register Logo:- It is changed as per domain.

Login Page

Template Name:- Login.php

Roles base access:

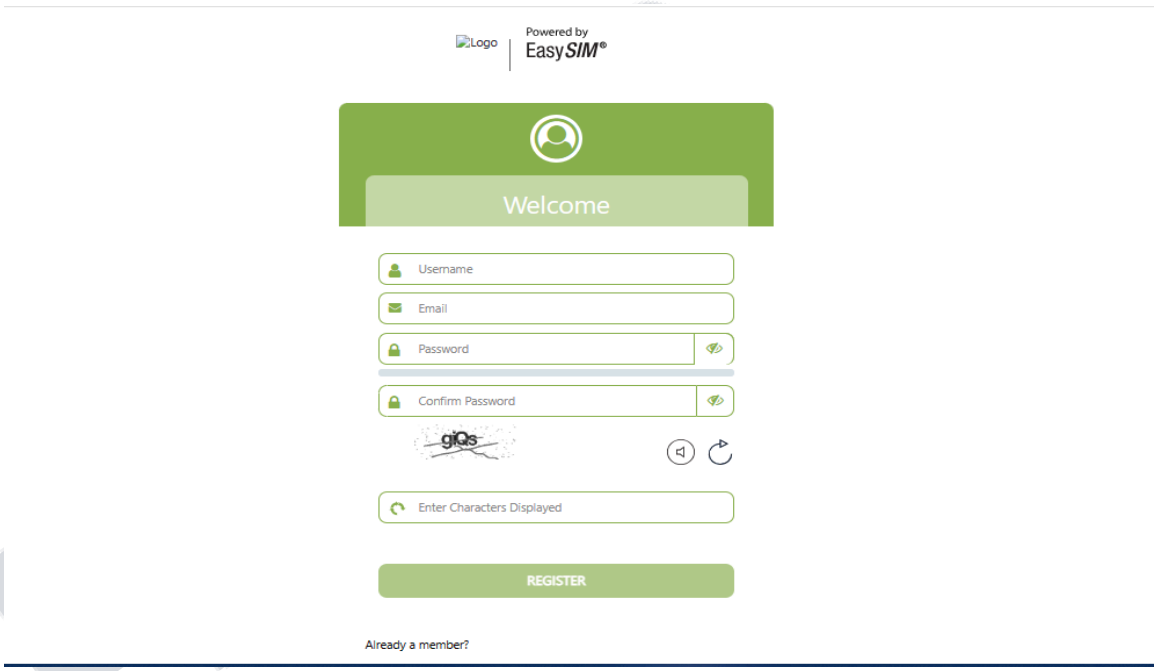
- Super Admin
- Admin
- Client Admin
- Learner
- Reviewer
- Author
- Educator
- Partner



1. Configure database
2. Used **webBranding()**:- Global function from "db.class.php". This is used for branding. Like Color, logo, font family, font size, etc.
3. Used **isLogin()**:- Global function from "db.class.php". A user is logged in based on session data and redirects them appropriately based on their status and role.
4. Used **getBaseURL()**:- Global function from "db.class.php". The getBaseUrl() function in PHP constructs and returns the base URL of the current application.
5. Role:- Define role, which type of role is available like Admin, Author, Reviewer, Learner, Educator, etc.
6. **getCaptchaHtml()**:- This is used for displaying a captcha on a webpage.

Register Page

Template Name:- Register.php



1. Configure database
2. Used **webBranding()**:- Global function from “db.class.php”. This is used for branding. Like Color, logo, font family, font size, etc.
3. Used **getBaseURL()**:- Global function from “db.class.php”. The getBaseUrl() function in PHP constructs and returns the base URL of the current application.
4. Role:- Define role, which type of role is available like Admin, Author, Reviewer, Learner, Educator, etc.
5. **getCaptchaHtml()**:- Global function from “securimage/securimage.php”. This is used for displaying a captcha on a webpage.

ES Role based access (RBAC)

Super Admin

Overview

After the Super admin authenticate process the user lands on the dashboard page and sees the below elements.

- Dashboard statics on the top of the page
  - Clients and subclient counts
  - Role based users counts
  - Search client based information and statics data
  - Client based allocated space donuts chart

- Manage Clients Module

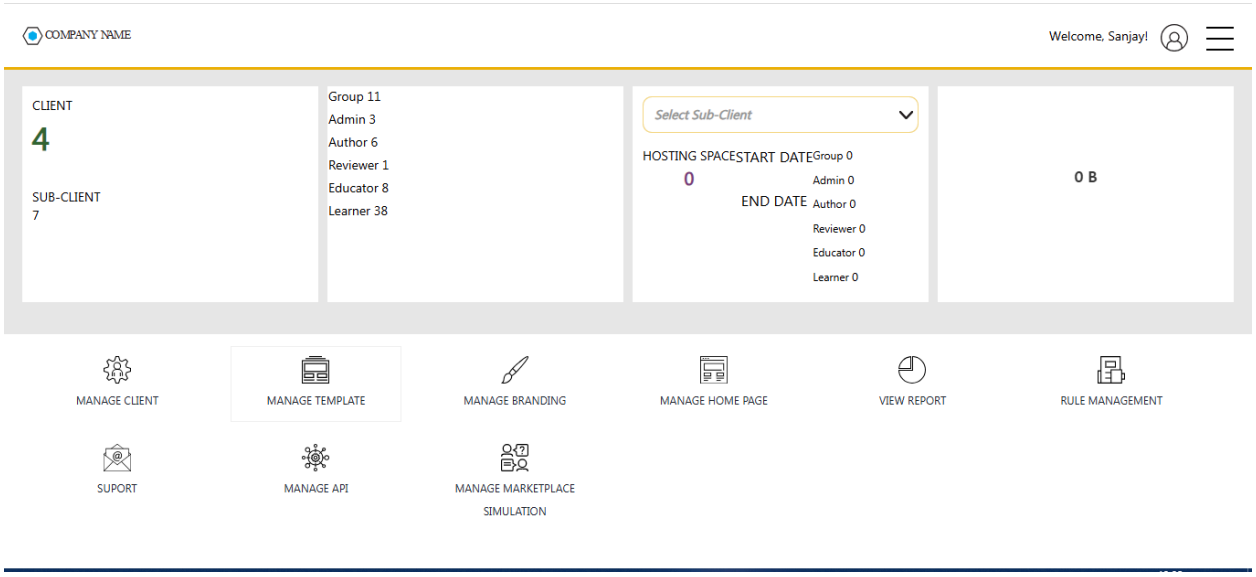
- Manage Template Module - This area we can manage different email templates with custom html layouts with predefined users variables like name, email etc

- Manage Topic and Topic Items:

Code Stuff

Dashboard

Template Name: super-admin-dashboard.php



PHP Code Functions

1. **checkLogin()**
   Global function from “db.class.php”. The checkLogin function ensures that a user is properly logged in by verifying that all essential session variables are set and not empty.
2. **checkAccountType()**
   Global function from “db.class.php”. The checkAccountType function determines the type of account based on the user's role stored in the session. It returns a corresponding numerical value for each role or FALSE if the role is not recognized.
3. **getUsedHostingData()**
   Global function from “db.class.php”. The function aims to retrieve the used hosting data for a specific client or all clients. It calculates the total size of the hosting space used by the client's domains.

Third Party JS Library

1. CanvasJS

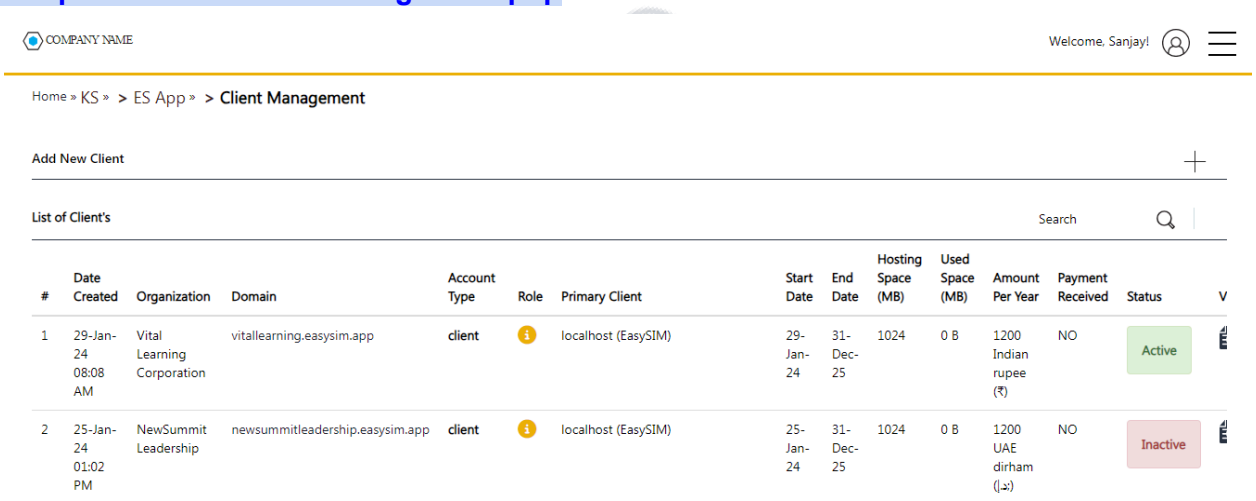
JS Functions Used

1. **function defaultData() {}**
   The defaultData function performs an asynchronous HTTP POST request to *'includes/main-process.php'* to retrieve default data from the server and updates various HTML elements with the received data. During the request, a loading overlay is shown to the user. If the request is successful and the server responds with a success flag, the HTML elements are populated with the corresponding data. If there is an error, either in the request or in the response, an error message is displayed to the user using SweetAlert.
2. **$('.multiselect-ui').multiselect({})**
   The given code initializes and configures the multiselect plugin for a dropdown, enabling features such as filtering and setting a maximum height. It also defines a change event handler that manages the selection process by enabling or disabling options based on a specified maximum count, sending AJAX requests to *'includes/main-process.php'* to fetch additional data when the selection changes, and updating the page's content accordingly. If the maximum count is not reached, it ensures all options are enabled and resets specific content areas.
3. **$.getJSON('includes/main-process.php?getClients=true', function(res) {})**
   This jQuery to make an asynchronous HTTP GET request to *'includes/main-process.php*?getClients=true' to fetch JSON data and then processes this data to update a multiselect dropdown.
4. **var UsedHostingData = new CanvasJS.Chart("UsedHostingData", {})**
   The provided code initializes and configures a doughnut chart using the CanvasJS library. It sets various properties such as the color set, height, background color, animation, and theme. It defines a single data series for the chart, specifying its type and

data points. The data points are dynamically populated using PHP to encode the data in JSON format. Finally, the chart is rendered on the page.

Manage Client

Template Name: client-management.php



#	Date Created	Organization	Domain	Account Type	Role	Primary Client	Start Date	End Date	Hosting Space (MB)	Used Space (MB)	Amount Per Year	Payment Received	Status	V
1	29-Jan-24 08:08 AM	Vital Learning Corporation	vitallearning.easysim.app	client		localhost (EasySIM)	29-Jan-24	31-Dec-25	1024	0 B	1200 Indian rupee (₹)	NO	Active	
2	25-Jan-24 01:02 PM	NewSummit Leadership	newsummitleadership.easysim.app	client		localhost (EasySIM)	25-Jan-24	31-Dec-25	1024	0 B	1200 UAE dirham (د.إ)	NO	Inactive	

PHP Functions & Codes

1. **checkLogin()**
   Global function from "db.class.php". The checkLogin function ensures that a user is properly logged in by verifying that all essential session variables are set and not empty.
2. **checkAccountType()**
   Global function from"db.class.php". The checkAccountType function determines the type of account based on the user's role stored in the session. It returns a corresponding numerical value for each role or FALSE if the role is not recognized.
3. **SQL Queries**
   *$sqlr = "SELECT *, DATE_FORMAT(create_date_time, '%d-%b-%y %h:%i %p') AS create_date, DATE_FORMAT(start_date, '%d-%b-%y') AS sdate, DATE_FORMAT(end_date, '%d-%b-%y') AS edate FROM client_tbl WHERE type != 'super_admin' ORDER BY client_id DESC";*
   The SQL query retrieves data from the client_tbl table with specific formatting for date columns and applies certain conditions and ordering.
4. **getClientMapData($clientId)**
   Global function from "db.class.php". The getClientMapData function retrieves the domain_name and organization of a client associated with a given sub_client_id from the mapping_tbl and client_tbl tables. It uses a left join to ensure that all mappings are considered, prepares and executes the SQL query to prevent SQL injection, and returns the fetched data if any rows are found, or an empty array if no rows are found.

Third Party JS Library

1. InputMask
2. Bootstrap-DatetimePicker

JS Functions Used

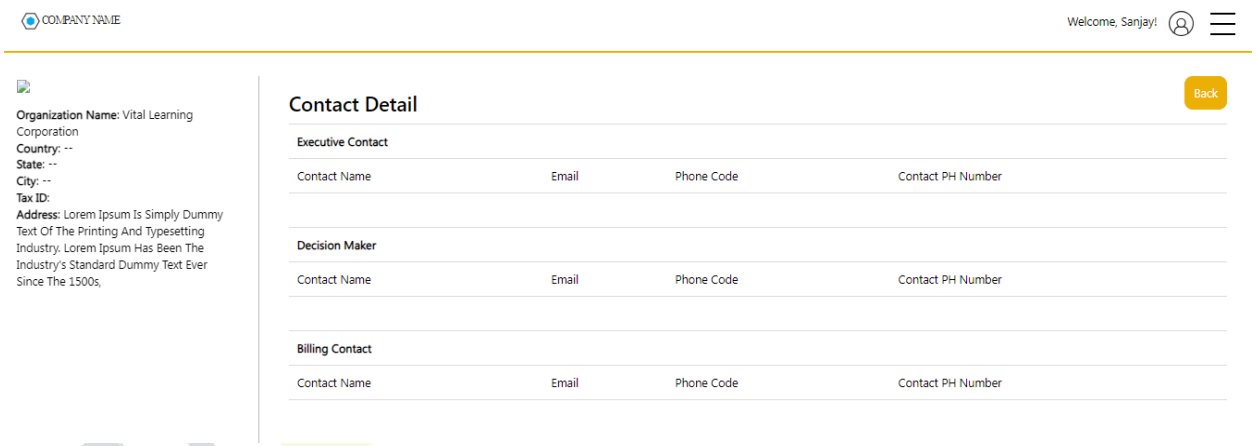
1. **$('input[name=mail_type]').on('click', function() {})**
   The jQuery code snippet sets up an event handler for click events on input elements with the name mail_type. Depending on the value of the clicked input, it toggles the visibility of certain elements and clears the values of specific input fields.
2. **$('.checked_all').on('change', function() {})**
   The jQuery code snippet sets up an event handler that synchronizes the checked state of a group of checkboxes based on the checked state of a master checkbox.
3. **$('.tmp_type').change(function() {})**
   The jQuery code sets up an event handler that monitors changes to a group of checkboxes with the class tmp_type. Depending on the state of these checkboxes, it updates the state of a master checkbox with the class checked_all.
4. **$('#accounttype').on('change', function(){})**
   The jQuery code snippet sets up an event handler that responds to changes in the selection of an element with the ID accounttype. Based on the selected value, it dynamically modifies the options and behavior of two multiselect dropdowns (#role and #loginType).
5. **$.ajax({...url: "includes/currencies.json",...})**
   This jQuery code snippet makes an AJAX request at *"includes/currencies.json"* to fetch data from a JSON file (includes/currencies.json) containing currency information. Upon successful retrieval, it dynamically populates a <select> dropdown with the currency options.
6. **$.getJSON('includes/main-process.php?type=getCountry', function(data) {})**
   This jQuery code snippet retrieves data from the server in JSON format using the $.getJSON() function from *'includes/main-process.php?type=getCountry'*. It fetches country information from a PHP script (main-process.php) with the type parameter set to getCountry. Upon successful retrieval, it dynamically populates a <select> dropdown with country options.
7. **jQuery('body').on('change', '.country', function() {})**
   This jQuery code snippet sets up an event handler to handle changes in the selection of a <select> dropdown with the class country. When the user selects a country, it dynamically populates another dropdown with state options corresponding to the selected country.
8. **jQuery('body').on('change', '.state', function() {})**
   This jQuery code snippet sets up an event handler that handles changes in the selection of a <select> dropdown with the class state. When a state is selected, it dynamically populates another dropdown with city options corresponding to the selected state.

9. **$('.form_datetime').datetimepicker({})**
   This jQuery code snippet initializes a datetime picker widget on elements with the class form_datetime using the datetimepicker plugin.
10. **$('#role').multiselect({})**
   This jQuery code snippet initializes a multi-select dropdown on an element with the ID role using the multiselect plugin.
11. **$('#db_name').keydown(function(event){})**
   This jQuery code snippet sets up an event handler that is triggered when a key is pressed down in an input field with the ID db_name. The code implements specific conditions to restrict certain actions based on the length of the input value and the key pressed.
12. **$("#domain_name").on('change', function(){})**
   This jQuery code snippet attaches an event handler to an input field with the ID domain_name. The handler is triggered whenever the value of the input field changes. The purpose of this handler is to validate the input against specific criteria and provide feedback to the user if the validation fails.
13. **$('#upload').click(function(){})**
   This jQuery code snippet sets up an event handler for when the button with the ID upload is clicked. The code handles the process of uploading a file (logo) to the server (*"includes/upload.php"*) and provides feedback to the user.
14. **$('.delete_logo').click(function() {})**
   This jQuery code snippet sets up an event handler for when an element with the class delete_logo is clicked. The code handles the process of deleting a logo file from the server (*"includes/delete-file.php"*) and provides feedback to the user.
15. **$('.delete').click(function () {})**
   This jQuery code snippet sets up an event handler to manage the deletion of a client account through AJAX request to *"includes/main-process.php"*
16. **$('.inactive_client').on('click', function(){})**
   This code sets up an event handler to manage the process of marking a client account as inactive:
   A. When an element with the class inactive_client is clicked, it retrieves the client ID.
   B. It displays a confirmation dialog to the user.
   C. If confirmed, it shows a loading overlay and sends an AJAX request to *'includes/main-process.php'* to mark the client account as inactive on the server.
   D. It handles the server response to update the UI and provide feedback to the user, either confirming the action or displaying an error message if something goes wrong.
17. **$('.active_client').on('click', function(){})**
   This code sets up an event handler to manage the process of marking a client account as active:
   A. When an element with the class active_client is clicked, it retrieves the client ID.
   B. It displays a confirmation dialog to the user.
   C. If confirmed, it shows a loading overlay and sends an AJAX request to mark the client account as active on the server.

D. It handles the server response to update the UI and provide feedback to the user, either confirming the action or displaying an error message if something goes wrong.

Manage Client > View

Template Name: view-client.php

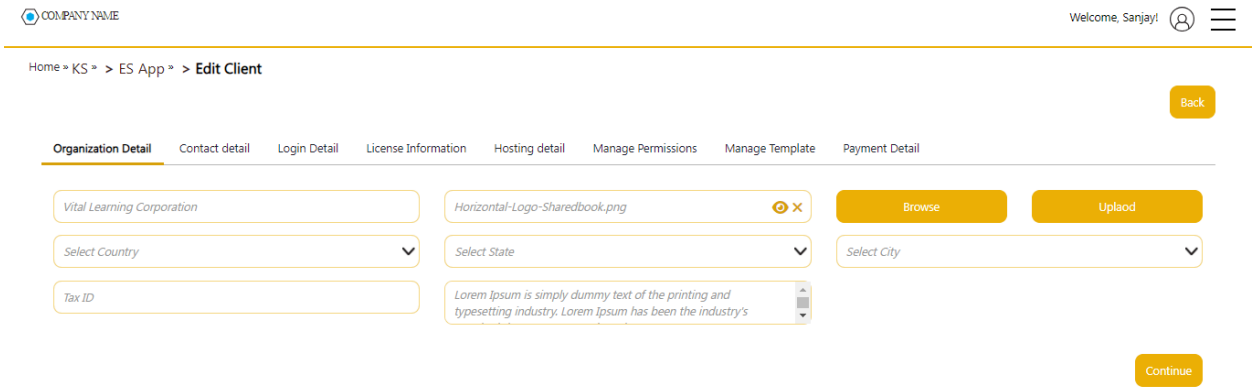


PHP Functions Used

1. **checkAccountType()**
   Global function from "db.class.php". The checkAccountType function determines the type of account based on the user's role stored in the session. It returns a corresponding numerical value for each role or FALSE if the role is not recognized.
2. **getClientData($columns, $tbl_name, $execond)**
   Global function from "db.class.php". This function provides a flexible way to retrieve data from a specified table with optional conditions. It encapsulates the process of preparing and executing the SQL query, as well as handling the result set.

Manage Client > Edit

Template Name: edit-client.php



PHP Functions Used

1. **checkLogin()**
   Global function from "db.class.php". The checkLogin function is designed to verify that a user is logged in by checking the presence of certain session variables. If any of these session variables are missing, the function redirects the user to the base URL.
2. **checkAccountType()**
   Global function from "db.class.php". The checkAccountType function determines the type of account based on the user's role stored in the session. It returns a corresponding numerical value for each role or FALSE if the role is not recognized.
3. **getClientData($columns, $tbl_name, $execond)**
   Global function from "db.class.php". This function provides a flexible way to retrieve data from a specified table with optional conditions. It encapsulates the process of preparing and executing the SQL query, as well as handling the result set.

Third Party JS Library

1. InputMask
2. Bootstrap-DateTimePicker

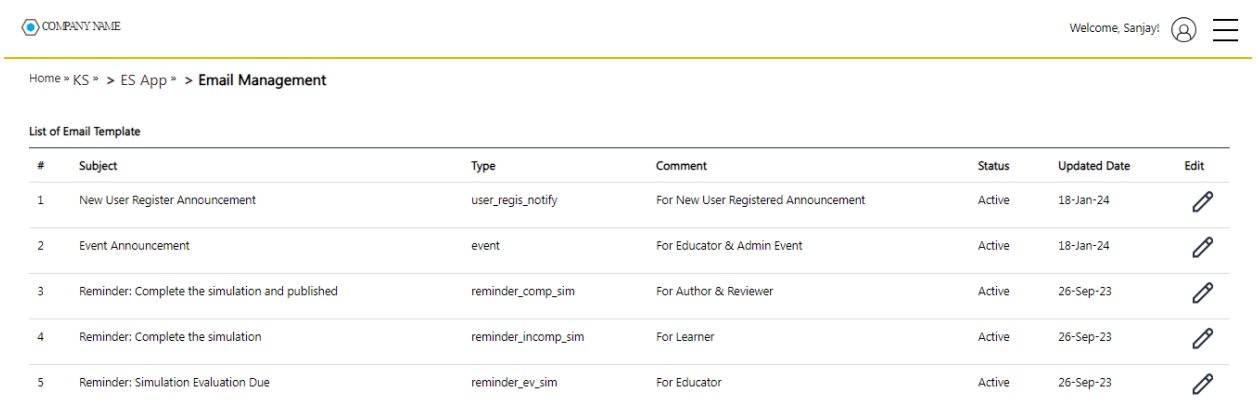
JS Functions Used

1. **$('input[name=mail_type]').on('click', function() {})**
   The jQuery code snippet sets up an event handler for click events on input elements with the name mail_type. Depending on the value of the clicked input, it toggles the visibility of certain elements and clears the values of specific input fields.
2. **$('.checked_all').on('change', function() {})**
   The jQuery code snippet sets up an event handler that synchronizes the checked state of a group of checkboxes based on the checked state of a master checkbox.
3. **$('.tmp_type').change(function() {})**
   The jQuery code sets up an event handler that monitors changes to a group of checkboxes with the class tmp_type. Depending on the state of these checkboxes, it updates the state of a master checkbox with the class checked_all.
4. **$.ajax({...url: "includes/currencies.json",...})**
   This jQuery code snippet makes an AJAX request to fetch data from a JSON file (*includes/currencies.json*) containing currency information. Upon successful retrieval, it dynamically populates a <select> dropdown with the currency options.
5. **$('.form_datetime').datetimepicker({})**
   This jQuery code snippet initializes a datetime picker widget on elements with the class form_datetime using the datetimepicker plugin.
6. **$('.multiselect-ui').multiselect({})**
   This code sets up a multiselect dropdown menu for elements with the class multiselect-ui. Overall, these options configure the behavior and appearance of the multiselect dropdown, making it user-friendly by allowing selection of multiple options with search and select functionalities.

7. **$("#domain_name").on('change', function(){})**
   This code ensures that the value entered into the domain_name input field meets specific criteria (alphanumeric characters, hyphens, underscores, and length requirements) and provides feedback to the user if the input is invalid.
8. **$('#upload').click(function() {})**
   This code handles the process of uploading a client logo asynchronously, providing visual feedback to the user and handling various scenarios such as success, failure, and errors.
9. **$('.delete_logo').click(function() {})**
   This code provides a confirmation dialog before deleting a logo file and handles the deletion process through AJAX request to *"includes/delete-file.php"* asynchronously, updating the UI and providing feedback to the user based on the server's response.
10. **jQuery('body').on('change', '.country', function() {})**
    This code dynamically loads states based on the selected country by fetching data through AJAX request to *'includes/main-process.php*?type=getState&countryId='+cid, providing a responsive and efficient user experience.
11. **jQuery('body').on('change', '.state', function() {})**
    This code dynamically loads cities based on the selected state by fetching data through AJAX request to *'includes/main-process.php*?type=getCity&stateId='+sid, providing a responsive and efficient user experience.
12. **$(".phonecode").each(function(index) {})**
    This code dynamically populates select elements with phone codes based on the retrieved country data through AJAX request to *'includes/main-process.php*?type=getCountry', providing a seamless user experience.

Manage Template

Template Name: email-management.php



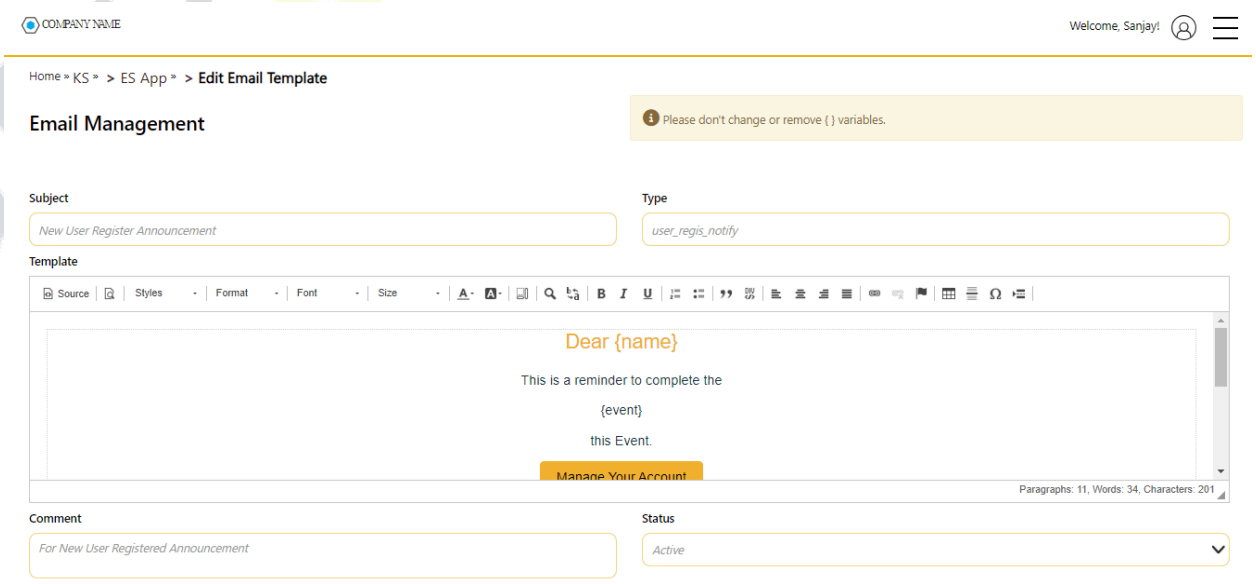
#	Subject	Type	Comment	Status	Updated Date	Edit
1	New User Register Announcement	user_regis_notify	For New User Registered Announcement	Active	18-Jan-24	
2	Event Announcement	event	For Educator & Admin Event	Active	18-Jan-24	
3	Reminder: Complete the simulation and published	reminder_comp_sim	For Author & Reviewer	Active	26-Sep-23	
4	Reminder: Complete the simulation	reminder_incomp_sim	For Learner	Active	26-Sep-23	
5	Reminder: Simulation Evaluation Due	reminder_ev_sim	For Educator	Active	26-Sep-23	

PHP Functions Used

1. **checkLogin()**
   Global function from "db.class.php". The checkLogin function ensures that a user is properly logged in by verifying that all essential session variables are set and not empty.
2. **SQL Queries**
   $sqlr = "SELECT eid, subject, type, comment, status, DATE_FORMAT(edate, '%d-%b-%y') AS date FROM email_template_tbl ORDER BY eid DESC";
   The query retrieves the columns eid, subject, type, comment, status, and a formatted version of the edate column (as date) from the email_template_tbl table. The results are ordered by the eid column in descending order, ensuring that the latest entries appear first.

Manage Template > Edit

Template Name: edit-email-template.php

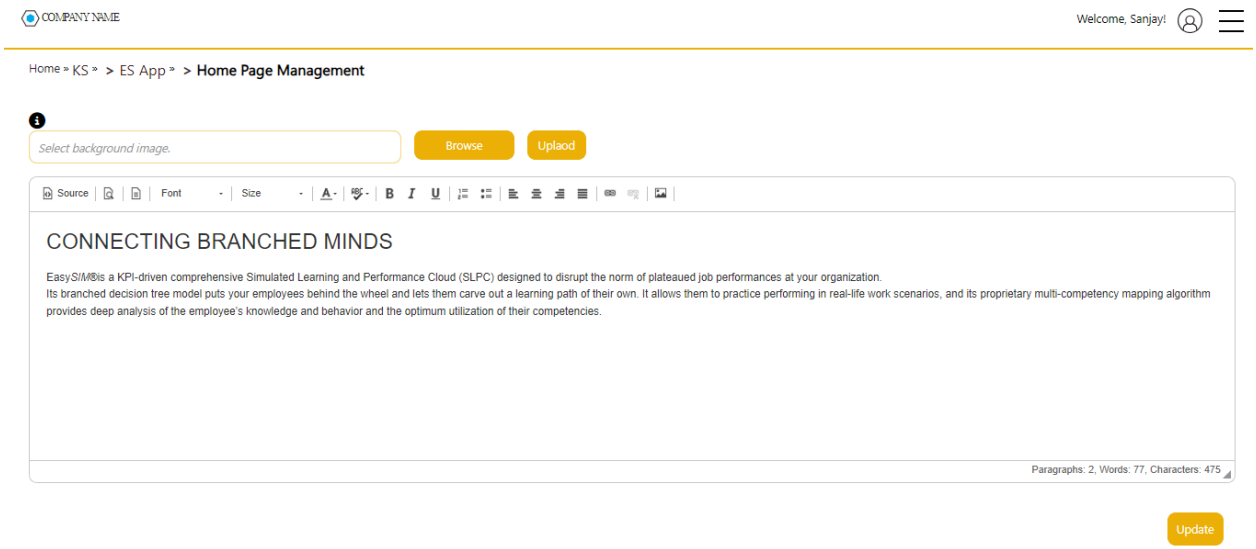


PHP Functions Used

1. **checkLogin()**
   Global function from "db.class.php". The checkLogin function ensures that a user is properly logged in by verifying that all essential session variables are set and not empty.
2. **getEmailTemplate($id)**
   Global function from "db.class.php". The getEmailTemplate function dynamically builds and executes an SQL query to fetch email template data based on the provided $id or $type. If $id is given, it looks for a record with the MD5 hash of eid matching $id. If $type is given, it looks for records with status = 1 and type matching $type. The function returns the fetched data if any records are found, or an empty array otherwise.

Manage Homepage

Template Name: home-page-management.php



PHP Functions Used

1. **checkLogin()**
   Global function from "db.class.php". The checkLogin function ensures that a user is properly logged in by verifying that all essential session variables are set and not empty.

JS Functions Used

1. **$('#upload').click(function() {})**
   This JavaScript code snippet uses jQuery to handle the process of uploading a file via an HTML form. This binds a click event to the HTML element with the ID upload. On click FormData object is created, and the selected file and a boolean flag page_bg_img are appended to it. This object will be used to send the data via AJAX to *"includes/upload.php"*. This code handles file uploading by showing feedback to the user at each step, managing the uploaded file's display, and handling errors gracefully.
2. **$('.delete_bg_file').click(function() {})**
   This code manages the deletion of a file by confirming the user's intent, handling the server through AJAX request to *"includes/delete-file.php"*, and providing feedback based on the server's response.

View Report

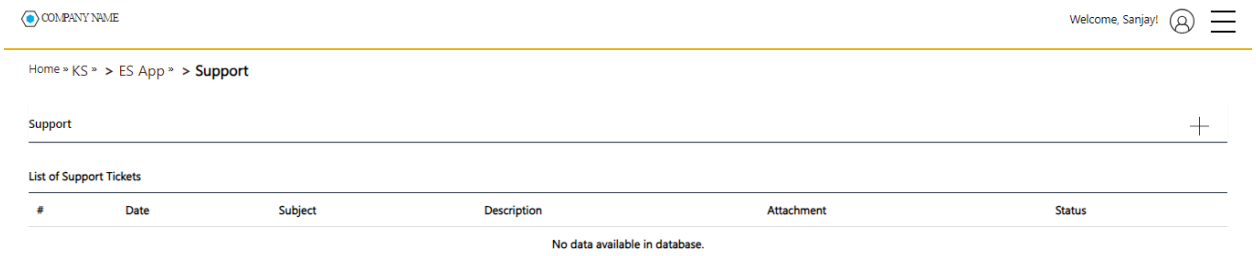
No URL Found/ Not for SuperAdmin

Rule Management

No URL Found/ Not for SuperAdmin

Support

Template Name: support.php



PHP Functions Used

1. **checkLogin()**
   Global function from "db.class.php". The checkLogin function ensures that a user is properly logged in by verifying that all essential session variables are set and not empty.
2. **SQL Queries**
   $sqlr = "SELECT s., u.username, u.role FROM support_tbl AS s LEFT JOIN users AS u ON s.user_id = u.id " . (($role != 'admin') ? " WHERE s.user_id = '" . $userid . "'" : " ") . " ORDER BY s.support_id DESC";*
   This SQL query selects data from a support table (support_tbl) and join it with a users table (users).

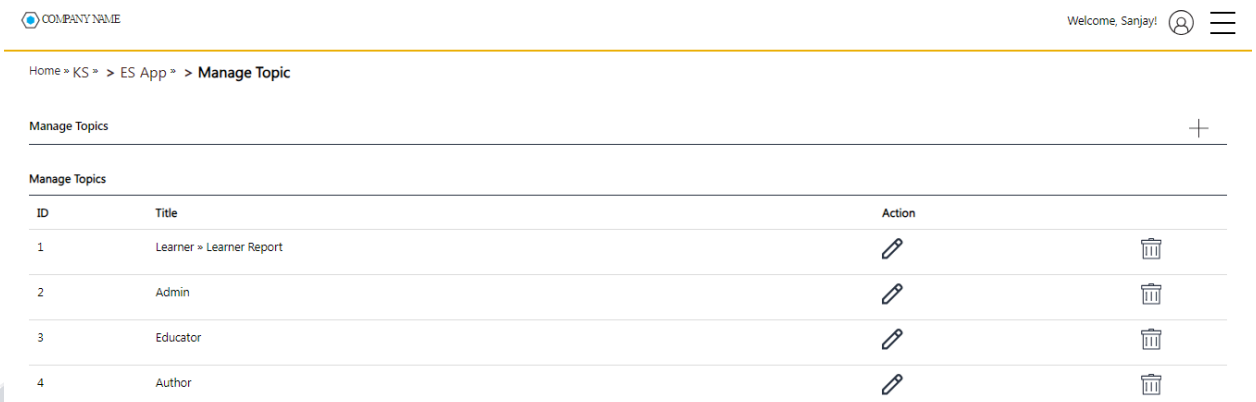
JS Functions Used

1. **$('#upload').click(function() {})**
   The function is triggered when the element with ID upload is clicked and shows a loading overlay to indicate ongoing process. Retrieves file data from the input element and creates a FormData object to hold the file and additional data. Than sends the data to the server via AJAX to *"includes/support-process.php"*. Finally it parses the response, updates the UI to show the uploaded file or an error message.
2. **$('.delete_media_file').click(function() {})**
   This JavaScript code snippet, using jQuery, manages the deletion of a media file. It includes confirmation prompts, server interaction via AJAX to *"includes/delete-file.php"*, and user feedback.
3. **$('.edit').click(function() {})**
   This JavaScript code snippet uses jQuery to manage the process of editing a support item. When an element with the class edit is clicked, it triggers an event that loads *'edit-support-modal.php'* into a modal window with the edit form.

4. **$("#support_form").on('submit', (function(e) {})**
   This JavaScript code snippet handles the submission of a form with the ID support_form using AJAX. It prevents the default form submission behavior, shows a loading overlay during the AJAX request, sends form data to a server-side script (*'includes/support-process.php'*), and handles the server's response.

Manage API/Topic

Template Name: manage-topic.php



PHP Functions Used

1. **checkLogin()**
   Global function from "db.class.php". The checkLogin function ensures that a user is properly logged in by verifying that all essential session variables are set and not empty.
2. **SQL Queries**
   $sqlr = "SELECT t1.id,t1.title,t1.parent_id,if(t1.parent_id > 0, (select CONCAT(t2.title, ' » ') from topics t2 where t2.id=t1.parent_id), '') as ptitle FROM topics t1";
   This SQL query retrieves data from a table named topics to construct a hierarchical structure where each topic may have a parent topic.

   $sqlItems = "SELECT i.,t.title as t_title, if(t.parent_id > 0, (select CONCAT(t2.title, ' » ') from topics t2 where t2.id=t.parent_id),'') as pt_title FROM topics t, topic_items i where t.id = i.topic_id ". $search ;*
   This SQL query retrieves data from two tables, topics and topic_items, joining them based on a common column, id in topics and topic_id in topic_items.

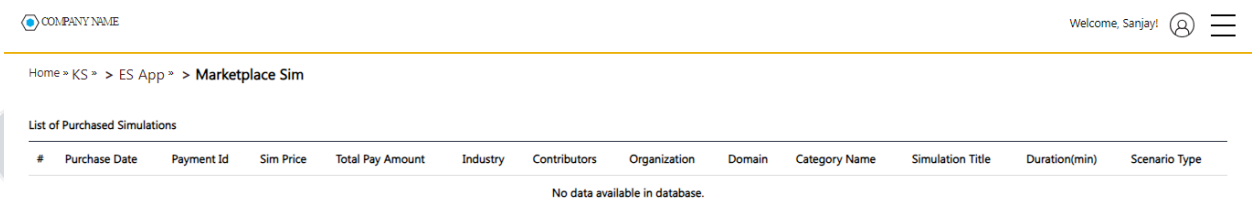
JS Functions Used

1. **$.getJSON('includes/main-process.php?type=getTopics', function (data) {})**
   This JavaScript code fetches JSON data from a server-side script at *'includes/main-process.php?type=getTopics'*, specifically requesting data of type getTopics.

2. **$('.edit_item').click(function () {})**
   The code handles the click event on elements with the class edit_item. It retrieves the ID associated with the clicked element. If the ID is not empty, it displays a loading overlay and loads *'edit-topic-item-modal.php'* as modal content via AJAX. After loading, it hides the loading overlay and either shows the modal with the content or displays an error message.
3. **$('.edit_topic').click(function () {})**
   The code handles the click event on elements with the class edit_topic. It retrieves the ID associated with the clicked element. If the ID is not empty, it displays a loading overlay and loads *'edit-topic-modal.php'* as modal content via AJAX. After loading, it hides the loading overlay and either shows the modal with the content or displays an error message.

Marketplace Simulation

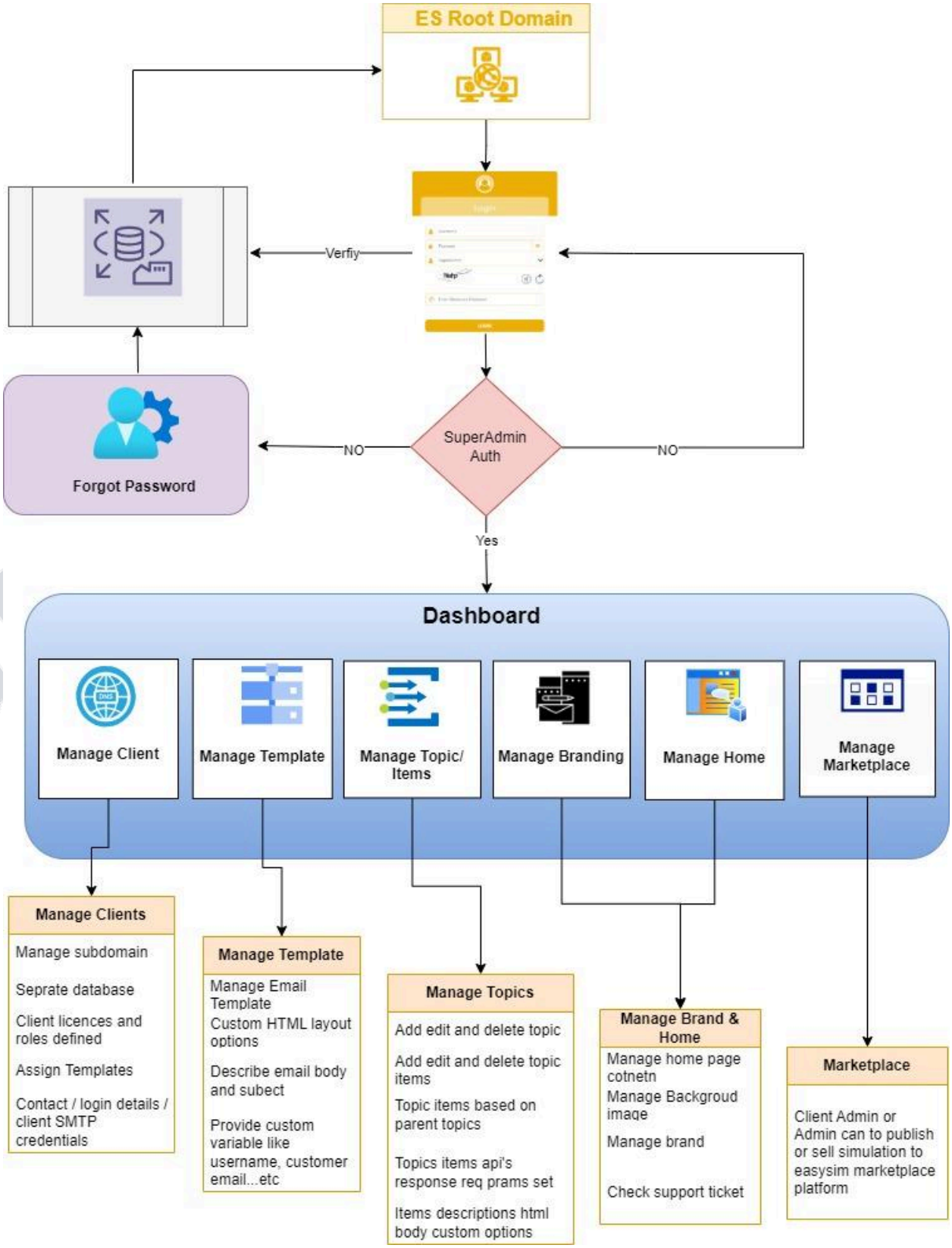
Template Name: marketplace-sim.php



PHP Functions Used

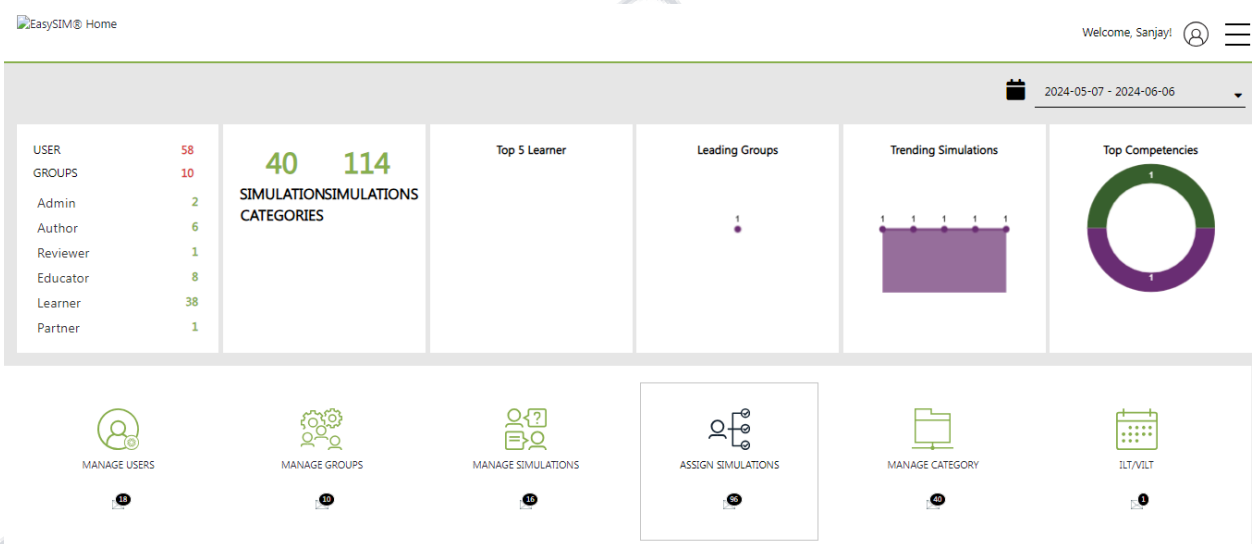
1. **checkLogin()**
   Global function from "db.class.php". The checkLogin function ensures that a user is properly logged in by verifying that all essential session variables are set and not empty.
2. **SQL Queries**
   $sql = "SELECT p., DATE_FORMAT(p.created_at, '%d-%b-%y %h:%i %p') AS pdate, l.industry, l.contributors, c.domain_name, c.db_name, c.organization FROM library_sim_payments AS p LEFT JOIN library_sim AS l ON p.sim_id = l.sim_id LEFT JOIN client_tbl AS c ON p.client_id = c.client_id WHERE 1=1 ORDER BY p.pay_id DESC";*
   This SQL query retrieves data related to library payments from the library_sim_payments table, joining it with additional tables (library_sim and client_tbl) to gather more information about the simulations and clients involved. The result set includes various columns from these tables along with some derived columns for better presentation. The data is ordered by payment ID in descending order, to display the most recent payments first.

Super Admin DFD


ES Root Domain
Verfiy
Forgot Password
NO
SuperAdmin Auth
NO
Yes
Dashboard
Manage Client
Manage Template
Manage Topic/ Items
Manage Branding
Manage Home
Manage Marketplace
Manage Clients
Manage subdomain
Seprate database
Client licences and roles defined
Assign Templates
Contact / login details / client SMTP credentials
Manage Template
Manage Email Template
Custom HTML layout options
Describe email body and subect
Provide custom variable like username, customer email...etc
Manage Topics
Add edit and delete topic
Add edit and delete topic items
Topic items based on parent topics
Topics items api's response req prams set
Items descriptions html body custom options
Manage Brand & Home
Manage home page cotnetn
Manage Backgroud image
Manage brand
Check support ticket
Marketplace
Client Admin or Admin can to publish or sell simulation to easysim marketplace platform

Admin

Overview



This domain will only be accessible by users with **Admin** credentials and will allow management of User, Group, Simulations, Category, Branding, Permission, Competency, Library, Data, My Training Assets, Data-Driven Insight, View Report, ILT/VILT, and Assign Simulation. After authentication, users will be redirected to the dashboard page and will be able to manage. Below is the components managed by users

- Admin Landing page:- Top Grid boxes having Statics Grid box data.:
    - 1. Role **Total Counted**
    - 2. Categories **counted** & Simulation **counted**
    - 3. Top 5 Learner detail
    - 4. Leading Groups
    - 5. Trending Events(simulation)
    - 6. Top Competencies name
- Manage Users:-
    - **Import Bulk User:** import users in Bulk.
    - **Add User:** add user individually
    - **List of Users:** After adding users to this list, The admin can edit users and delete them.
- Manage Group:-
    - **Create Group:** Enter the group name, then select the educator and partner of this group, and then click the "Add Group" button.
    - **Add Users to Group:** created group show in this list after that:

    - Click on the "**Add User**" icon and then add a learner to this group.
    - The admin **edit** the icon of this group and change the educator or partner if the admin wants to change it.
    - Click on the "**User Mail**" icon to send email to users.
    - Click on the "**Assigned Mail"** icon to send an email to learners.
    - Click on the "**Delete**" icon to delete this group.
- Manage Events
    - **Create Simulation:**
    - 1. Create new category for simulation
    - 2. Created category shown in **select category** box, The admin can **select a category** after that enter **simulation title** and then select **scenario type** and then select **author** and **reviewer** who you want to assign the simulation.
    - **Simulation List:**
    - Created simulation shows on this list the admin can **Edit Detail, Edit Simulation, Preview Simulation, Delete Simulation, Duplicate Simulation etc.**
- Assign Simulation
    - **Assign Simulation:**
      Select category , select simulation and enter passing score
      select group, select individually learner list and give start date - end date and give sim attempt number after that click on assign simulation btn(+ icon add more learner group assign simulation)

    - **2. Edit Assigned Simulation:**
      select category and select simulation show all assign simulation the admin can edit date attempt of number after that click on "edit assign simulation" button save data

    - **Export Assigned Simulation & Send Assigned Email:**
      select category and select simulation show all assign simulation in table after that the admin can export all simulation and we can assign email it.

    - **Reset Simulation Attempt Data:**
      The admin can reset data of selected simulation click on reset data button

    - **Assign Learning Path:**
      Select path and give it passing score after that choose start date, end date, and give it attempts number click "Assign Learning Path" btn. Its work same as assign simulation tab

    - **Edit Learning Path Assignment:**
      It works the same as the "Edit Assigned Simulation" tab.

    - **View Assigned Learning Paths:**

You can view Assigned Learning Paths and send the email to Assigned Mail by just clicking on the Assigned Mail button.

- **Export Learning Path & Send Assigned Email:**
  Its work same as "Export Assigned Simulation & Send Assigned Email" tab

- **Simulation List:**
  Here showing all Simulation lists.

- Manage Category
  - Create New Category:
    Enter category name and then click on "add category" button.
  - List of Category
  - Created categories show on this list the admin can edit or delete this category.

- ILT/vILT
  - Create Events:
    When click on date then show popup where the admin Select Category , event name , class venue, event URL, Geographical Location and select group list after that click on event invite button
- Manage Branding
  - **Branding**:
    Primary Colors: The admin uses two colors. 1 normal and 2 hover colors
    Secondary Colors: This color is used for graph
    Font: this use for Font Type, Font Color and Font Size

  - **Email Templates**: The admin can edit the "Email Templates"

  - **Manage Home Page**: The admin can manage the "Home page"

  - **Manage Terms & Conditions**: The admin can manage the "Manage Terms & Conditions"
- Manage Permissions
  - **Role Permission**: The admin can rename the role and update
  - **Chat Permission**: The admin has two options: chat enable or disable. If admin can enable the chat option
  - **Words / Headings - Learner:** The admin can handle all heading changes dynamically for Learner.
  - **Words / Headings - Author**: The admin can handle all heading changes dynamically for Author.
  - **Words / Headings - Admin**: The admin can handle all heading changes dynamically for Admin.

    - **Words / Headings - Educator:** The admin can handle all heading changes dynamically for Educator.
- Manage Competency
    - **Import Competency Framework:** The admin can import competency csv files.
    - **List of Competencies:** The admin can delete competency by clicking on the delete icon.
- Manage Library
    - This page manage for Simulation on Marketplace page
- View Report

    All report data show
    - **System Workflow Report**
        - Overview Report: Overview report has all simulation details like how many simulations pending, total simulation, new Simulation count, Assigned learner, Ready to publish simulation, publish simulation, how many simulations under reviewed an review and review fixed
        - **Assignment Report**

    - **Competency Mapped Event**
        - Group Performance: The admin can download the report Group Performance based.
        - Systemwide Analytics:
        - Consolidated Systemwide Analytics
        - Competency Analytics Report
        - Comparative Competence Analytics

    - **E-Learning Reports**
        - Scorm E-learning Report: The admin can download the report Scrom E-Learning Type Simulation based.
        - Video Report: The admin can download the report Video Type Simulation based.
        - Pdf Report: The admin can download the report PDF Type Simulation based.
        - Learning Path Report: The admin can download the report Learning Path Type Simulation based.

    - **ILT/vILT Events**
        - ILT/vILT Attendance Report: The admin can download the report ILT/vILT Type event Simulation based.
        - Educator Event Report: The admin can download the report Educator Event report.
- Data Driven Insights
    - All Data Driven Insights data show in this page.
- My Training Assets

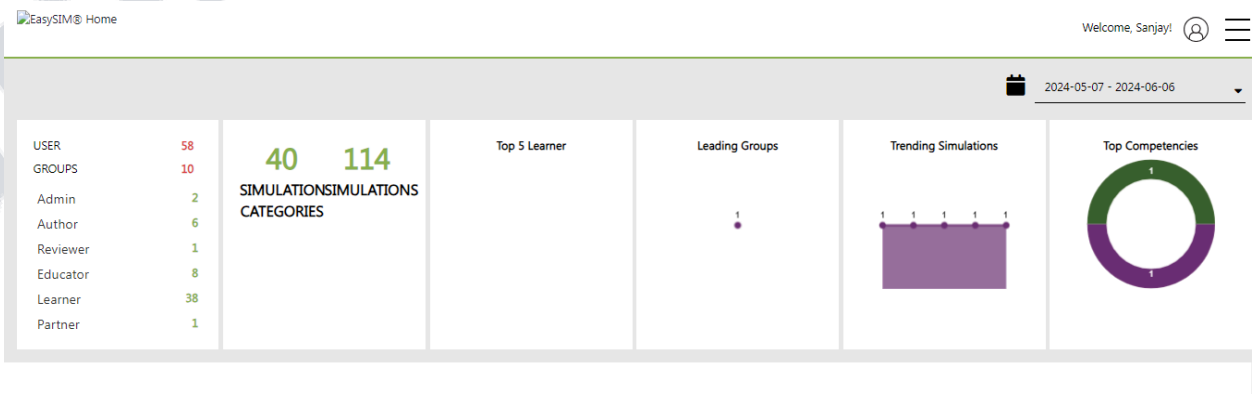
- **Add Assets** in this page as a **Link** or as an Upload choose option. After that added assets show in the list. After that the admin can edit and select educator and partner roles to assign the assets.
- These assets show on the educator role and partner role page.

- Manage Data
  - **Common Data (Users, Group and assign simulation) Bulk Import:** The admin can import Common Data (Users, Group and assign simulation) bulk
  - **Learning Path Data Bulk Import:** The admin can import Learning Path Data bulk
  - **Learning Path Assign Bulk Import**: The admin can import Learning Path Assign Bulk
  - **Learning Path List:** All import Learning Path List show in this table the admin can export all data click on Export LP Data.

Code Stuff

Dashboard

Template Name: admin-dashboard.php



It shows overall data regarding number of users, groups, admins, authors, reviewers, educators, leaners and partners with total numbers of Simulations created and categories. Also displays top 5 learner2, leading groups, trending simulations and top competencies.

PHP Functions Used

1. **getClientPermission()**
   Global function from "db.class.php". The getClientPermission function retrieves permissions for a specific client from a database and returns the results. Here's a detailed explanation of the function's code:
   a. **public function getClientPermission(int $client_id){**
      i. The function getClientPermission takes a single argument $client_id of type integer. This argument represents the unique identifier of the client whose permissions are to be fetched.
   b. **$sql = "SELECT *, (admin + auth + reviewer + learner + educator) AS totalUsers FROM client_permissions**

WHERE client_id=?";

i. The SQL query selects all columns (*) from the client_permissions table.
ii. Additionally, it computes the sum of five columns (admin, auth, reviewer, learner, educator) and labels this sum as totalUsers.
iii. The WHERE client_id=? clause specifies that the query should return only the row where the client_id matches the provided value. The ? is a placeholder for a parameter that will be bound later.

2. **checkAccountType()**
   Global function from "db.class.php". The checkAccountType function determines the type of account based on the user's role stored in the session. It returns a corresponding numerical value for each role or FALSE if the role is not recognized.
3. **getTotalUsers()**
   Global function from "db.class.php". The getTotalUsers function retrieves the total number of users from the user's table, optionally filtered by a specific role.
4. **getTotalGroup()**
   Global function from "db.class.php". The getTotalGroup function retrieves the total number of groups from the group_tbl table.
5. **getTotalCategory()**
   Global function from "db.class.php". The getTotalCategory function retrieves the total number of categories from the category_tbl table.
6. **getTotalScenario()**
   Global function from "db.class.php". The getTotalScenario function retrieves the total number of scenarios from the scenario_master table.
7. **getBaseUrl()**
   Global function from "db.class.php". The getBaseUrl() function in PHP constructs and returns the base URL of the current application.
8. **getTotalNewUsers()**
   Global function from "db.class.php". The getTotalNewUsers function retrieves the total number of users from the user's table who have never logged in (i.e., their last_login field is NULL).
9. **getDraftScenario()**
   Global function from "db.class.php". The getDraftScenario function is designed to fetch scenario IDs from a database based on their status and return the count of such scenarios.
10. **getAssignScenario()**
    Global function from "db.class.php". The getAssignScenario function is designed to fetch scenario IDs from a database where the status is one of several specified values (5, 8, or 13) and return the count of such scenarios.

Third Party JS Library

1. DateRangePicker
2. CanvasJS

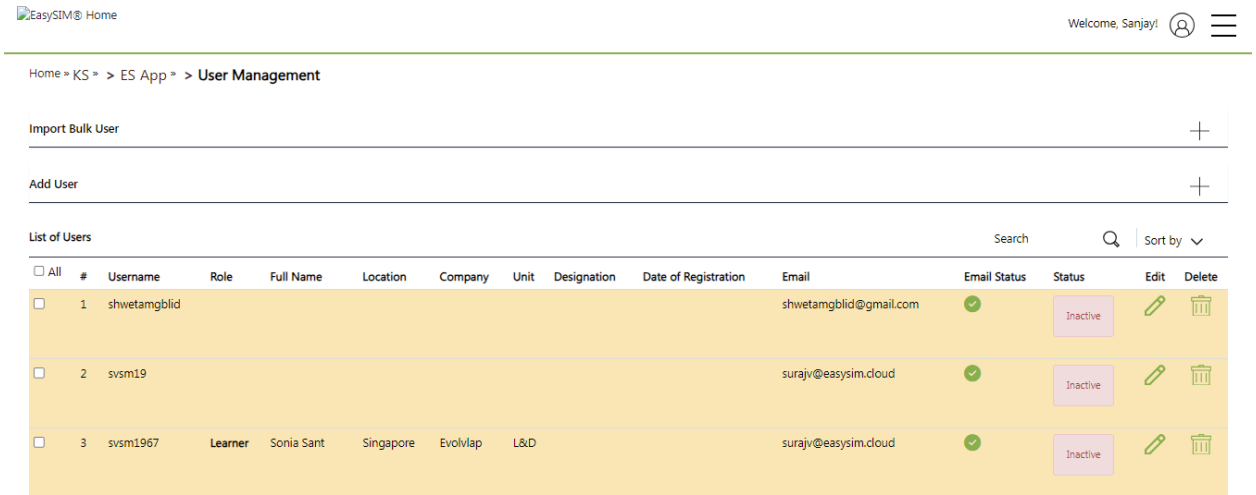
3. OwlCarousel

JS Functions Used

1. **function cb(start, end) {}**
   The function cb(start, end) is designed to update the content of a specific HTML element with the ID reportrange by setting the inner HTML of a span element within it to display a formatted date range.
2. **$('#reportrange').daterangepicker({})**
   The provided code snippet initializes a date range picker on an HTML element with the ID reportrange. It uses the daterangepicker plugin, a popular jQuery-based date range picker.
3. **function defaultGraphData(date){})**
   The defaultGraphData(date) function is designed to fetch and display various types of charts based on data retrieved via AJAX requests to *'includes/client-admin-dash-grid-dataNew.php'*. The charts display different metrics such as top learners, top competencies, trending simulations, and leading groups.
4. **$('#reportrange').on('change', function(){})**
   The provided jQuery code snippet sets up an event listener on an HTML element with the ID reportrange. This listener responds to changes in the selected date range by updating the graphs displayed on the page.

Manage Users

Template Name: user-management.php



PHP Functions Used

1. **checkLogin()**
   Global function from "db.class.php". The checkLogin function ensures that a user is properly logged in by verifying that all essential session variables are set and not empty.
2. **SQL Queries**
   *$sqlr = "SELECT *, REPLACE(CONCAT_WS(' ',fname,lname),' ',' ') AS fullname, DATE_FORMAT(date_of_join, '%e-%b-%y') AS dj FROM users WHERE 1=1 $norderby";*
   The query selects all columns from the users table, along with fullname column created by concatenating fname and lname, with any double spaces removed and dj column that formats the date_of_join date to a more readable format where "WHERE 1=1" clause allows for easy addition of more conditions, and $norderby is likely used to include dynamic ordering or additional conditions to the query.

Third Party JS Library

1. DateTimePicker

JS Functions Used

1. **validateUserName() {}**
   The validateUserName function ensures that the username entered by the user only contains letters and numbers.
2. **const updateStatus = (status, id) => {}**

   The updateStatus function is designed to update the status of a user in a web application, either setting them as 'Active' or 'Inactive'. The function uses SweetAlert for confirmation dialogs and jQuery for making an AJAX request to *'includes/process.php'*, with parameters as { user_update_status: true, status: status, uid: id }.
3. **$('.checked_all').on('change', function() {})**
   This jQuery code snippet handles the functionality of a "select all" checkbox that toggles the checked state of multiple individual checkboxes.
4. **$('.checkbox').change(function() {})**
   The provided jQuery code snippet implements a mechanism to synchronize the state of a "select all" checkbox with that of multiple individual checkboxes.
5. **$('.form_datetime').datetimepicker({})**
   The provided jQuery code initializes a datetime picker plugin for elements with the class form_datetime.

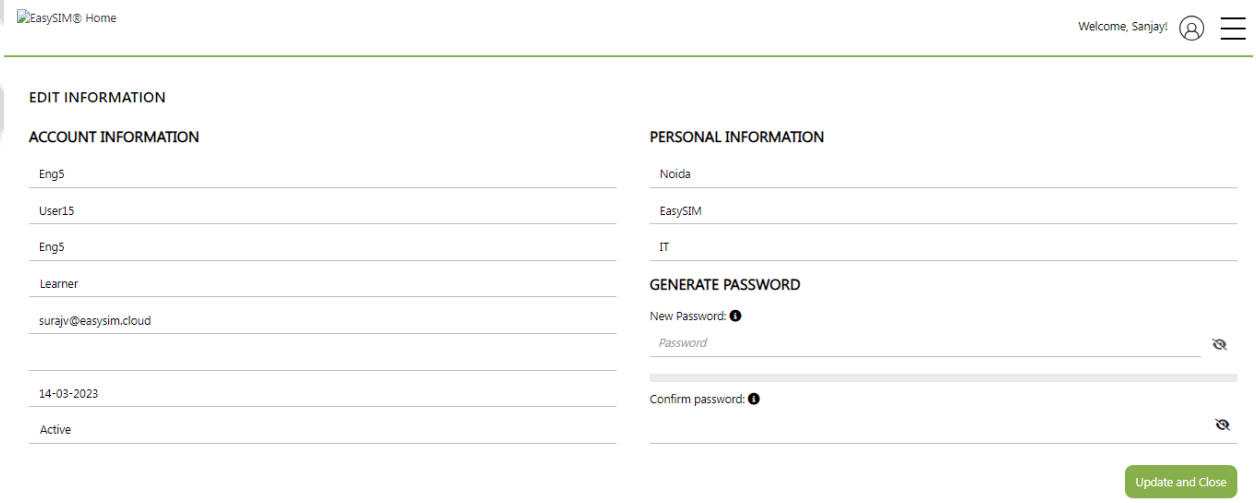
6. **$('#upload').click(function() {})**

The provided jQuery code handles the process of uploading a file via AJAX to *"import-csv-file.php"* and displaying appropriate messages based on the response.

7. **$('.delete_user').click(function() {})**
   This jQuery code handles the deletion of a user by sending an AJAX request to the server at *'includes/process.php'* and displaying a confirmation dialog using SweetAlert before proceeding with the deletion.
8. **$('#delete_Multiple_Users').click(function() {})**
   This jQuery code handles the deletion of multiple users by sending an AJAX request to the server at *'includes/process.php'* and displaying a confirmation dialog using SweetAlert before proceeding with the deletion.

Manage Users > Edit

Template Name: user-management.php



PHP Functions Used

1. **checkLogin()**
   Global function from "db.class.php". The checkLogin function ensures that a user is properly logged in by verifying that all essential session variables are set and not empty.
2. **getUsers($uid)**
   Global function from "db.class.php". This PHP function, getUsers, retrieves user data from the database based on an optional user ID parameter.

JS File Used

File Path: content/js/pwd-script.js

1. **findParentNode = (elem, parentClass) => {}**
   This JavaScript function, findParentNode, is designed to find the nearest ancestor element of a given element (elem) that has a specified class (parentClass).
2. **testPassw = passw => {}**
   This JavaScript function, testPassw, assesses the strength of a given password based on various criteria, such as length and the presence of different character types (uppercase letters, lowercase letters, digits, and special characters).
3. **testPasswError = passw => {}**
   This JavaScript function, testPasswError, checks if a given password contains any error symbols.
4. **setStrengthBarValue = (bar, strength) => {}**
   This JavaScript function, setStrengthBarValue, sets the value of a strength bar element based on the provided strength level.
5. **setStrengthBarStyles = (bar, strengthValue) => {}**
   This JavaScript function, setStrengthBarStyles, applies visual styles to a strength bar element based on the provided strength value.
6. **setStrengthBar = (bar, strength) => {}**
   This JavaScript function, setStrengthBar, combines two separate functionalities related to managing the visual representation of a strength bar element for password strength assessment.
7. **unblockSubmitBtn = (btn, strength) => {}**
   This JavaScript function, unblockSubmitBtn, manages the state of a submit button based on the strength level of a password.
8. **findErrorMsg = input => {}**
   This JavaScript function, findErrorMsg, is designed to locate and return the error message element associated with a given input field.
9. **showErrorMsg = input => {}**
   This JavaScript function, showErrorMsg, is responsible for displaying an error message associated with a given input field.
10. **hideErrorMsg = input => {}**
    This JavaScript function, hideErrorMsg, is responsible for hiding an error message associated with a given input field.
11. **passwordStrength = (input, strengthBar, btn) => {}**
    This JavaScript function, passwordStrength, is responsible for evaluating the strength of a password entered into an input field and updating the corresponding strength bar and submit button accordingly.
12. **passwordVisible = passwField => {}**
    This function named passwordVisible, toggles the visibility of a password field in an HTML form.
13. **changeVisibiltyBtnIcon = (btn, status) => {}**
    This function named changeVisibilityBtnIcon is responsible for changing the icon displayed on a button that toggles the visibility of a password field.

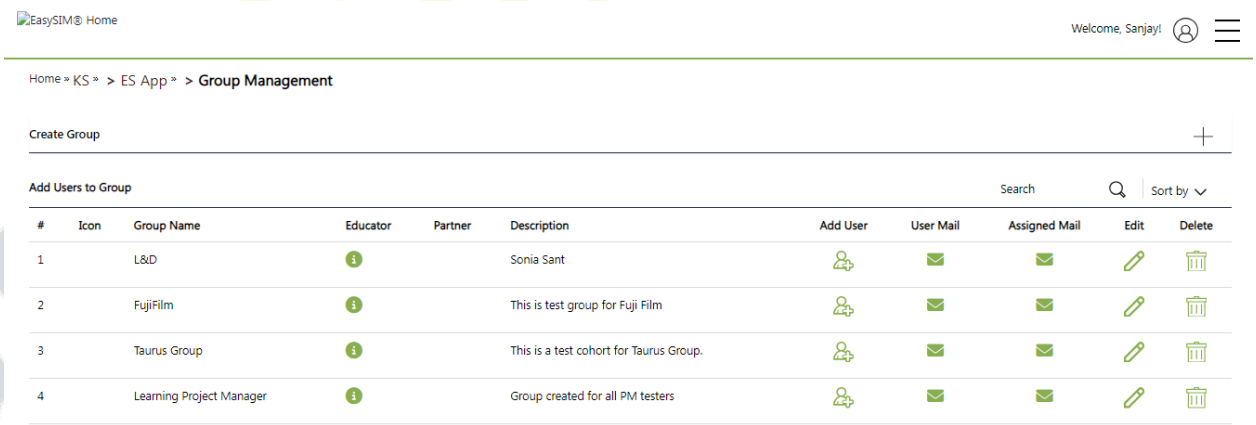
14. **passwVisibilitySwitcher = (passwField, visibilityToggler) => {}**
    This function named passwVisibilitySwitcher, is used to toggle the visibility of a password field and update the icon on a visibility toggler button accordingly.

JS Functions Used

1. **isNumberKey(evt) {}**
   This JavaScript function isNumberKey is typically used in web development to restrict user input in a text field to only numeric values.

Manage Groups

Template Name: group-management.php



PHP Functions Used

1. **checkLogin()**
   Global function from "db.class.php". The checkLogin function ensures that a user is properly logged in by verifying that all essential session variables are set and not empty.
2. **getRole()**
   Global function from "db.class.php". This PHP function getRole() is designed to retrieve roles from a database table named "role" and return them as an associative array.
3. **SQL Queries**
   $sql = "DELETE t1., t2.* FROM group_tbl t1 LEFT JOIN assignment_tbl t2 ON t1.group_id = t2.group_id WHERE t1.group_id = ?";*
   This SQL statement is a DELETE statement that removes data from two tables, group_tbl and assignment_tbl, based on a condition specified in a JOIN operation.

   *$sqlr = "SELECT * FROM group_tbl WHERE 1=1 $norderby";*
   This SQL query selects all columns from the group_tbl table without any specific filtering conditions (1=1 being a placeholder), and any additional conditions or ordering specified by the $norderby variable are appended to the query string dynamically.

Third Party JS Library

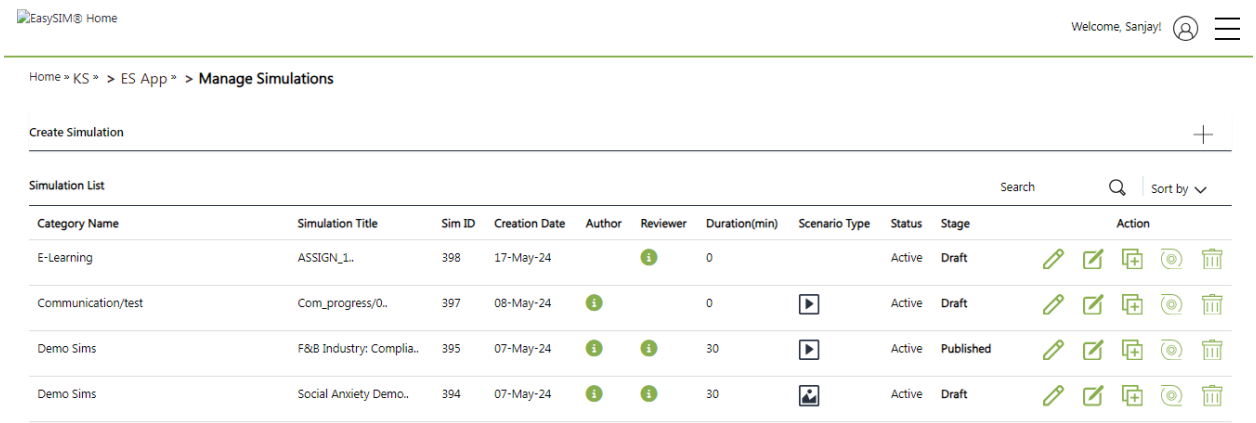
1. TooltipJS

JS Functions Used

1. **$('.selection').multiselect({})**
   This code snippet is using the jQuery plugin "multiselect" to enhance the functionality of a dropdown menu with the class "selection".
2. **$('.edit').click(function () {})**
   This code snippet loads *'edit-group-modal.php'* with parameters {'data-id':id} content into a modal dialog box when an element with the class "edit" is clicked. It shows a loading overlay during the loading process and displays an error message if the loading fails.
3. **$('.add_users').click(function () {})**
   This code snippet loads *'add-user-group-modal.php'* with parameters {'data-id':id} content into a modal dialog box when an element with the class "add_users" is clicked. It shows a loading overlay during the loading process and displays an error message if the loading fails.
4. **$('.user_mail').click(function() {})**
   This code snippet displays a confirmation dialog when an element with the class "user_mail" is clicked. If the user confirms the action, it sends a request to the server at *'includes/process.php'* via AJAX to send a new account mail to a group of users. It then displays a success or error message based on the server response.
5. **$.post(*'includes/ajax.php'*, { get_users: true, role: "'educator','partner'" })**
   This code sends a request to fetch users based on specific roles, updates multiselect dropdowns with the returned data, and handles any errors by displaying an appropriate message.
6. **$('.assigned_mail').click(function() {})**
   This code snippet shows a confirmation dialog when an element with the class "assigned_mail" is clicked. If the user confirms, it sends a request to the server at *'includes/process.php'* to send an assigned event mail, and then displays a success or error message based on the server's response.

Manage Simulations

Template Name: manage-simulations.php



PHP Functions Used

1. **checkLogin()**
   Global function from "db.class.php". The checkLogin function ensures that a user is properly logged in by verifying that all essential session variables are set and not empty.
2. **getRole()**
   Global function from "db.class.php". This PHP function getRole() is designed to retrieve roles from a database table named "role" and return them as an associative array.
3. **SQL Queries**
   $sqlr = "SELECT s.scenario_id, s.Scenario_title, LEFT(s.Scenario_title, 25) AS stitle, s.sim_ques_type, s.sim_temp_type, DATE_FORMAT(s.created_on, '%d-%b-%y') AS cdate, s.assign_author, s.assign_reviewer, s.duration, s.status, s.vtype, s.web_object, s.web_object_file, c.category, st.status_name, t.type_name, t.type_icon FROM scenario_master AS s LEFT JOIN category_tbl AS c ON s.scenario_category = c.cat_id LEFT JOIN scenario_status AS st ON s.status = st.sid LEFT JOIN scenario_type AS t ON s.author_scenario_type = t.st_id WHERE 1=1 $orderby";
   This query retrieves various details about scenarios from the scenario_master table, including titles, types, statuses, and associated category information. The LEFT JOIN statements ensure that even if there are no matching records in the category_tbl, scenario_status, or scenario_type tables, the scenarios will still be included in the results. The query is dynamically constructed to potentially include additional ordering or filtering specified by the $orderby variable.

Third Party JS Library

1. TooltipJS
2. Bootstrap-DateTimePicker

JS Functions Used

1. **$('.multiselect-ui').multiselect({})**
   This code snippet initializes a multi-select dropdown using the jQuery plugin called multiselect.
2. **$('#category').autocomplete({})**
   This code snippet initializes an autocomplete feature for an input field with the ID category using jQuery UI's autocomplete widget.
3. **isNumberKey(evt) {}**
   This JavaScript function isNumberKey is typically used in web development to restrict user input in a text field to only numeric values.
4. **$('.edit').click(function () {})**
   This code snippet uses jQuery to handle the click event for elements with the class edit. When one of these elements is clicked, it triggers a series of actions including showing a loading overlay, loading *'edit-scenario-modal.php'* with parameters {'data-id': id} content into a modal, and displaying the modal.
5. **$('.clone_sim').click(function () {})**
   This code snippet handles the cloning of a simulation when an element with the class clone_sim is clicked. It involves showing a confirmation dialog, sending an AJAX request to *"includes/process.php"*, and displaying success or error messages.
6. **$('.delete_sim').click(function() {})**
   This code snippet handles the deletion of a simulation when an element with the class delete_sim is clicked. It involves showing a confirmation dialog, sending an AJAX request to *'includes/process.php'*, and displaying success or error messages.
7. **$('.export_data').click(function() {})**
   This code snippet handles the exporting of data when an element with the class export_data is clicked. It involves opening *"includes/export.php"* in a new window to download the data and displaying a success message.
8. **$.post('includes/ajax.php', {get_users:true, role:"'author','reviewer'"})**
   This code snippet sends an AJAX request to *'includes/ajax.php'* to retrieve user data based on specific roles and populates select elements with the retrieved data using the MultiSelect plugin.
9. **$('.form_datetime').datetimepicker({})**
   This code snippet initializes a datetime picker for elements with the class form_datetime using the Bootstrap DateTimePicker plugin.

JS Files Used

File Path: content/js/scorm.js

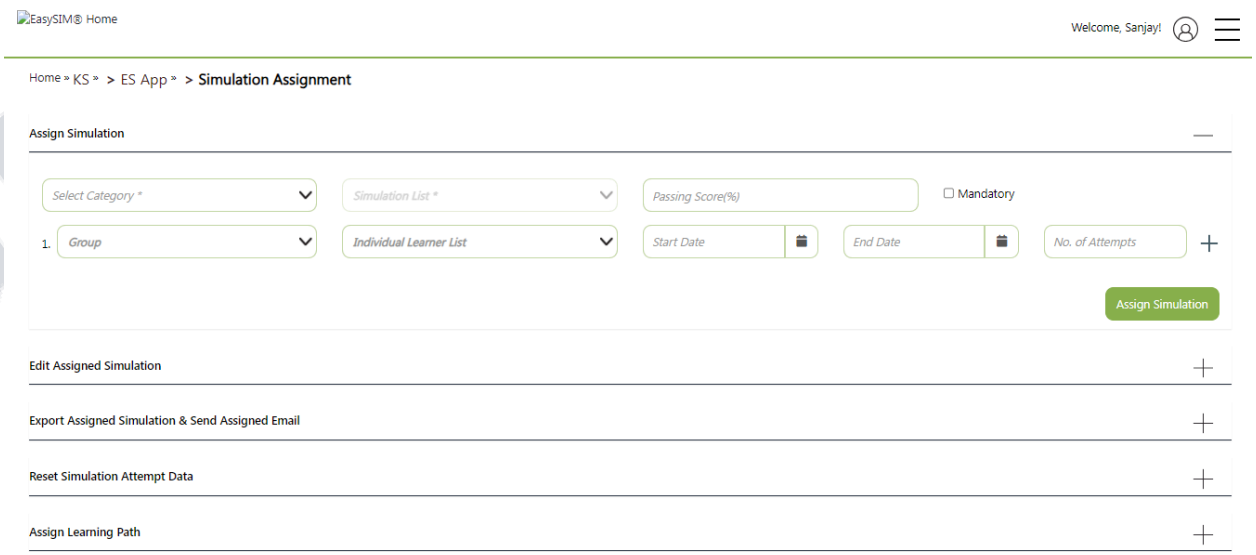
1. **function setupScormApi() {}**
   The "setupScormApi" function is responsible for setting up the SCORM API functionality. It initializes SCORM communication and handles launching SCORM content within a popup window.

2. **function LMSInitialize(initializeInput) {}**
   The "LMSInitialize" function is a part of the SCORM (Sharable Content Object Reference Model) API. It is called to initialize communication between the SCORM content and the SCORM runtime environment (usually a learning management system or LMS).
3. **function LMSGetValue(varname) {}**
   The "LMSGetValue" function is another essential part of the SCORM API. It's called to retrieve the value of a specific data model element (variable) from the SCORM runtime environment (typically an LMS).
4. **function convertTimeString(timeString) {}**
   The "convertTimeString" function is designed to convert a time string into a standardized format of "HH:MM".
5. **function setStatusAndResult() {}**
   The "setStatusAndResult" function is responsible for setting the status and result of a SCORM lesson based on various conditions.
6. **function saveOrUpdate() {}**
   The "saveOrUpdate" function is responsible for saving or updating the SCORM attempt data based on certain conditions.
7. **function resetStatusVars() {}**
   The "resetStatusVars" function is used to reset the status-related variables to their initial values.
8. **function LMSSetValue(varname, varvalue) {}**
   The "LMSSetValue" function is responsible for setting values to SCORM data model elements based on the provided varname and varvalue parameters.
9. **function LMSFinish(finishInput) {}**
   The "LMSFinish" function is responsible for finalizing the SCORM session when the learner finishes interacting with the content.
10. **const base_url = (page = '') => {}**
    The "base_url" function is used to generate the base URL of the current page, optionally appending a specific page or path to it.
11. **const getUid = (sid) => {}**
    The "getUid" function sends AJAX request to *'includes/ajax.php'* with parameter user ID (uid) from the server and retrieves data, then creates an attempt entry in the "scenario_attempt_tbl" table with this "uid" by sending AJAX request to 'api-process.php', a given scenario ID (sid), and a user ID (userid). If successful, it sets the attempt ID (aid) and calls another function to handle additional actions.
12. **function getLaunchURL(path) {}**
    The "getLaunchURL" function retrieves the launch URL from an IMS manifest file (*imsmanifest.xml*) located at a given path. This function returns a promise that resolves with the full launch URL.
13. **function getOrgName(path, callback) {}**
    The "getOrgName" function retrieves the organization name and schema version from an IMS manifest file (*imsmanifest.xml*) located at a specified path. This function uses asynchronous operations (await and fetch) to handle the network request and parsing of the XML document.

14. **function msToTime(s) {}**
    The "msToTime" function converts milliseconds to a formatted time string in the format "HH:MM".
15. **function update_attempt_tbl() {}**
    The "update_attempt_tbl" function is responsible for updating the attempt table with relevant information.
16. **function toHoursAndMinutes(totalSeconds) {}**
    The "toHoursAndMinutes" function is used to convert a total number of seconds into hours, minutes, and seconds.
17. **function insert_attempt_tbl() {}**
    The "insert_attempt_tbl" function is responsible for inserting a new record into the attempt table.

Assign Simulations

Template Name: simulation-assignment.php



PHP Functions Used

1. **checkLogin()**
   Global function from “db.class.php”. The checkLogin function ensures that a user is properly logged in by verifying that all essential session variables are set and not empty.
2. **getRole()**
   Global function from “db.class.php”. This PHP function getRole() is designed to retrieve roles from a database table named "role" and return them as an associative array.
3. **SQL Queries**
   $sqlr = "SELECT DISTINCT s.scenario_id, s.Scenario_title, LEFT(s.Scenario_title, 25) AS stitle, DATE_FORMAT(s.created_on, '%d-%b-%y') AS cdate, DATE_FORMAT(s.assign_start_date, '%d-%b-%y') AS sdate,

DATE_FORMAT(s.assign_end_date, '%d-%b-%y') AS edate, s.assign_author, s.assign_reviewer, s.duration, c.category, st.status_name, t.type_name, t.type_icon FROM scenario_master AS s LEFT JOIN category_tbl AS c ON s.scenario_category = c.cat_id LEFT JOIN scenario_status AS st ON s.status = st.sid LEFT JOIN scenario_type AS t ON s.author_scenario_type = t.st_id WHERE s.status = 8 $search ORDER BY s.scenario_id DESC";
This query retrieves a distinct list of scenarios along with related information from joined tables, applies specific formatting to certain fields, filters the results based on status, optionally includes additional search conditions, and orders the results by scenario ID in descending order.

$path_sql = "SELECT path_id, DATE_FORMAT(created_at, '%d-%b-%y') AS pdate, title, sim FROM path_assignment_tbl WHERE 1=1 ORDER BY path_id DESC";
The query retrieves the path_id, a formatted created_at date as pdate, the title, and the sim from the path_assignment_tbl table, with the results ordered by path_id in descending order. The WHERE 1=1 clause allows for easy addition of more filtering conditions if needed.

Third Party JS Library

1. Datatables
2. TooltipJS
3. Bootstrap-DateTimePicker

JS File Used

File Path: content/js/sim-assignment.js

1. **$("#reset_user_id").multiselect({})**
   $("#scenario").multiselect({})
   This code configures a jQuery multiselect dropdown with specific options for the HTML element as per the ID passed in the function.
2. **$('.form_datetime').datetimepicker({})**
   This code initializes a datetime picker widget on elements with the class form_datetime using the datetimepicker plugin. Each option within the function configures specific behaviors and properties of the datetime picker.
3. **$('#category').on('change', function() {})**
   This code snippet sets up an event listener for changes on a select element with the ID category. When the selected option changes, it triggers a series of actions to update another element with the ID scenario based on the selected category.
4. **$('#update_category').on('change', function() {})**
   This code snippet sets up an event listener for changes on a select element with the ID update_category. When the selected option changes, it triggers a series of actions to update other elements based on the selected category.

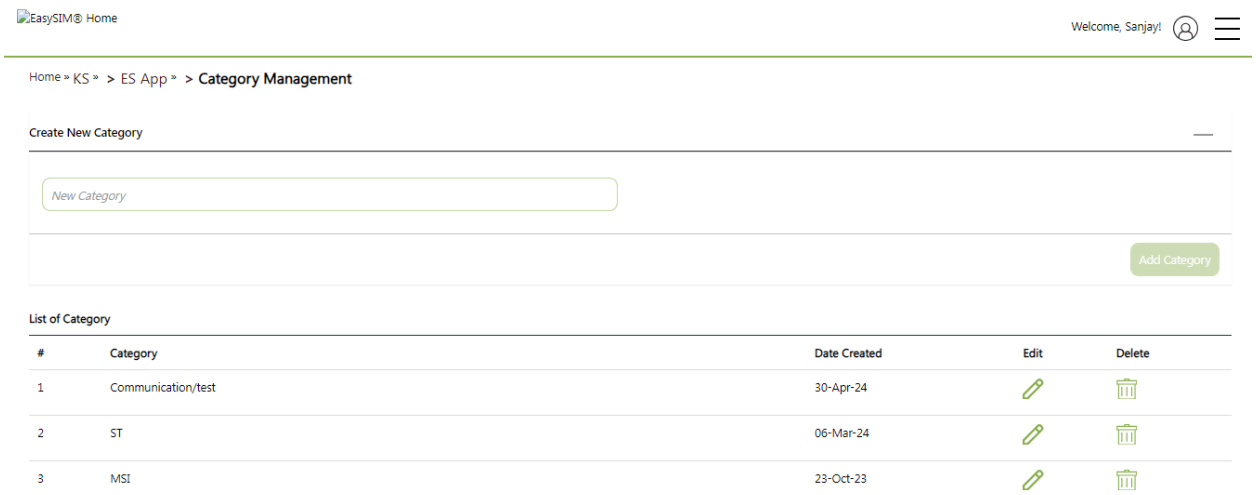
5. **$('#reset_category').on('change', function() {})**
   This code listens for changes in the reset_category dropdown, shows a loading overlay, clears and resets relevant form elements, fetches new scenario options through AJAX request to *'includes/ajax.php'* based on the selected category, hides the loading overlay, updates the reset_sim_id dropdown with the fetched data, or clears the dropdown if no data is found.
6. **$('#reset_sim_id').on('change', function(){})**
   This code listens for changes in the reset_sim_id dropdown, shows a loading overlay, clears and resets the reset_user_id dropdown, fetches new user options based on the selected simulation ID, hides the loading overlay, updates the reset_user_id dropdown with the fetched data from *'includes/ajax.php*?getResetSimLearner=true&sim_id='+ id, or rebuilds and refreshes the dropdown if no data is found.
7. **isNumberKey = (evt) => {}**
   This JavaScript function isNumberKey is typically used in web development to restrict user input in a text field to only numeric values.
8. **groupConfig = (id) => {}**
   editGroupConfig = (id) => {}
   learnerConfig = (id) => {}
   editLearnerConfig = (id) => {}
   learnerConfigPath = (id) => {}
   The function groupConfig is used to initialize and configure a multiselect dropdown for a specific element, identified by a dynamic ID.
9. **$('#update_scenario').on('change', function() {})**
   This code essentially handles the dynamic loading and updating of assignment-related form fields based on the selected scenario, providing a user-friendly interface for managing assignments.
10. **jQuery('body').on('click', '.add_more_multi_assign', function() {})**
    This code dynamically adds a set of form fields when a certain element is clicked, sets up some configurations for group and learner selection, and updates the total assignment count.
11. **$('body').on('click', '.edit_remove', function (e) {})**
    This code handles the removal of a dynamically added row or block of data in response to a click on an element with the class .edit_remove. It displays a confirmation dialog before performing the deletion action.
12. **$('#path_assignment').on('click', function (){})**
    This code snippet captures user input for a learning path title using SweetAlert, processes the input through a POST request to a server-side script at *'includes/process.php'*, and displays appropriate messages based on the response.
13. **$('#edit_path').on('change', function() {})**
    This code snippet dynamically updates a form based on the selected learning path. It fetches assignment data associated with the selected path and populates the form accordingly.
14. **jQuery('body').on('click', '.add_more_multi_assign_path', function() {})**
    This code dynamically adds a set of form fields for a new assignment when a certain

element with the class .add_more_multi_assign_path is clicked. It populates select boxes for group and learner selection and updates the total assignment count accordingly.

15. **$('body').on('click', '.edit_remove_path', function (e) {})**
    This code snippet handles the removal of dynamically added rows or blocks of data in response to a click on an element with the class .edit_remove_path. It displays a confirmation dialog before performing the deletion action and updates the total assignment count accordingly.
16. **transformAssignmentData = (filterData, reportType) => {}**
    This function, transformAssignmentData, takes two parameters: filterData and reportType, and transforms the data stored in filterData based on the reportType.
17. **genAssignmentReportDataTable = (columns = [], reportData = [], dataTableId, reportType) => {}**
    This function creates a DataTable with specific configurations and settings, handles sending assigned emails via AJAX requests to *'includes/ajax.php'*, and displays the data transformed based on the report type. It also integrates functionality for exporting data to Excel.
18. **$('#view_category').on('change', function() {})**
    This code dynamically loads scenario options based on the selected category when the user changes the category selection. It provides feedback to the user during the loading process and handles the display of scenarios based on the response from the server.
19. **$('#view_scenario').on('change', function() {})**
    This code is another event handler, triggered when the value of an element with the ID view_scenario changes, typically a dropdown menu for selecting scenarios.
20. **$('#view_path').on('change', function() {})**
    This code dynamically loads assigned data for the selected path through AJAX request at *'includes/ajax.php'* when the user changes the path selection. It provides feedback during the loading process and generates a report table based on the response from the server.
21. **$('.edit_path_title').on('click', function(){})**
    This script handles the editing of learning path titles. When a user clicks on an element with the class .edit_path_title, it prompts them to update the title. Upon submission, it sends an AJAX request to *'includes/process.php'* to update the title and provides feedback to the user based on the response.

Manage Category

Template Name: category-management.php



PHP Functions Used

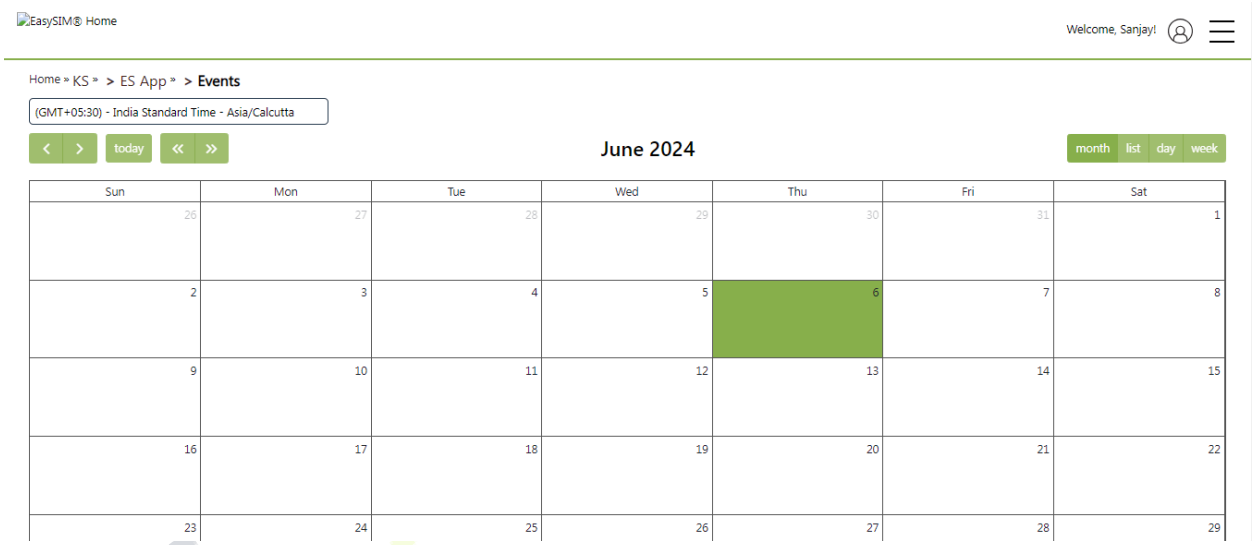
1. **checkLogin()**
   Global function from "db.class.php". The checkLogin function ensures that a user is properly logged in by verifying that all essential session variables are set and not empty.
2. **SQL Queries**
   $sqlr = "SELECT cat_id, category, category, DATE_FORMAT(datetime, '%d-%b-%y') AS date FROM category_tbl ORDER BY cat_id DESC";
   This SQL query retrieves data from the category_tbl table, including category IDs, categories, formatted dates, and orders the result set based on the category IDs in descending order.

JS Functions Used

1. **$('#category').autocomplete({})**
   This script sets up an autocomplete feature for an input field named category, fetching suggestions from a server-side script at *'includes/process.php?type=2'* and displaying them to users as they type.
2. **$('.update_cat').click(function () {})**
   On click on element with class .update_cat, this script retrieves data from the clicked element, shows a loading overlay, loads content from a PHP file *'edit-category-modal.php'* into a modal, and displays the modal with the loaded content. If an error occurs during this process, it displays an error message.
3. **$('.delete_cat').click(function() {})**
   This script prompts the user with a confirmation dialog on click of an element with the class .delete_cat. If the user confirms deletion, it sends a POST request to the server at *'includes/process.php'* with parameters {delete_cat:true, cat_id:id} to delete the category. Depending on the server's response, it displays success or error messages using SweetAlert.

ILT/VILT

Template Name: events.php



PHP Functions Used

1. **checkLogin()**
   Global function from "db.class.php". The checkLogin function ensures that a user is properly logged in by verifying that all essential session variables are set and not empty.
2. **isAdmin()**
   Global function from "db.class.php". This function, isAdmin, checks if the current user has administrative privileges based on their role stored in the session.

JS File Used

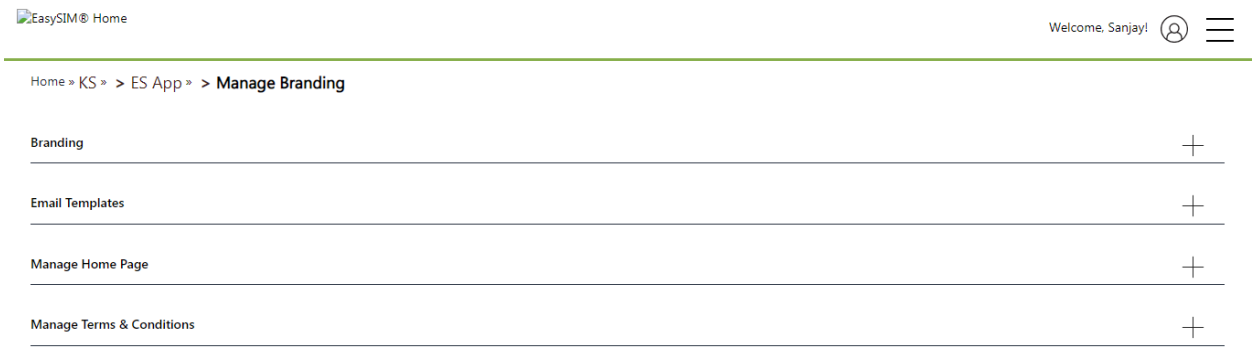
File Path: content/js/events.js

1. **baseUrl = (path = '') => {}**
   The "getBaseUrl" function generates the base URL for a web page, optionally appending a specific page path.
2. **getCategory = () => {}**
   This is a JavaScript function named getCategory which makes an AJAX request to *'includes/ajax.php'* to retrieve category data from a server endpoint.
3. **getGroups = () => {}**
   This is a JavaScript function named getGroups which makes an AJAX request to *'includes/ajax.php'* to retrieve groups data from a server endpoint.
4. **eventsTemp = (data = false, start = false) => {}**
   This function eventsTemp dynamically generates HTML content for adding or editing events based on the provided event data and start time. It includes form elements for event details, attachment uploads, cover image uploads, and group assignments.

5. **viewEventTemp = (info) => {}**
   This function viewEventTemp dynamically generates HTML content to display detailed information about an event, including its title, category, assigned groups, geographical location, description, creator details, and links to the event URL, attachment, and cover image (if available).
6. **$('body').on('change', '#time-zone-selector', function (e) {})**
   This script dynamically updates the time zone setting for a calendar component on a web page and sends the selected time zone to the server at 'includes/ajax.php' for further processing or storage.
7. **$('body').on('click', '.upload_event_attachment', function (e) {})**
   This script handles the uploading of event attachments via AJAX to *'includes/upload.php'*, providing feedback to the user during the process and updating the UI accordingly based on the server's response.
8. **$('body').on('click', '.delete_event_attachment', function (e) {})**
   This script handles the deletion of event attachments via AJAX to *'includes/delete-file.php'*, providing feedback to the user during the process and updating the UI accordingly based on the server's response.
9. **$('body').on('click', '.upload_cover_image', function (e) {})**
   This script handles the upload of cover images via AJAX to 'includes/upload.php', providing feedback to the user during the process and updating the UI accordingly based on the server's response.
10. **$('body').on('click', '.delete_cover_image', function (e) {})**
    This script provides a confirmation dialog before deleting the cover image and then sends an AJAX request to the server at 'includes/delete-file.php' to delete the image file. It updates the UI accordingly based on the server's response.
11. **$('body').on('click', '.view_event_attachment,.view_cover_image', function (e) {})**
    This script essentially allows users to view attachments or cover images in a modal window (using Fancybox) without leaving the current page. It enhances the user experience by providing a seamless and visually appealing way to preview images.

Manage Branding

Template Name: manage-branding.php

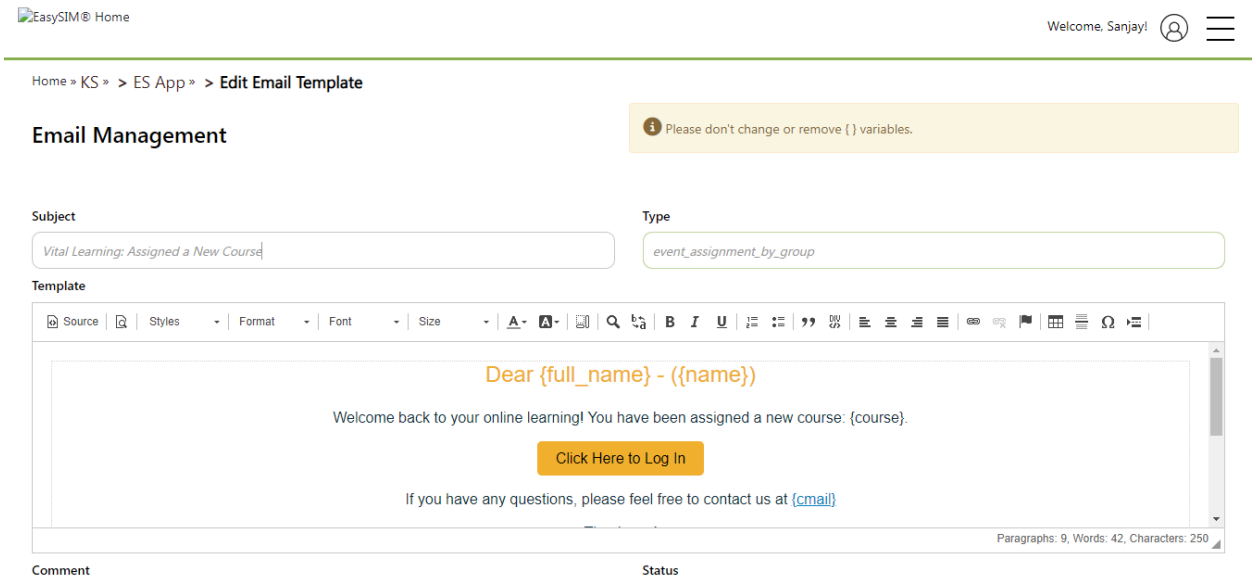


PHP Functions Used

1. **webBranding($client_id)**
   Global function from "db.class.php". This function returns all brand related information for the client.
   a. **public function webBranding($cid = NULL)**
   This function takes a single argument $client_id of type integer. This argument represents the unique identifier of the client whose brand information are to be fetched.
   b. **$sql = "SELECT * FROM client_branding WHERE client_id = "".$client_id.""";**
      i. The SQL query selects all the columns form table client_branding where client_id matches.
2. **Email Template Data -**
   a. sqlr = "SELECT eid, subject, type, comment, status, DATE_FORMAT(edate, '%d-%b-%y') AS date FROM email_template_tbl ORDER BY eid DESC";
   This SQL query selects all email template data from table email_template_tbl and create list to be shown on the page.
3. **getCMSPage($page_id)**
   Global function from "db.class.php". This function returns page content information according to the page id passed as argument which is 1 for the "Home" page and 2 for "Terms & Conditions" page.
   a. $sql = "SELECT * FROM cms_page_tbl WHERE cms_page_id= '$page_id'";
   This SQL query select CMS Page content according to the page id passed as argument from table cms_page_tbl.

Manage Branding > Edit Email Template

Template Name: edit-email-template.php

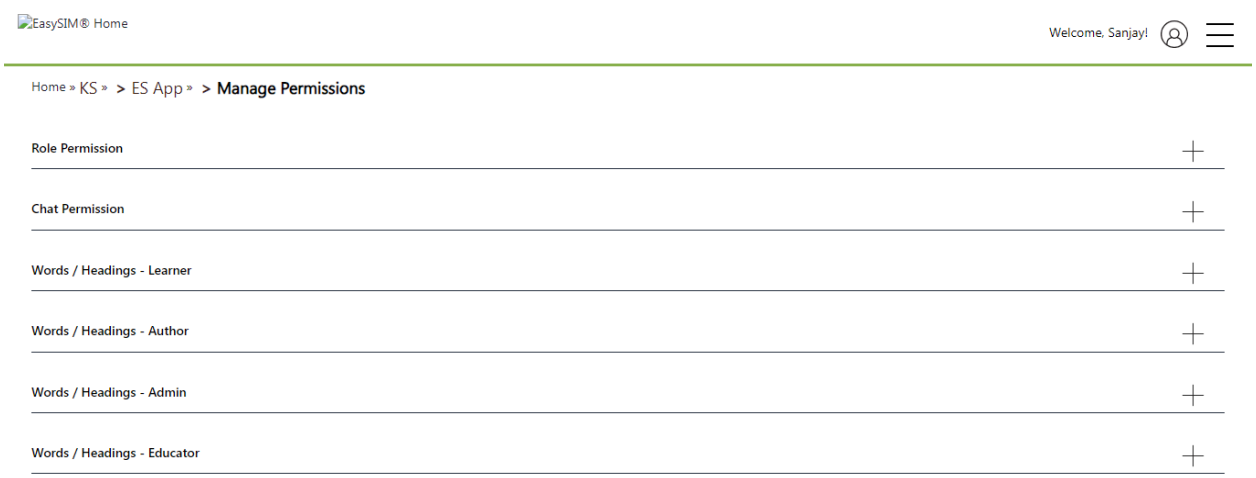


PHP Functions Used

1. **getEmailTemplate($id)**
   GlGlobal function from "db.class.php". This function returns information about email template according to the template id passed as argument.
   a. sql = "SELECT * FROM email_template_tbl WHERE 1=1 AND MD5(eid) = '". $id ."'";
      This SQL query select all email template information as per template id passed from table email_template_tbl.

Manage Permissions

Template Name: manage-permissions.php



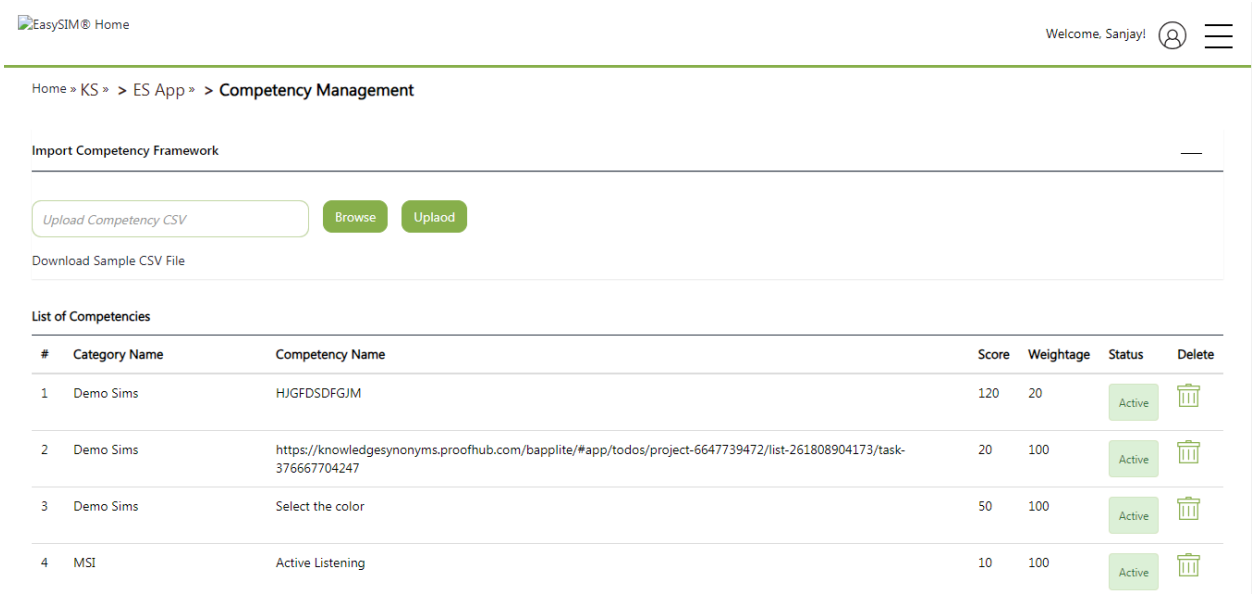
PHP Functions Used

1. **getClientPermission($client_id)**
   Global function from "db.class.php". This function returns all information regarding permission for the client according to the client id passed as argument.
   a. $sql = "SELECT *, (admin + auth + reviewer + learner + educator) AS totalUsers FROM client_permissions WHERE client_id= '$client_id'";
      This SQL query select all the permission information including total no. of users for the client as per client id passed as argument.
2. **getRole()**
   Global function from "db.class.php". This function return all the roles created from role table
   a. $sql = "SELECT * FROM role";
3. **getHeading($type_id, 'id, heading, status')**
   Global function from "db.class.php". This function returns all the headings for the $type_id for different roles like Learner, Author passed as an 1st argument and columns names separated by commas as 2nd argument.
   a. $sql = "SELECT $column_names FROM headings WHERE type= $type_id";
   Type IDs for different types of roles are as follows:

Role	**Type ID**
Learner	1
Author	2
Admin	3
Educator	4

Competency Management

Template Name: competency-management.php

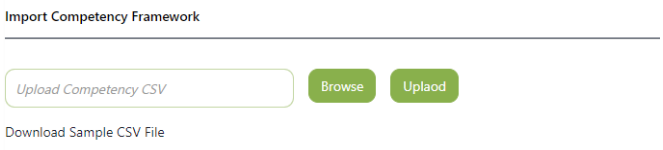


PHP Functions Used

1. **Competency Data -**
   $sqlr = "SELECT * FROM competency_lib WHERE 1=1 ORDER BY cid DESC";
   This SQL query selects competency data from table competency_lib creates list to be shown with their status.

JS Functions Used

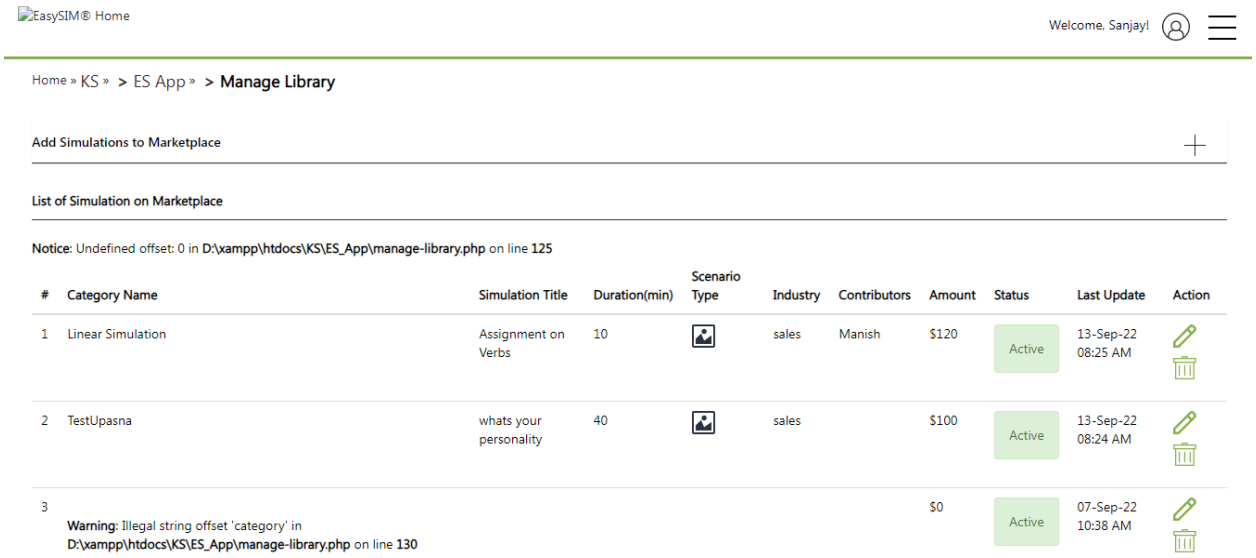
1. **$('#upload').click()**



   This click function POST CSV file of competency to server at *'import-csv-file.php'* bulk Competencies creation.
2. **const deleteComp = (id) => {}**
   This JS function is called on the click of delete icon, sends POST request at *'includes/process.php'* with arguments delete_comp_lib as true and cid as competency ID.
3. **const updateStatus = (status, id) => {}**
   This JS function is called on the click of Status against each competency and sends POST request at *'includes/process.php'* with arguments status_lib_sim as true, status to be changed to and cid as competency ID.

Manage Library

Template Name: manage-library.php

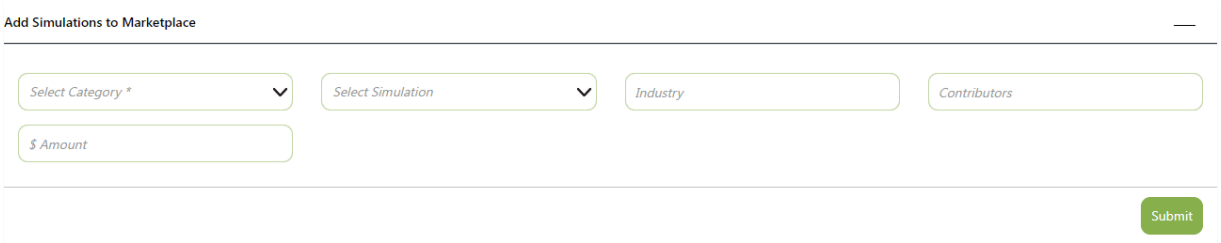


PHP Functions Used

1. **getClientPermission($client_id)**
   Global function from "db.class.php". This function returns all information regarding permission for the client according to the client id passed as argument.
   a. $sql = "SELECT *, (admin + auth + reviewer + learner + educator) AS totalUsers FROM client_permissions WHERE client_id= '$client_id'";
      This SQL query selects all the permission information including total no. of users for the client as per client id passed as argument.
2. **List of Simulation on Marketplace -**
   $sqlr = "SELECT *, DATE_FORMAT(updated_at, '%d-%b-%y %h:%i %p') AS last_update FROM library_sim WHERE client_id = '$client_id' ORDER BY lid DESC";
   This SQL query selects library simulation information from library_sim table for the client id passed.
3. **List of Purchased Simulations -**
   $psql = "SELECT *, DATE_FORMAT(created_at, '%d-%b-%y %h:%i %p') AS cdate FROM library_sim_payments WHERE client_id = '$client_id' ORDER BY pay_id DESC";
   This SQL query selects all purchased simulation information from library_sim_payments table for the client id passed.

JS Functions Used

1. **$('#category').on('change', function() {})**



This JS function is fired on change of Category and in response gets list of Simulaitons for the selected category in JSON format which will be updated in the “Select Simulation” dropdown.

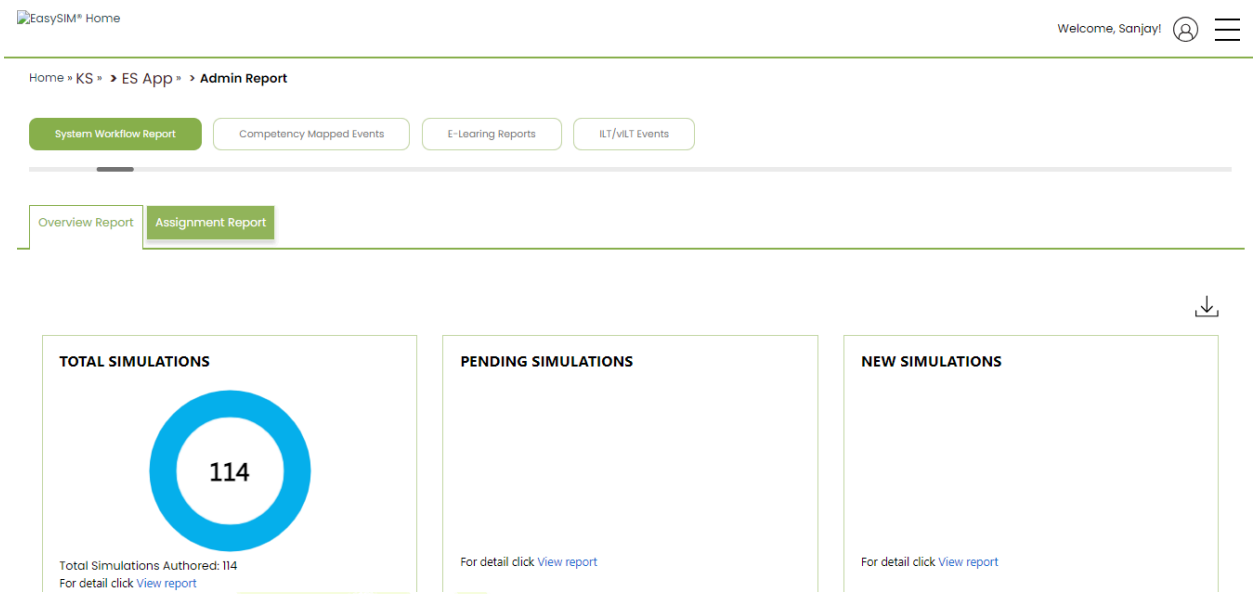
2. **const updateStatus = (type, id) => {}**
   This JS function is called on click of Status (Active/Inactive) against each Simulation and updates the status accordingly as per the type and simulation id passed as arguments.
3. **const editLibSim = (id) => {}**

This function fetches details of Simulation as per id passed in argument and opens form in popup for editing details with pre-filled values.

4. **const deleteLibSim = (id) => {}**
   This function sends request to API with Simulation ID for deletion and in return gets confirmation.

View Report > System Workflow Report > Overview Report

Template Name: admin-report.php



PHP Functions Used

1. **checkLogin()**
   Global function from "db.class.php". The checkLogin function ensures that a user is properly logged in by verifying that all essential session variables are set and not empty.
2. **SQL Queries**
   $sql = "SELECT DISTINCT s.Scenario_title, GROUP_CONCAT(DISTINCT a.group_id SEPARATOR ',') as groupid FROM scenario_master s LEFT JOIN assignment_tbl a ON s.scenario_id = a.scenario_id INNER JOIN group_tbl g LEFT JOIN category_tbl c ON s.scenario_category = c.cat_id WHERE s.assign_author != '' AND s.author_scenario_type != '0' AND a.group_id IS NOT NULL GROUP BY a.scenario_id";
   This query retrieves distinct scenario titles along with the corresponding group IDs, aggregating multiple group IDs into a single concatenated string for each scenario. It provides a comprehensive view of scenarios and their assigned groups.

   $sqlp = "SELECT DISTINCT scenario_id FROM scenario_master WHERE scenario_id NOT IN (SELECT scenario_id FROM question_tbl)";
   This SQL query retrieves distinct scenario IDs from the scenario_master table where the scenario ID is not present in the question_tbl.

   $sqlnew = "SELECT Scenario_title FROM scenario_master WHERE sim_ques_type = '0'";
   This SQL query selects Scenario_title from the scenario_master table where the sim_ques_type column is equal to '0'.

   $sqlreviewed = "SELECT Scenario_title FROM scenario_master WHERE status = '14'";
   This SQL query selects the Scenario_title from the scenario_master table where the

status column is equal to '14'.

$sqlpro = "SELECT Scenario_title FROM scenario_master WHERE status = '14'";
This SQL query selects the Scenario_title from the scenario_master table where the status column is equal to '14'.

$sqlreview = "SELECT Scenario_title FROM scenario_master WHERE status = '15'";
This SQL query selects the Scenario_title from the scenario_master table where the status column is equal to '15'.

$sqlreadypub = "SELECT Scenario_title FROM scenario_master WHERE status = '5'";
This SQL query selects the Scenario_title from the scenario_master table where the status column is equal to '5'.

$sqlpub = "SELECT DISTINCT scenario_id FROM scenario_master WHERE status = '8'";
This SQL query selects distinct scenario_id values from the scenario_master table where the status column is equal to '8'.

Third Party JS Library

1. CanvasJS
2. Polyfill
3. CircleChart
4. JSPdf
5. HTML2Canvas

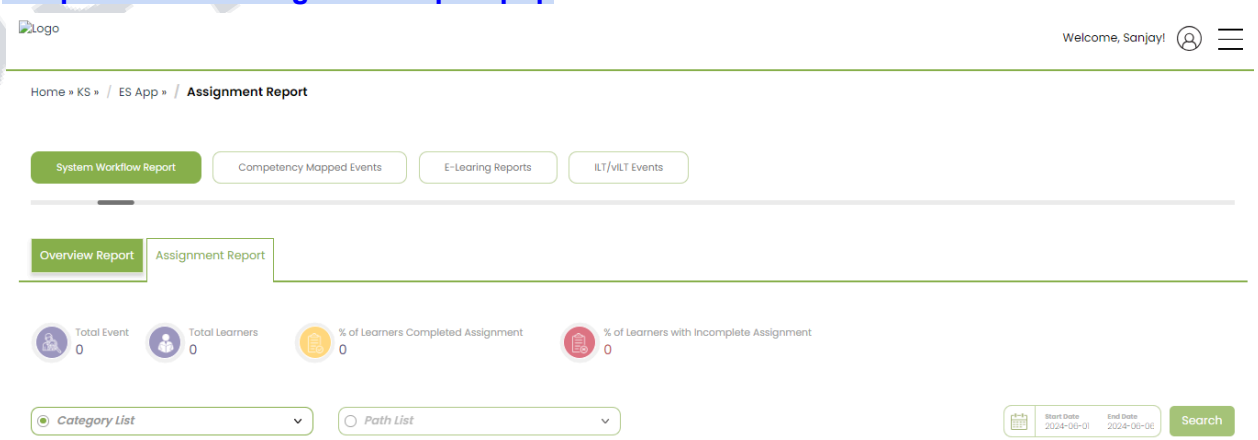
JS Functions Used

1. **chart = new CanvasJS.Chart("chartContainer", {})**
   This code snippet is using the CanvasJS library to create a column chart. This sets up a column chart with customized appearance and tooltip content, using data points provided by PHP.
2. **pendingchart = new CanvasJS.Chart("pendingchartContainer", {})**
   This code snippet utilizes the CanvasJS library to create a pie chart with customized appearance and renders it within the specified HTML container. The data points are provided by PHP and encoded using JSON.
3. **newchart = new CanvasJS.Chart("newchartContainer", {})**
   This code creates a doughnut chart using the CanvasJS library and sets up a doughnut chart with a customizable title and renders it within the specified HTML container. The data points are provided by PHP and encoded using JSON.
4. **reviewedchart = new CanvasJS.Chart("reviewedchartContainer", {})**
   This code creates a funnel chart using the CanvasJS library and sets up a funnel chart with customizable styling and renders it within the specified HTML container. The data points are provided by PHP and encoded using JSON.

5. **reviewprochart = new CanvasJS.Chart("reviewprodataPoints", {})**
   This code creates a pie chart using the CanvasJS library and sets up a pie chart with customizable styling and renders it within the specified HTML container. The data points are provided by PHP and encoded using JSON.
6. **reviewfixedchart = new CanvasJS.Chart("reviewfixedchartContainer", {})**
   This JavaScript code creates a pyramid chart using the CanvasJS library and sets up a pyramid chart with customizable styling and renders it within the specified HTML container. There seems to be a redundant colorSet property, which might need to be corrected.
7. **readypubchart = new CanvasJS.Chart("readypubdataPoints", {})**
   This code sets up a bar chart with customizable styling and dynamic data, and it also calculates and displays a summary based on the chart data.
8. **$(".circleChart").circleChart({})**
   This JavaScript code initializes a circular chart using the CircleChart plugin and initializes a circular chart with customizable colors, size, thickness, animation, and text properties.
9. **function getPDF(id) {}**
   This JavaScript function, getPDF(id), converts an HTML element specified by its ID into a PDF document.

View Report > System Workflow Report > Assignment Report

Template Name: assignment-report.php



PHP Functions Used

1. **checkLogin()**
   Global function from “db.class.php”. The checkLogin function ensures that a user is properly logged in by verifying that all essential session variables are set and not empty.

Third Party JS Library

1. Datatables
2. DateRangePicker

JS File Used

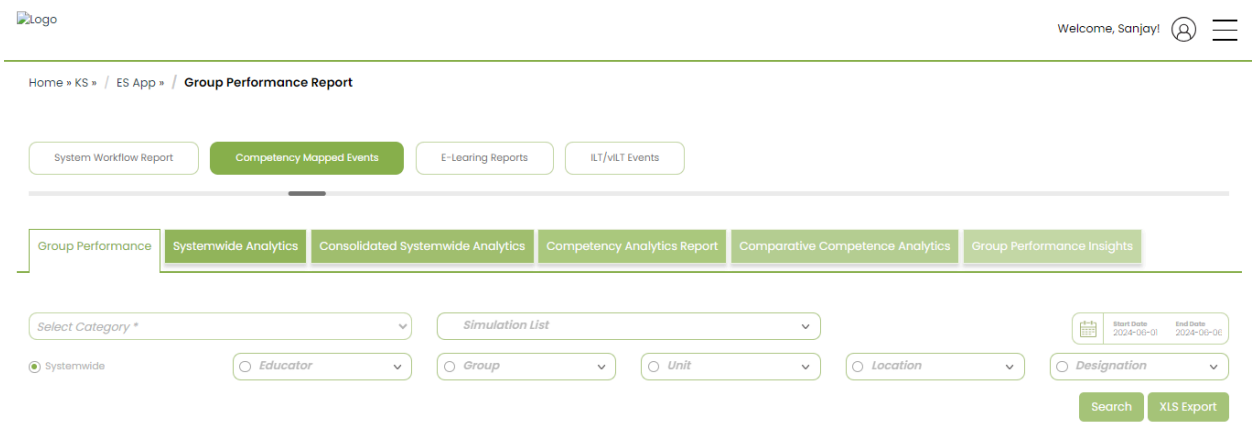
File Path: content/js/admin/learner-report.js

1. **getPercentage = (x, y, html = false) => {}**
   This JavaScript code defines a function named "getPercentage" that calculates the percentage of x relative to y, optionally returning the result as a formatted string with a percentage sign.
2. **transformData = (filterData) => {}**
   This function is useful for converting an array of user data objects into a format suitable for tabular display or further manipulation, providing consistency and flexibility in handling the data.
3. **transformCategoryPathAssignmentData = (filterData, report_type) => {}**
   This transformCategoryPathAssignmentData function takes two parameters: filterData, which is an object containing filtered data, and report_type, which specifies the type of report being generated. The function transforms the input data into a specific format based on the report type.
4. **genReportDataTable = (columns = [], reportData = [], dataTableId) => {}**
   This genReportDataTable function is responsible for generating and populating a DataTable with provided columns and report data.
5. **genCategoryPathReportDataTable = (columns = [], reportData = [], dataTableId, report_type) => {}**
   This function genCategoryPathReportDataTable generates a data table for displaying reports related to category or path assignments.
6. **CircleBar = () => {}**
   This JavaScript code defines a function named "CircleBar" that dynamically updates the style of elements representing circular progress bars on a webpage.
7. **resetCircleBar = () => {}**
   This function ensures that the circular progress bars and their associated text indicators are reset to zero before potentially updating them with new data.
8. **cb(start, end) {}**
   This JavaScript code defines a function named "cb" that updates the values of specific input elements with formatted date strings based on the given start and end dates.
9. **$('#reportrange').daterangepicker({})**
   This code initializes a date range picker with specific configuration options and preset date ranges for users to choose from.
10. **$('.multiselection').multiselect({})**
    This code snippet initializes a multi-select dropdown menu for elements with the class .multiselection.

11. **$('#category').on('change', function() {})**
    This code dynamically populates dropdown options based on the selected category and handles loading overlays, AJAX requests to *'includes/ajax.php'*, and error messages.
12. **$('#scenario,#scenario_multi').on('change', function(){})**
    This code dynamically populates dropdown options and enables/disables filters based on the selected scenario. It also handles AJAX requests to 'includes/ajax.php' and responses to fetch and populate data.
13. **$("#group_performance").submit(function(e) {})**
    This script handles the form submission, makes an AJAX request to *'includes/admin/learner-report-data.php'* to fetch data, and populates a DataTable or generates a report based on the received data.
14. **$.post(ajax, { get_assigned_groups: true })**
    This script is used to fetch assigned groups from the server from *'includes/ajax.php'* and update the options of a multiselect dropdown accordingly.
15. **$('#group_repo').on('change', function() {})**
    This script is used to dynamically update the options of a dropdown menu for learners based on the selected group. It also resets the table and circle bar and displays a loading overlay during the operation.
16. **$("#elearning_group_report").submit(function(e) {})**
    This script sets up a form submission event listener for the form with the id #elearning_group_report. When this form is submitted, the script prevents the default form submission behavior, displays a loading overlay, and sends an AJAX request to the server at *'includes/admin/learner-report-data.php'* to process the form data.
17. **$("#assignment_report").submit(function(e) {})**
    This script sets up a form submission event listener for the form with the id #assignment_report. When this form is submitted, the script prevents the default form submission behavior, displays a loading overlay, and sends an AJAX request to the server at *'includes/admin/learner-report-data.php'* to process the form data.

View Report > Competency Mapped Events > Group Performance

Template Name: group-performance-report.php



PHP Functions Used

1. **checkLogin()**
   Global function from "db.class.php". The checkLogin function ensures that a user is properly logged in by verifying that all essential session variables are set and not empty.
2. **getRole()**
   Global function from "db.class.php". This PHP function getRole() is designed to retrieve roles from a database table named "role" and return them as an associative array.
3. **SQL Queries**
   $sql = "SELECT GROUP_CONCAT(DISTINCT IF(group_id = '', null, group_id)) AS gid, GROUP_CONCAT(DISTINCT IF(learner_id = '', null, learner_id)) AS lid FROM assignment_tbl WHERE scenario_id IN ($get_sim_id)";
   This query selects distinct group_id and learner_id values from the assignment_tbl table where the scenario_id matches one of the values provided in the $get_sim_id variable. The result is returned as concatenated strings, with null values excluded and separated by commas.

   $date_sql = "SELECT GROUP_CONCAT(userid) AS learner FROM scenario_attempt_tbl WHERE scenario_id IN ($get_sim_id) AND userid IN ($get_learner_id) AND DATE(attempt_date) BETWEEN '". $fdate ."' AND '". $tdate ."'";
   This query retrieves the userid values from the scenario_attempt_tbl table where the scenario_id matches any of the values provided in $get_sim_id, the userid matches any of the values provided in $get_learner_id, and the attempt_date falls within the specified date range. The result is returned as a concatenated string of userid values.

Third Party JS Library

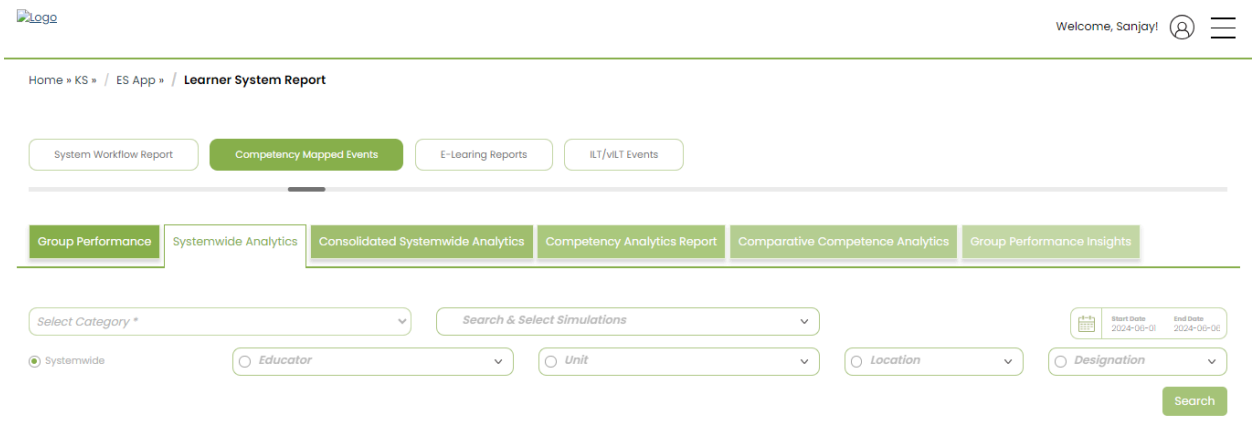
1. Datatables
2. DateRangePicker
3. Bootstrap-DateTimePicker

JS Functions Used

1. **cb(start, end) {}**
   This JavaScript code defines a function named "cb" that updates the values of specific input elements with formatted date strings based on the given start and end dates.
2. **$('#reportrange').daterangepicker({})**
   This JavaScript code initializes a date range picker using the daterangepicker function provided by a library (likely Moment.js or a similar date library) and configures its options.
3. **$('.form_datetime').datetimepicker({})**
   This code initializes a datetime picker using the datetimepicker function, likely provided by a library such as Bootstrap DateTimePicker or jQuery DateTimePicker.
4. **$('#example').DataTable({})**
   This configuration sets up a DataTable without sorting functionality, with a fixed display of 20 entries per page, and without a search/filter input field.
5. **$('#educator').on('change', function() {})**
   This code dynamically updates the value of an input field based on the selected options in a dropdown list.
6. **$('.multiselection').multiselect({})**
   This code configures the multiselect dropdown to provide a user-friendly interface for selecting multiple options from a list.
7. **$('.filter_type').on('change', function() {})**
   This code controls the behavior of multiselect dropdowns based on the selected option in another dropdown, allowing users to filter data based on different criteria.
8. **$('#category').on('change', function() {})**
   This JavaScript code snippet is an event listener that triggers when the value of the element with the ID #category changes. It sends a GET request to the specified URL (*includes/ajax.php*) with the catid as a parameter. Inside the callback function, it updates the options in the #scenario dropdown using $('#scenario').multiselect('dataprovider', res.data).
9. **$('#scenario').on('change', function() {})**
   This JavaScript code snippet is an event listener that triggers when the value of the element with the ID #scenario changes. It sends a GET request to the specified URL (*includes/ajax.php*) with the sid as a parameter. Inside the callback function, populates various dropdown menus (#educator, #unit, #location, #designation, #group, #learner) with data retrieved from the server using the .multiselect('dataprovider', res.data) method. It also manages enabling and disabling certain filter elements based on the value of res.assign_group.

View Report > Competency Mapped Events > Systemwide Analytics

Template Name: learner-system-report.php



PHP Functions Used

1. **checkLogin()**
   Global function from "db.class.php". The checkLogin function ensures that a user is properly logged in by verifying that all essential session variables are set and not empty.
2. **getRole()**
   Global function from "db.class.php". This PHP function getRole() is designed to retrieve roles from a database table named "role" and return them as an associative array.
3. **SQL Queries**
   $sqlComp = "SELECT (SUM(comp_val_1) + SUM(comp_val_2) + SUM(comp_val_3) + SUM(comp_val_4) + SUM(comp_val_5) + SUM(comp_val_6)) as totalCompval FROM competency_tbl WHERE scenario_id = '" . $scenario . "'";
   This query calculates the total value of competencies (totalCompval) for a specific scenario ID in the competency_tbl table.

   $sql = "SELECT GROUP_CONCAT(DISTINCT IF(group_id = '', null, group_id)) AS gid, GROUP_CONCAT(DISTINCT IF(learner_id = '', null, learner_id)) AS lid FROM assignment_tbl WHERE FIND_IN_SET('". $scenario."', scenario_id)";
   This query retrieves concatenated lists of distinct group IDs (gid) and learner IDs (lid) from the assignment_tbl table where the scenario ID matches the specified value.

Third Party JS Library

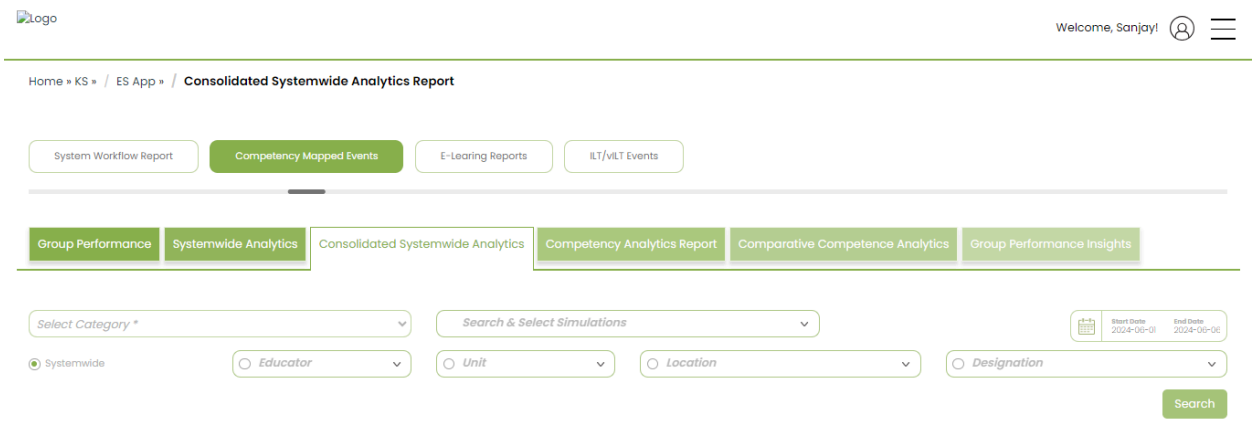
1. Polyfills
2. JSPdf
3. HTML2Canvas
4. Datatables
5. DateRangePicker
6. Bootstrap-DateTimePicker
7. TooltipJS

JS Functions Used

1. **getPDF(id) {}**
   This JavaScript function, getPDF(id), converts an HTML element specified by its ID into a PDF document.
2. **cb(start, end) {}**
   This JavaScript code defines a function named "cb" that updates the values of specific input elements with formatted date strings based on the given start and end dates.
3. **$('#reportrange').daterangepicker({})**
   This JavaScript code initializes the date range picker for an HTML element with the ID reportrange.
4. **$('.form_datetime').datetimepicker({})**
   This JavaScript code initializes a datetime picker for HTML elements with the class form_datetime.
5. **$('.collapse-init').on('click', function() {})**
   This code toggles the collapse state of elements and updates the button text accordingly between "Expand All" and "Collapse All".
6. **$('#educator').on('change', function() {})**
   This JavaScript code snippet listens for changes on the #educator element. It retrieves the text content of all selected options and updates the value of another input field with the id #get_educator to reflect the selected options.
7. **$('.multiselection').multiselect({})**
   This code snippet initializes a dropdown multiselect widget using the multiselect function on elements with the class multiselection.
8. **$('.filter_type').on('change', function() {})**
   This code dynamically controls the behavior of multiselect dropdowns based on the selected filter type, enabling or disabling them as necessary.
9. **$('#category').on('change', function() {})**
   This JavaScript code snippet is an event listener attached to the change event of an element with the ID category. It handles the behavior of elements on the page when the category selection changes.
10. **$('#scenario').multiselect({})**
    This code dynamically loads additional data from "*'includes/ajax.php*?get_system_reporting_other_data=true&sim_id=' + sim_id" based on the selected options and adjusts the dropdown behavior accordingly, providing a smooth user experience.

View Report > Competency Mapped Events > Consolidated Systemwide Analytics

Template Name: consolidated-systemwide-analytics-report.php



PHP Functions Used

1. **checkLogin()**
   Global function from "db.class.php". The checkLogin function ensures that a user is properly logged in by verifying that all essential session variables are set and not empty.
2. **getRole()**
   Global function from "db.class.php". This PHP function getRole() is designed to retrieve roles from a database table named "role" and return them as an associative array.
3. **SQL Queries**
   *$sqlComp = "SELECT *, (SUM(comp_val_1) + SUM(comp_val_2) + SUM(comp_val_3) + SUM(comp_val_4) + SUM(comp_val_5) + SUM(comp_val_6)) as totalCompval FROM competency_tbl WHERE scenario_id = ?";*
   This SQL query selects all columns (*) from the table competency_tbl and calculates the sum of several columns (comp_val_1 through comp_val_6) for each row. The total sum is aliased as totalCompval.

   $sql = "SELECT GROUP_CONCAT(DISTINCT IF(group_id = '', null, group_id)) AS gid, GROUP_CONCAT(DISTINCT IF(learner_id = '', null, learner_id)) AS lid FROM assignment_tbl WHERE FIND_IN_SET('". $scenario."', scenario_id)";
   This query retrieves concatenated lists of distinct group IDs (gid) and learner IDs (lid) from the assignment_tbl table where the scenario ID matches the specified value.

Third Party JS Library

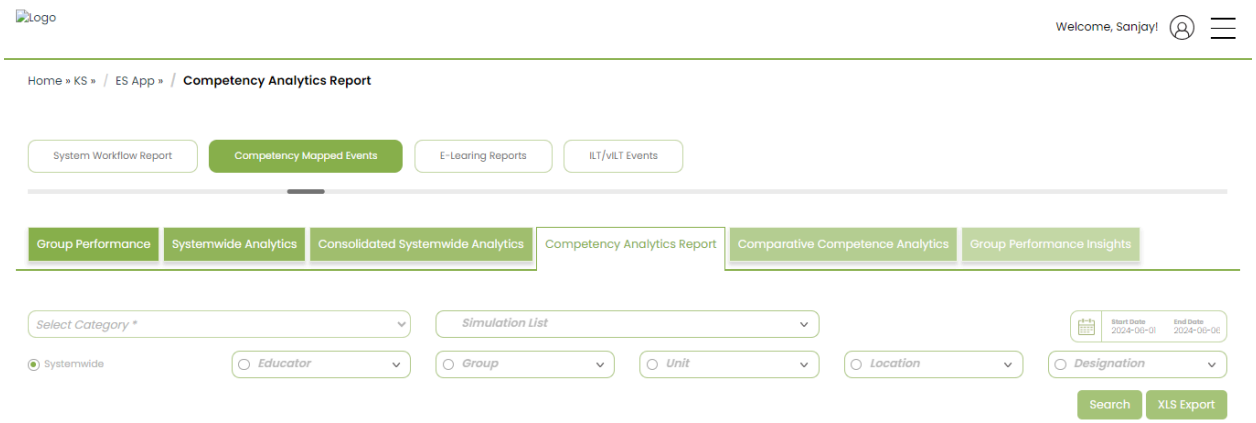
1. Datatables
2. DateRangePicker
3. Bootstrap-DateTimePicker
4. TooltipJS

JS Functions Used

1. **cb(start, end) {}**
   The provided code defines a function named cb that updates the values of specific input fields based on the start and end dates provided as arguments. This function is handling date ranges in a web application, using a library like jQuery and a date picker plugin.
2. **$('#reportrange').daterangepicker({})**
   This code snippet initializes a date range picker using the daterangepicker function provided by the Bootstrap Datepicker library.
3. **$('#example').DataTable({})**
   This script initializes a DataTable using the DataTable() function provided by the DataTables library.
4. **$('.form_datetime').datetimepicker({})**
   This code initializes a datetime picker using the Bootstrap DateTimePicker plugin for the elements with the class "form_datetime".
5. **$('#educator').on('change', function() {})**
   This code is a jQuery event handler that listens for changes on the educator select element. When a change event occurs (i.e., when the selected option in the dropdown changes), the associated function is executed.
6. **$('.multiselection').multiselect({})**
   This code initializes a multiselect dropdown menu for elements with the class multiselection.
7. **$('.filter_type').on('change', function() {})**
   This script is a part of a filtering mechanism where different options can be selected to filter data displayed on a webpage, and the multiselect dropdowns are enabled or disabled based on the selected filter type.
8. **$('#category').on('change', function() {})**
   This script sets up an event listener for changes on the element with the ID category. When the value of this element changes, the script executes a function.
9. **$('#scenario').multiselect({})**
   This script initializes a multiselect dropdown on an element with the ID scenario. The multiselect plugin is configured with various options. Additionally, an onChange event handler is defined for the multiselect dropdown. This function is called whenever an option in the dropdown is changed (selected or deselected). It also makes AJAX request to a specified URL (*includes/ajax.php*) with sim_id value as a parameter.

View Report > Competency Mapped Events > Competency Analytics Report

Template Name: competency-analytics-report.php



PHP Functions Used

1. **checkLogin()**
   Global function from "db.class.php". The checkLogin function ensures that a user is properly logged in by verifying that all essential session variables are set and not empty.
2. **getRole()**
   Global function from "db.class.php". This PHP function getRole() is designed to retrieve roles from a database table named "role" and return them as an associative array.
3. **SQL Queries**
   $sql = "SELECT GROUP_CONCAT(DISTINCT IF(group_id = '', null, group_id)) AS gid, GROUP_CONCAT(DISTINCT IF(learner_id = '', null, learner_id)) AS lid FROM assignment_tbl WHERE scenario_id IN ($get_sim_id)";
   This query selects distinct group_id and learner_id values from the assignment_tbl table where the scenario_id matches one of the values provided in the $get_sim_id variable. The result is returned as concatenated strings, with null values excluded and separated by commas.

   $date_sql = "SELECT GROUP_CONCAT(userid) AS learner FROM scenario_attempt_tbl WHERE scenario_id IN ($get_sim_id) AND userid IN ($get_learner_id) AND DATE(attempt_date) BETWEEN '". $fdate ."' AND '". $tdate ."'";
   This query retrieves the userid values from the scenario_attempt_tbl table where the scenario_id matches any of the values provided in $get_sim_id, the userid matches any of the values provided in $get_learner_id, and the attempt_date falls within the specified date range. The result is returned as a concatenated string of userid values.

Third Party JS Library

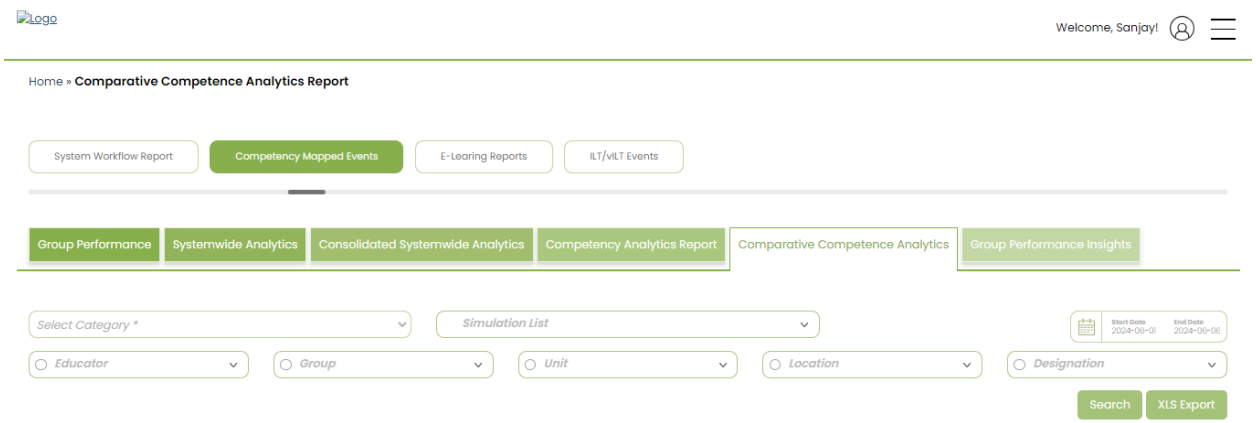
1. Datatables
2. DateRangePicker
3. Bootstrap-DateTimePicker

JS Functions Used

1. **cb(start, end) {}**
   This JavaScript code defines a function named "cb" that updates the values of specific input elements with formatted date strings based on the given start and end dates.
2. **$('#reportrange').daterangepicker({})**
   This triggers date range picker library for the selection of dates for reports generated.
3. **$('.form_datetime').datetimepicker({})**
   This code sets up a datetime picker for HTML elements with the class ".form_datetime".
4. **$('#example').DataTable({})**
   This configuration sets up a DataTable without sorting functionality, with a fixed display of 20 entries per page, and without a search/filter input field.
5. **$('#educator').on('change', function() {})**
   This code dynamically updates the value of an input field based on the selected options in a dropdown list.
6. **$('.multiselection').multiselect({})**
   This code configures the multiselect dropdown to provide a user-friendly interface for selecting multiple options from a list.
7. **$('.filter_type').on('change', function() {})**
   This code controls the behavior of multiselect dropdowns based on the selected option in another dropdown, allowing users to filter data based on different criteria.
8. **$('#category').on('change', function() {})**
   This JavaScript code snippet is an event listener that triggers when the value of the element with the ID #category changes. It sends a GET request to the specified URL (*includes/ajax.php*) with the catid as a parameter. Inside the callback function, it updates the options in the #scenario dropdown using $('#scenario').multiselect('dataprovider', res.data).
9. **$('#scenario').on('change', function() {})**
   This JavaScript code snippet is an event listener that triggers when the value of the element with the ID #scenario changes. It sends a GET request to the specified URL (*includes/ajax.php*) with the sid as a parameter. Inside the callback function, populates various dropdown menus (#educator, #unit, #location, #designation, #group, #learner) with data retrieved from the server using the .multiselect('dataprovider', res.data) method. It also manages enabling and disabling certain filter elements based on the value of res.assign_group.

View Report > Competency Mapped Events > Comparative Competence Analytics

Template Name: comparison-competency-analytics-report.php



PHP Functions Used

1. **checkLogin()**
   Global function from "db.class.php". The checkLogin function ensures that a user is properly logged in by verifying that all essential session variables are set and not empty.
2. **getRole()**
   Global function from "db.class.php". This PHP function getRole() is designed to retrieve roles from a database table named "role" and return them as an associative array.
3. **SQL Queries**
   $sql = "SELECT GROUP_CONCAT(DISTINCT IF(group_id = '', null, group_id)) AS gid, GROUP_CONCAT(DISTINCT IF(learner_id = '', null, learner_id)) AS lid FROM assignment_tbl WHERE scenario_id IN ($sim_id)";
   This SQL query selects distinct group IDs (group_id) and learner IDs (learner_id) from the assignment_tbl table based on the condition that the scenario_id is included in the list of simulation IDs provided by the variable $sim_id.
4. **filterLearner($id, $cond)**
   Global function from "db.class.php". This function filters learner IDs based on certain conditions and returns the filtered list of IDs.
5. **simDetails($sid)**
   Global function from "db.class.php". This function retrieves the title of a simulation based on its ID from the database.
6. **groupDetails($gid)**
   Global function from "db.class.php". This function, groupDetails($gid), retrieves the name of a group based on the provided group ID.
7. **getSimAssign($gid)**
   Global function from "db.class.php". This function, getSimAssign($gid), retrieves the simulation IDs assigned to a specific group.
8. **getSimAssignByLearner($lid)**
   Global function from "db.class.php". This function, getSimAssignByLearner($lid), retrieves the simulation IDs assigned to a specific learner.

Third Party JS Library

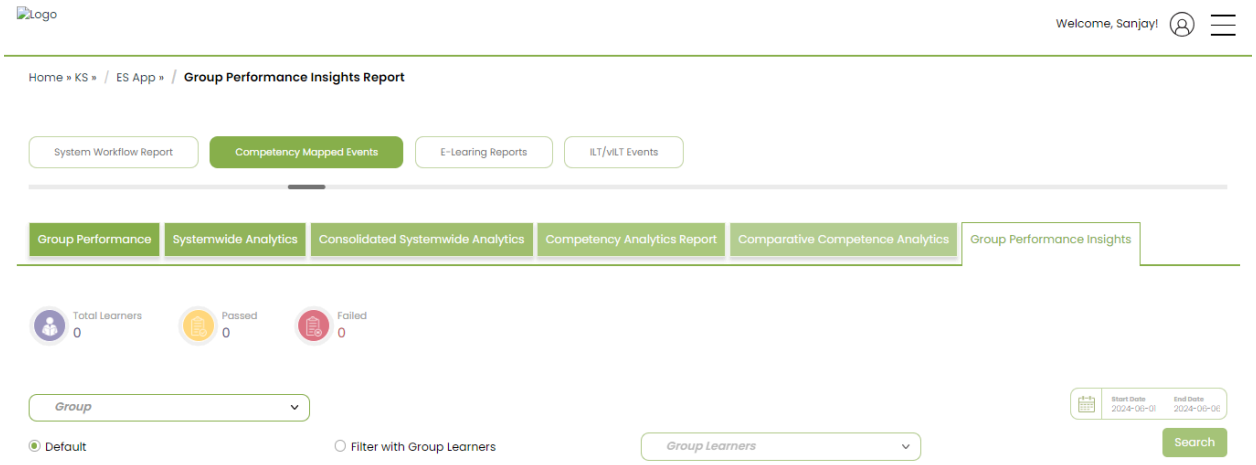
1. Datatables
2. DateRangePicker
3. Bootstrap-DateTimePicker

JS Functions Used

1. **cb(start, end) {}**
   This JavaScript code defines a function named "cb" that updates the values of specific input elements with formatted date strings based on the given start and end dates.
2. **$('#reportrange').daterangepicker({})**
   This triggers date range picker library for the selection of dates for reports generated.
3. **$('.form_datetime').datetimepicker({})**
   This code sets up a datetime picker for HTML elements with the class ".form_datetime".
4. **$('.comparison_comp_res').DataTable({})**
   This DataTable configuration sets up a table with buttons, pagination, and information display enabled, but it disables sorting and searching functionalities.
5. **$('#educator').on('change', function() {})**
   This code dynamically updates the value of an input field based on the selected options in a dropdown list.
6. **$('.multiselection').multiselect({})**
   This code configures the multiselect dropdown to provide a user-friendly interface for selecting multiple options from a list.
7. **$('.filter_type').on('change', function() {})**
   This code controls the behavior of multiselect dropdowns based on the selected option in another dropdown, allowing users to filter data based on different criteria.
8. **$('#category').on('change', function() {})**
   This JavaScript code snippet is an event listener that triggers when the value of the element with the ID #category changes. It sends a GET request to the specified URL (*includes/ajax.php*) with the catid as a parameter. Inside the callback function, it updates the options in the #scenario dropdown using $('#scenario').multiselect('dataprovider', res.data).
9. **$('#scenario').on('change', function() {})**
   This JavaScript code snippet is an event listener that triggers when the value of the element with the ID #scenario changes. It sends a GET request to the specified URL (*includes/ajax.php*) with the sid as a parameter. Inside the callback function, populates various dropdown menus (#educator, #unit, #location, #designation, #group, #learner) with data retrieved from the server using the .multiselect('dataprovider', res.data) method. It also manages enabling and disabling certain filter elements based on the value of res.assign_group.

View Report > Competency Mapped Events > Group Performance Insights

Template Name: group-performance-insights-report.php



PHP Functions Used

1. **getAssigneeName($userid)**
   Global function from "db.class.php". This function will return all the data for user as per the User ID passed in the argument.
2. **checkLogin()**
   Global function from "db.class.php". The checkLogin function ensures that a user is properly logged in by verifying that all essential session variables are set and not empty.

Third Party JS Library

1. Datatables
2. DateRangePicker

JS File Used

File Path: content/js/admin/learner-report.js

1. **getPercentage = (x, y, html = false) => {}**
   This JavaScript code defines a function named "getPercentage" that calculates the percentage of x relative to y, optionally returning the result as a formatted string with a percentage sign.
2. **transformData = (filterData) => {}**
   This function is useful for converting an array of user data objects into a format suitable for tabular display or further manipulation, providing consistency and flexibility in handling the data.
3. **transformCategoryPathAssignmentData = (filterData, report_type) => {}**
   This transformCategoryPathAssignmentData function takes two parameters: filterData, which is an object containing filtered data, and report_type, which specifies the type of

report being generated. The function transforms the input data into a specific format based on the report type.

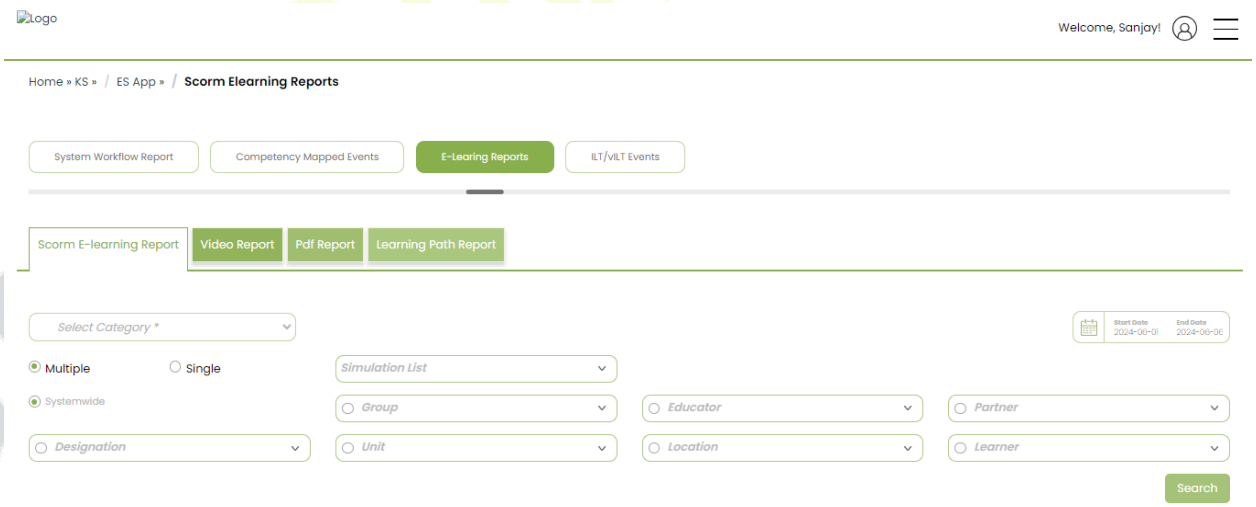
4. **genReportDataTable = (columns = [], reportData = [], dataTableId) => {}**
   This genReportDataTable function is responsible for generating and populating a DataTable with provided columns and report data.
5. **genCategoryPathReportDataTable = (columns = [], reportData = [], dataTableId, report_type) => {}**
   This function genCategoryPathReportDataTable generates a data table for displaying reports related to category or path assignments.
6. **CircleBar = () => {}**
   This JavaScript code defines a function named "CircleBar" that dynamically updates the style of elements representing circular progress bars on a webpage.
7. **resetCircleBar = () => {}**
   This function ensures that the circular progress bars and their associated text indicators are reset to zero before potentially updating them with new data.
8. **cb(start, end) {}**
   This JavaScript code defines a function named "cb" that updates the values of specific input elements with formatted date strings based on the given start and end dates.
9. **$('#reportrange').daterangepicker({})**
   This code initializes a date range picker with specific configuration options and preset date ranges for users to choose from.
10. **$('.multiselection').multiselect({})**
    This code snippet initializes a multi-select dropdown menu for elements with the class .multiselection.
11. **$('#category').on('change', function() {})**
    This code dynamically populates dropdown options based on the selected category and handles loading overlays, AJAX requests to *'includes/ajax.php'*, and error messages.
12. **$('#scenario,#scenario_multi').on('change', function(){})**
    This code dynamically populates dropdown options and enables/disables filters based on the selected scenario. It also handles AJAX requests to *'includes/ajax.php'* and responses to fetch and populate data.
13. **$("#group_performance").submit(function(e) {})**
    This script handles the form submission, makes an AJAX request to *'includes/admin/learner-report-data.php'* to fetch data, and populates a DataTable or generates a report based on the received data.
14. **$.post(ajax, { get_assigned_groups: true })**
    This script is used to fetch assigned groups from the server from *'includes/ajax.php'* and update the options of a multiselect dropdown accordingly.
15. **$('#group_repo').on('change', function() {})**
    This script is used to dynamically update the options of a dropdown menu for learners based on the selected group. It also resets the table and circle bar and displays a loading overlay during the operation.
16. **$("#elearning_group_report").submit(function(e) {})**
    This script sets up a form submission event listener for the form with the id

#elearning_group_report. When this form is submitted, the script prevents the default form submission behavior, displays a loading overlay, and sends an AJAX request to the server at *'includes/admin/learner-report-data.php'* to process the form data.

17. **$("#assignment_report").submit(function(e) {})**
    This script sets up a form submission event listener for the form with the id #assignment_report. When this form is submitted, the script prevents the default form submission behavior, displays a loading overlay, and sends an AJAX request to the server at *'includes/admin/learner-report-data.php'* to process the form data.

View Report > E-Learning Reports > Scorm E-Learning Report

Template Name: scorm-elearning-reports.php



PHP Functions Used

1. **getAssigneeName($userid)**
   Global function from "db.class.php". This function will return all the data for user as per the User ID passed in the argument.
2. **checkLogin()**
   Global function from "db.class.php". The checkLogin function ensures that a user is properly logged in by verifying that all essential session variables are set and not empty.

Third Party JS Library

1. Datatables
2. DateRangePicker

JS File Used

File Path: content/js/admin/learner-report.js

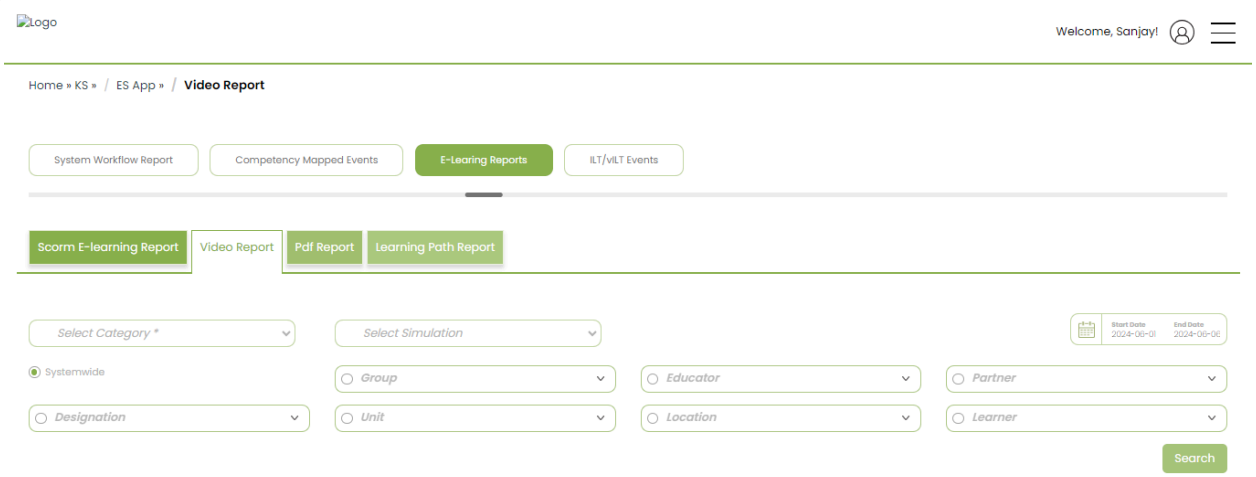
1. **getPercentage = (x, y, html = false) => {}**
   This JavaScript code defines a function named "getPercentage" that calculates the percentage of x relative to y, optionally returning the result as a formatted string with a percentage sign.
2. **transformData = (filterData) => {}**
   This function is useful for converting an array of user data objects into a format suitable for tabular display or further manipulation, providing consistency and flexibility in handling the data.
3. **transformCategoryPathAssignmentData = (filterData, report_type) => {}**
   This transformCategoryPathAssignmentData function takes two parameters: filterData, which is an object containing filtered data, and report_type, which specifies the type of report being generated. The function transforms the input data into a specific format based on the report type.
4. **genReportDataTable = (columns = [], reportData = [], dataTableId) => {}**
   This genReportDataTable function is responsible for generating and populating a DataTable with provided columns and report data.
5. **genCategoryPathReportDataTable = (columns = [], reportData = [], dataTableId, report_type) => {}**
   This function genCategoryPathReportDataTable generates a data table for displaying reports related to category or path assignments.
6. **CircleBar = () => {}**
   This JavaScript code defines a function named "CircleBar" that dynamically updates the style of elements representing circular progress bars on a webpage.
7. **resetCircleBar = () => {}**
   This function ensures that the circular progress bars and their associated text indicators are reset to zero before potentially updating them with new data.
8. **cb(start, end) {}**
   This JavaScript code defines a function named "cb" that updates the values of specific input elements with formatted date strings based on the given start and end dates.
9. **$('#reportrange').daterangepicker({})**
   This code initializes a date range picker with specific configuration options and preset date ranges for users to choose from.
10. **$('.multiselection').multiselect({})**
    This code snippet initializes a multi-select dropdown menu for elements with the class .multiselection.
11. **$('#category').on('change', function() {})**
    This code dynamically populates dropdown options based on the selected category and handles loading overlays, AJAX requests to *'includes/ajax.php'*, and error messages.
12. **$('#scenario,#scenario_multi').on('change', function(){})**
    This code dynamically populates dropdown options and enables/disables filters based on the selected scenario. It also handles AJAX requests to *'includes/ajax.php'* and responses to fetch and populate data.
13. **$("#group_performance").submit(function(e) {})**
    This script handles the form submission, makes an AJAX request to

'includes/admin/learner-report-data.php' to fetch data, and populates a DataTable or generates a report based on the received data.

14. **$.post(ajax, { get_assigned_groups: true })**
    This script is used to fetch assigned groups from the server from *'includes/ajax.php'* and update the options of a multiselect dropdown accordingly.
15. **$('#group_repo').on('change', function() {})**
    This script is used to dynamically update the options of a dropdown menu for learners based on the selected group. It also resets the table and circle bar and displays a loading overlay during the operation.
16. **$("#elearning_group_report").submit(function(e) {})**
    This script sets up a form submission event listener for the form with the id #elearning_group_report. When this form is submitted, the script prevents the default form submission behavior, displays a loading overlay, and sends an AJAX request to the server at *'includes/admin/learner-report-data.php'* to process the form data.
17. **$("#assignment_report").submit(function(e) {})**
    This script sets up a form submission event listener for the form with the id #assignment_report. When this form is submitted, the script prevents the default form submission behavior, displays a loading overlay, and sends an AJAX request to the server at *'includes/admin/learner-report-data.php'* to process the form data.

View Report > E-Learning Reports > Video Report

Template Name: video-report.php



PHP Functions Used

1. **getAssigneeName($userid)**
   Global function from "db.class.php". This function will return all the data for user as per the User ID passed in the argument.

2. **checkLogin()**
   Global function from “db.class.php”. The checkLogin function ensures that a user is properly logged in by verifying that all essential session variables are set and not empty.

Third Party JS Library

1. Datatables
2. DateRangePicker

JS File Used

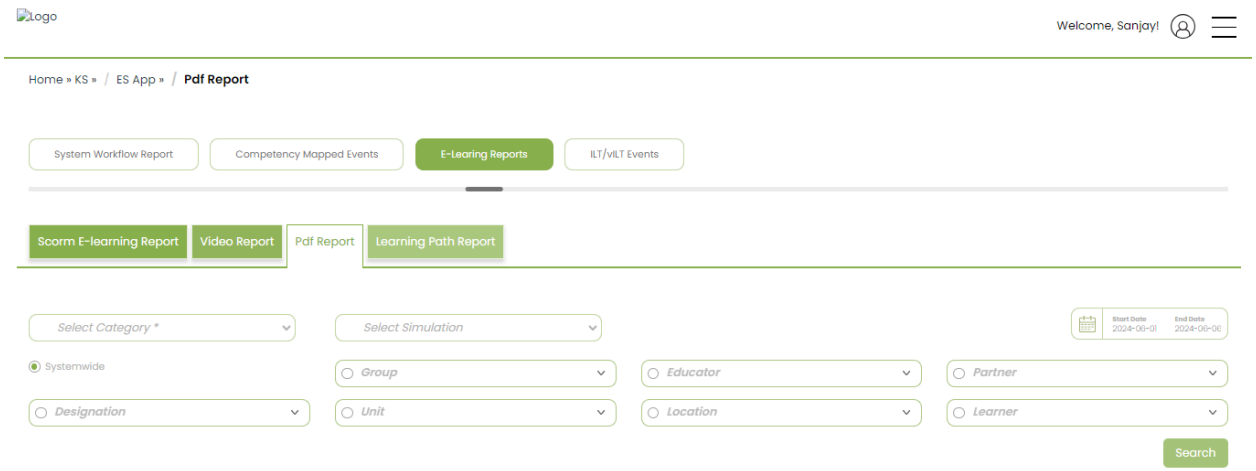
File Path: content/js/admin/learner-report.js

1. **getPercentage = (x, y, html = false) => {}**
   This JavaScript code defines a function named "getPercentage" that calculates the percentage of x relative to y, optionally returning the result as a formatted string with a percentage sign.
2. **transformData = (filterData) => {}**
   This function is useful for converting an array of user data objects into a format suitable for tabular display or further manipulation, providing consistency and flexibility in handling the data.
3. **transformCategoryPathAssignmentData = (filterData, report_type) => {}**
   This transformCategoryPathAssignmentData function takes two parameters: filterData, which is an object containing filtered data, and report_type, which specifies the type of report being generated. The function transforms the input data into a specific format based on the report type.
4. **genReportDataTable = (columns = [], reportData = [], dataTableId) => {}**
   This genReportDataTable function is responsible for generating and populating a DataTable with provided columns and report data.
5. **genCategoryPathReportDataTable = (columns = [], reportData = [], dataTableId, report_type) => {}**
   This function genCategoryPathReportDataTable generates a data table for displaying reports related to category or path assignments.
6. **CircleBar = () => {}**
   This JavaScript code defines a function named "CircleBar" that dynamically updates the style of elements representing circular progress bars on a webpage.
7. **resetCircleBar = () => {}**
   This function ensures that the circular progress bars and their associated text indicators are reset to zero before potentially updating them with new data.
8. **cb(start, end) {}**
   This JavaScript code defines a function named "cb" that updates the values of specific input elements with formatted date strings based on the given start and end dates.

9. **$('#reportrange').daterangepicker({})**
   This code initializes a date range picker with specific configuration options and preset date ranges for users to choose from.
10. **$('.multiselection').multiselect({})**
    This code snippet initializes a multi-select dropdown menu for elements with the class .multiselection.
11. **$('#category').on('change', function() {})**
    This code dynamically populates dropdown options based on the selected category and handles loading overlays, AJAX requests to *'includes/ajax.php'*, and error messages.
12. **$('#scenario,#scenario_multi').on('change', function(){})**
    This code dynamically populates dropdown options and enables/disables filters based on the selected scenario. It also handles AJAX requests to *'includes/ajax.php'* and responses to fetch and populate data.
13. **$("#group_performance").submit(function(e) {})**
    This script handles the form submission, makes an AJAX request to *'includes/admin/learner-report-data.php'* to fetch data, and populates a DataTable or generates a report based on the received data.
14. **$.post(ajax, { get_assigned_groups: true })**
    This script is used to fetch assigned groups from the server from *'includes/ajax.php'* and update the options of a multiselect dropdown accordingly.
15. **$('#group_repo').on('change', function() {})**
    This script is used to dynamically update the options of a dropdown menu for learners based on the selected group. It also resets the table and circle bar and displays a loading overlay during the operation.
16. **$("#elearning_group_report").submit(function(e) {})**
    This script sets up a form submission event listener for the form with the id #elearning_group_report. When this form is submitted, the script prevents the default form submission behavior, displays a loading overlay, and sends an AJAX request to the server at *'includes/admin/learner-report-data.php'* to process the form data.
17. **$("#assignment_report").submit(function(e) {})**
    This script sets up a form submission event listener for the form with the id #assignment_report. When this form is submitted, the script prevents the default form submission behavior, displays a loading overlay, and sends an AJAX request to the server at *'includes/admin/learner-report-data.php'* to process the form data.

View Report > E-Learning Reports > PDF Report

Template Name: pdf-report.php



PHP Functions Used

1. **getAssigneeName($userid)**
   Global function from "db.class.php". This function will return all the data for user as per the User ID passed in the argument.
2. **checkLogin()**
   Global function from "db.class.php". The checkLogin function ensures that a user is properly logged in by verifying that all essential session variables are set and not empty.

Third Party JS Library

1. Datatables
2. DateRangePicker

JS File Used

File Path: content/js/admin/learner-report.js

1. **getPercentage = (x, y, html = false) => {}**
   This JavaScript code defines a function named "getPercentage" that calculates the percentage of x relative to y, optionally returning the result as a formatted string with a percentage sign.
2. **transformData = (filterData) => {}**
   This function is useful for converting an array of user data objects into a format suitable for tabular display or further manipulation, providing consistency and flexibility in handling the data.
3. **transformCategoryPathAssignmentData = (filterData, report_type) => {}**
   This transformCategoryPathAssignmentData function takes two parameters: filterData, which is an object containing filtered data, and report_type, which specifies the type of report being generated. The function transforms the input data into a specific format based on the report type.

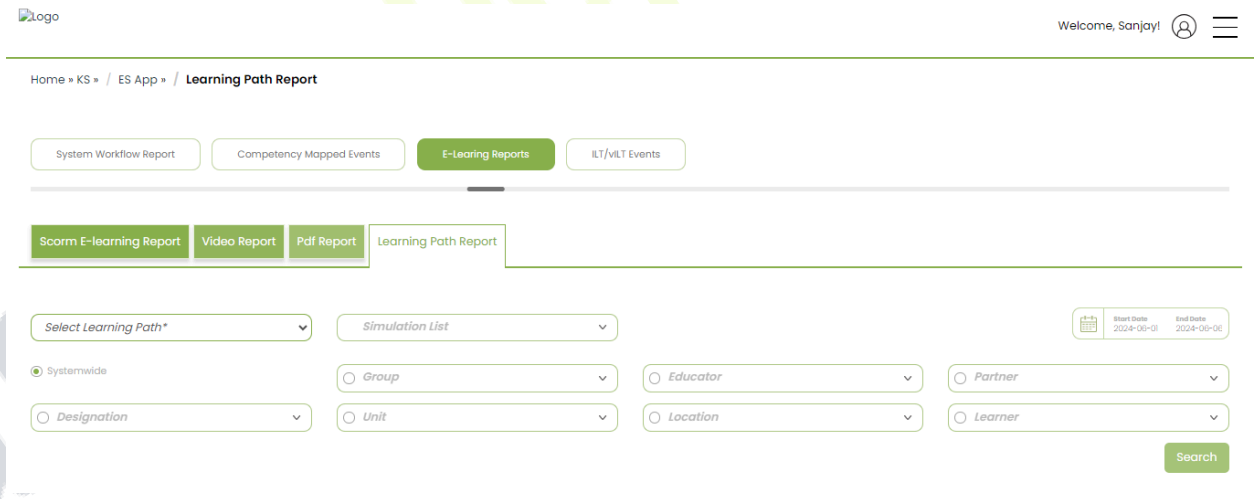
4. **genReportDataTable = (columns = [], reportData = [], dataTableId) => {}**
   This genReportDataTable function is responsible for generating and populating a DataTable with provided columns and report data.
5. **genCategoryPathReportDataTable = (columns = [], reportData = [], dataTableId, report_type) => {}**
   This function genCategoryPathReportDataTable generates a data table for displaying reports related to category or path assignments.
6. **CircleBar = () => {}**
   This JavaScript code defines a function named "CircleBar" that dynamically updates the style of elements representing circular progress bars on a webpage.
7. **resetCircleBar = () => {}**
   This function ensures that the circular progress bars and their associated text indicators are reset to zero before potentially updating them with new data.
8. **cb(start, end) {}**
   This JavaScript code defines a function named "cb" that updates the values of specific input elements with formatted date strings based on the given start and end dates.
9. **$('#reportrange').daterangepicker({})**
   This code initializes a date range picker with specific configuration options and preset date ranges for users to choose from.
10. **$('.multiselection').multiselect({})**
    This code snippet initializes a multi-select dropdown menu for elements with the class .multiselection.
11. **$('#category').on('change', function() {})**
    This code dynamically populates dropdown options based on the selected category and handles loading overlays, AJAX requests to *'includes/ajax.php'*, and error messages.
12. **$('#scenario,#scenario_multi').on('change', function(){})**
    This code dynamically populates dropdown options and enables/disables filters based on the selected scenario. It also handles AJAX requests to *'includes/ajax.php'* and responses to fetch and populate data.
13. **$("#group_performance").submit(function(e) {})**
    This script handles the form submission, makes an AJAX request to *'includes/admin/learner-report-data.php'* to fetch data, and populates a DataTable or generates a report based on the received data.
14. **$.post(ajax, { get_assigned_groups: true })**
    This script is used to fetch assigned groups from the server from *'includes/ajax.php'* and update the options of a multiselect dropdown accordingly.
15. **$('#group_repo').on('change', function() {})**
    This script is used to dynamically update the options of a dropdown menu for learners based on the selected group. It also resets the table and circle bar and displays a loading overlay during the operation.
16. **$("#elearning_group_report").submit(function(e) {})**
    This script sets up a form submission event listener for the form with the id #elearning_group_report. When this form is submitted, the script prevents the default

form submission behavior, displays a loading overlay, and sends an AJAX request to the server at *'includes/admin/learner-report-data.php'* to process the form data.

17. **$("#assignment_report").submit(function(e) {})**
    This script sets up a form submission event listener for the form with the id #assignment_report. When this form is submitted, the script prevents the default form submission behavior, displays a loading overlay, and sends an AJAX request to the server at *'includes/admin/learner-report-data.php'* to process the form data.

View Report > E-Learning Reports > Learning Path Report

Template Name: learning-path-report.php



PHP Functions Used

1. **getAssigneeName($userid)**
   Global function from "db.class.php". This function will return all the data for user as per the User ID passed in the argument.
2. **checkLogin()**
   Global function from "db.class.php". The checkLogin function ensures that a user is properly logged in by verifying that all essential session variables are set and not empty.

Third Party JS Library

1. Datatables
2. DateRangePicker

JS File Used

File Path: content/js/admin/event-report.js

1. **getPercentage = (x, y, html = false) => {}**
   This JavaScript code defines a function named "getPercentage" that calculates the

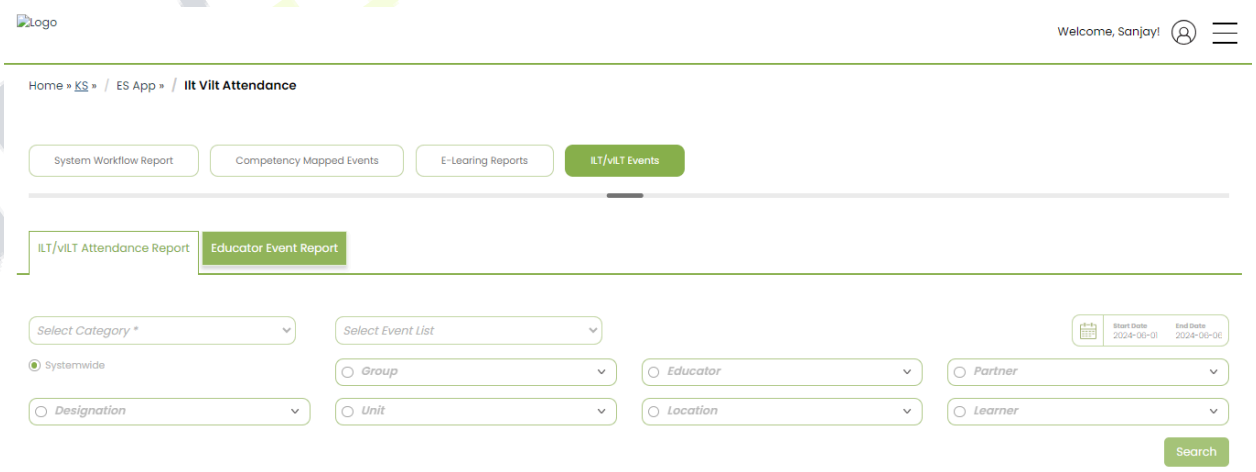
percentage of x relative to y, optionally returning the result as a formatted string with a percentage sign.

2. **cb(start, end) {}**
   The provided code defines a function named cb that updates the values of specific input fields based on the start and end dates provided as arguments. This function is handling date ranges in a web application, using a library like jQuery and a date picker plugin.
3. **$('#reportrange').daterangepicker({})**
   This triggers date range picker library for the selection of dates for reports generated.
4. **$('.multiselection').multiselect({})**
   This code snippet initializes a multi-select dropdown menu for elements with the class .multiselection.
5. **$('.filter_type').on('change', function() {})**
   This code controls the behavior of multiselect dropdowns based on the selected option in another dropdown, allowing users to filter data based on different criteria.
6. **resetTable = () => {}**
   The resetTable function is a JavaScript function designed to reset and clear specific DataTables on a webpage, as well as remove certain elements.
7. **transformData = (filterData) => {}**
   This function is useful for converting an array of user data objects into a format suitable for tabular display or further manipulation, providing consistency and flexibility in handling the data.
8. **learningPathReport = (columns = [], reportData = []) => {}**
   The learningPathReport function is a JavaScript function that initializes and configures a DataTable on an HTML element with the ID #path_report_tbl.
9. **$('#category').on('change', function() {})**
   This code snippet sets up an event listener for changes on a select element with the ID category. When the selected option changes, it triggers a series of actions to update another element with the ID scenario based on the selected category.
10. **$('#event').on('change', function(){})**
    The provided code is a jQuery script that attaches an event listener to an HTML element with the ID #event. This listener responds to changes in the element's value, typically a dropdown menu or input field.
11. **$("#group_performance").submit(function(e) {})**
    The provided code is a jQuery script that handles the submission of a form with the ID #group_performance. It prevents the default form submission behavior, sends an AJAX request to *'includes/admin/learner-report-data.php'* to fetch data, and processes the response to populate a DataTable.
12. **$('#path').on('change', function() {})**
    The provided code is a jQuery script that handles the change event for an HTML element with the ID #path. When the value of this element changes, it triggers a series of actions to reset tables, show loading overlays, disable certain elements, and fetch new data via an AJAX request to *'includes/ajax.php'* + '?getEducatorScenarioPath=true&path='+ id.

13. **$('#scenario').on('change', function(){})**
    The provided code is a jQuery script that handles the change event for an HTML element with the ID #scenario. When the value of this element changes, it performs a series of actions to reset tables, disable certain elements, and fetch new data via an AJAX request to *'includes/ajax.php*?get_system_reporting_other_data=true&sim_id=${id}' to update various multiselect elements based on the new selection.
14. **$("#learning_path_report").submit(function(e) {})**
    The provided code is a jQuery script that handles the submit event for a form with the ID #learning_path_report. When the form is submitted, it prevents the default form submission, shows a loading overlay, resets the table, and sends an AJAX request to *'includes/admin/learner-report-data.php'* to retrieve and display data.

View Report > ILT/VILT Events > ILT/VILT Attendance Report

Template Name: ilt-vilt-attendance.php



PHP Functions Used

1. **getAssigneeName($userid)**
   Global function from "db.class.php". This function will return all the data for user as per the User ID passed in the argument.
2. **checkLogin()**
   Global function from "db.class.php". The checkLogin function ensures that a user is properly logged in by verifying that all essential session variables are set and not empty.

Third Party JS Library

1. Datatables
2. DateRangePicker

JS File Used

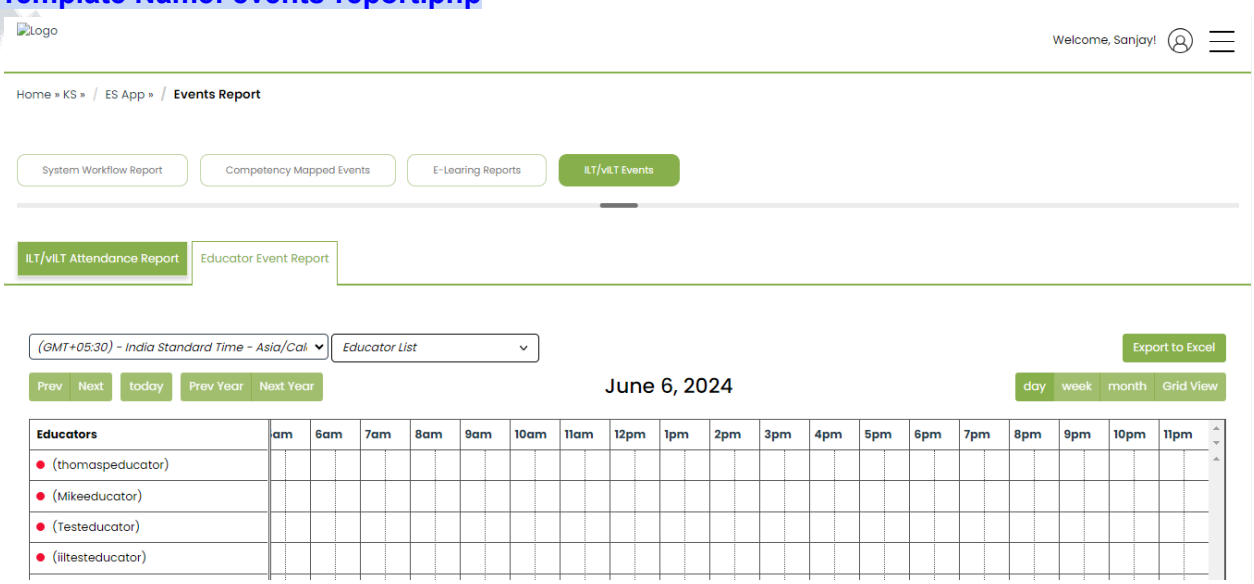
File Path: content/js/admin/event-report.js

1. **getPercentage = (x, y, html = false) => {}**
   This JavaScript code defines a function named "getPercentage" that calculates the percentage of x relative to y, optionally returning the result as a formatted string with a percentage sign.
2. **cb(start, end) {}**
   The provided code defines a function named cb that updates the values of specific input fields based on the start and end dates provided as arguments. This function is handling date ranges in a web application, using a library like jQuery and a date picker plugin.
3. **$('#reportrange').daterangepicker({})**
   This triggers date range picker library for the selection of dates for reports generated.
4. **$('.multiselection').multiselect({})**
   This code snippet initializes a multi-select dropdown menu for elements with the class .multiselection.
5. **$('.filter_type').on('change', function() {})**
   This code controls the behavior of multiselect dropdowns based on the selected option in another dropdown, allowing users to filter data based on different criteria.
6. **resetTable = () => {}**
   The resetTable function is a JavaScript function designed to reset and clear specific DataTables on a webpage, as well as remove certain elements.
7. **transformData = (filterData) => {}**
   This function is useful for converting an array of user data objects into a format suitable for tabular display or further manipulation, providing consistency and flexibility in handling the data.
8. **learningPathReport = (columns = [], reportData = []) => {}**
   The learningPathReport function is a JavaScript function that initializes and configures a DataTable on an HTML element with the ID #path_report_tbl.
9. **$('#category').on('change', function() {})**
   This code snippet sets up an event listener for changes on a select element with the ID category. When the selected option changes, it triggers a series of actions to update another element with the ID scenario based on the selected category.
10. **$('#event').on('change', function(){})**
    The provided code is a jQuery script that attaches an event listener to an HTML element with the ID #event. This listener responds to changes in the element's value, typically a dropdown menu or input field.
11. **$("#group_performance").submit(function(e) {})**
    The provided code is a jQuery script that handles the submission of a form with the ID #group_performance. It prevents the default form submission behavior, sends an AJAX request to *'includes/admin/learner-report-data.php'* to fetch data, and processes the response to populate a DataTable.

12. **$('#path').on('change', function() {})**
    The provided code is a jQuery script that handles the change event for an HTML element with the ID #path. When the value of this element changes, it triggers a series of actions to reset tables, show loading overlays, disable certain elements, and fetch new data via an AJAX request to *'includes/ajax.php'* + '?getEducatorScenarioPath=true&path='+ id.
13. **$('#scenario').on('change', function(){})**
    The provided code is a jQuery script that handles the change event for an HTML element with the ID #scenario. When the value of this element changes, it performs a series of actions to reset tables, disable certain elements, and fetch new data via an AJAX request to *'includes/ajax.php*?get_system_reporting_other_data=true&sim_id=${id}' to update various multiselect elements based on the new selection.
14. **$("#learning_path_report").submit(function(e) {})**
    The provided code is a jQuery script that handles the submit event for a form with the ID #learning_path_report. When the form is submitted, it prevents the default form submission, shows a loading overlay, resets the table, and sends an AJAX request to *'includes/admin/learner-report-data.php'* to retrieve and display data.

View Report > ILT/VILT Events > Educator Event Report

Template Name: events-report.php



PHP Functions Used

1. **checkLogin()**
   Global function from "db.class.php". The checkLogin function ensures that a user is properly logged in by verifying that all essential session variables are set and not empty.

Third Party JS Library

1. FullCalender
2. Fancybox
3. TooltipJS
4. XLSXFull
5. SweelAlert

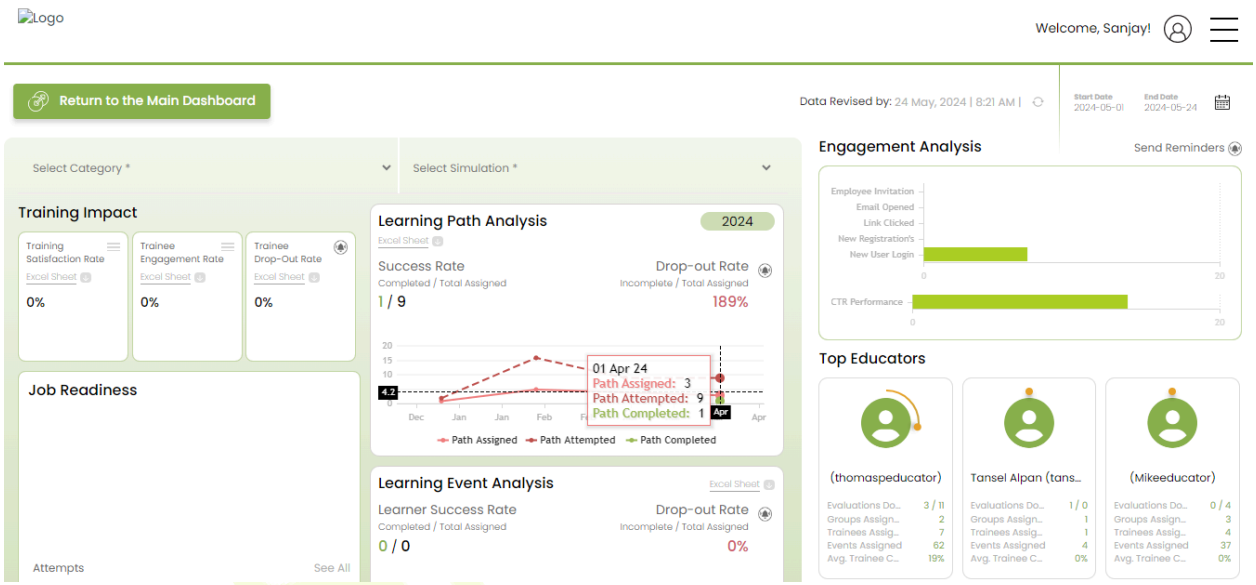
JS File Used

File Path: content/js/admin/event-report-admin.js

1. **baseUrl(path = '') {}**
   The provided function baseUrl constructs and returns a base URL for your application, optionally appending a given path to it.
2. **exportEventsToExcel = (events) => {}**
   The exportEventsToExcel function converts a list of event objects into an Excel file and downloads it.
3. **viewEventTemp = (info) => {}**
   The function viewEventTemp generates an HTML string to display event details in a structured format. This HTML can be used to render the event details in a web page.
4. **calendar = new FullCalendar.Calendar(calendarEl, {})**
   This code initializes a FullCalendar instance and configures various options to customize its appearance and behavior.
5. **$('body').on('change', '#time-zone-selector', function (e) {})**
   This code effectively allows users to change the time zone setting for the FullCalendar dynamically without reloading the page. When a user selects a different time zone from the dropdown menu (#time-zone-selector), it updates the calendar display accordingly and sends the selected time zone information to the server for further processing or storage.
6. **$('#educator').on('change', function() {})**
   This code sets up an event listener using jQuery that listens for changes on the #educator element. When the value of #educator changes, it triggers a series of actions related to FullCalendar and event handling.

Data Driven Insights

Template Name: admin-overview-dashboard.php



This section displays reports regarding various data. Below are the reports generated for a selected period of time.

Training Impact	Job Readiness
Generative Ai Prompt	Learning Path Analysis
Learning Event Analysis	Most liked and disliked events
Engagement Analysis	Top Educators
Performance Analysis	Assignment Status
User Feedback	

Third Party Libraries Used

1. CanvasJS
2. DateRangePicker
3. YearPicker

JS File Used

JS File Path: content/js/admin-overview-dash.js

1. **const tooltip = () => {}**
   This initializes the tooltip functionality.
2. **const getPercentage = (x, y, html = false) => {}**
   "getPercentage" computes the percentage of x out of y and returns it as either a number or a string with a percentage sign, depending on the html parameter.

3. **const progressBar = () => {}**
   The “progressBar” function iterates over each element with the class progressBar, adds a span element inside it, and sets the width of the span to match the value of the data-progress attribute of the original element
4. **const CircleBar = () => {}**
   The "CircleBar" function initializes circular progress bars on elements with specific classes by setting the "stroke-dashoffset" CSS property based on a data attribute.
5. **const calculateTotalMinutes = (timeDurations) => {}**
   The "calculateTotalMinutes" function computes the total number of minutes from an array of time durations given in the format "HH:MM:SS".
6. **const isValidTimeString = (timeString) => {}**
   The "isValidTimeString" function checks if a given string is in a valid time format of "h:m:s" where hours can be negative and consist of one or two digits, and both minutes and seconds consist of exactly two digits
7. **const removeDuplicates = (str) => {}**
   The "removeDuplicates" function removes duplicate words from a comma-separated string and returns an array of unique words.
8. **const calculateTotalProductivityTime = (object) => {}**
   The "calculateTotalProductivityTime" function computes the total productivity time from an object's values (assumed to be in seconds), converts this total time into hours, minutes, and seconds, and updates the HTML of a specific DOM element with the formatted time.
9. **const convertToHMSTimeFormat = (inputObj) => {}**
   The "convertToHMSTimeFormat" function takes an object where the values represent time durations in seconds. It converts each duration into a human-readable format of "HH:MM:SS" and returns a new object with these formatted durations.
10. **const calculateTotalProductivityTimeByRole = (data) => {}**
    The "calculateTotalProductivityTimeByRole" function calculates the total productivity time for each role based on the data provided. It then formats the total time into "HH:MM:SS" format and returns it as an object.
11. **const timeInMinutes = (timeString) => {}**
    The "timeInMinutes" function takes a time string in the format "HH:MM:SS" and converts it into the total number of minutes.
12. **const convertRolesToChartData = (inputData) => {}**
    The "convertRolesToChartData" function converts input data representing time spent by different roles into a format suitable for chart data, specifically for a bar chart.
13. **const groupAndSumData = (data) => {}**
    The "groupAndSumData" function takes an array of data objects, groups them by a common key, sums up the values associated with each group, and returns both the grouped data and the total sum of the values.
14. **const removeDuplicateValues = (inputObj) => {}**
    The "removeDuplicateValues" function takes an object where each key represents a timestamp, and the corresponding value is an array of values associated with that

timestamp. It removes duplicate values from each array and returns a new object with the same timestamps, but with arrays containing only unique values.

15. **const limit = (string = '', limit = 0) => {}**
    The "limit" function is designed to truncate a string to a specified length and append an ellipsis ('...') if the string exceeds the given limit.
16. **function cb(start, end) {}**
    The provided code defines a function named cb that updates the values of specific input fields based on the start and end dates provided as arguments. This function is handling date ranges in a web application, using a library like jQuery and a date picker plugin.
17. **$('#reportrange').daterangepicker({})**
    This triggers date range picker library for the selection of dates for reports generated.
18. **$('#category').on('change', function() {})**
    This function is triggered on change of category which sends api request to *'includes/ajax.php*?getScenarioUpdate=true&catid='+ id' where is the category id passed as a parameter which return list Simulations to be updated in the "Select Simulation" dropdown.

JS Functions Used

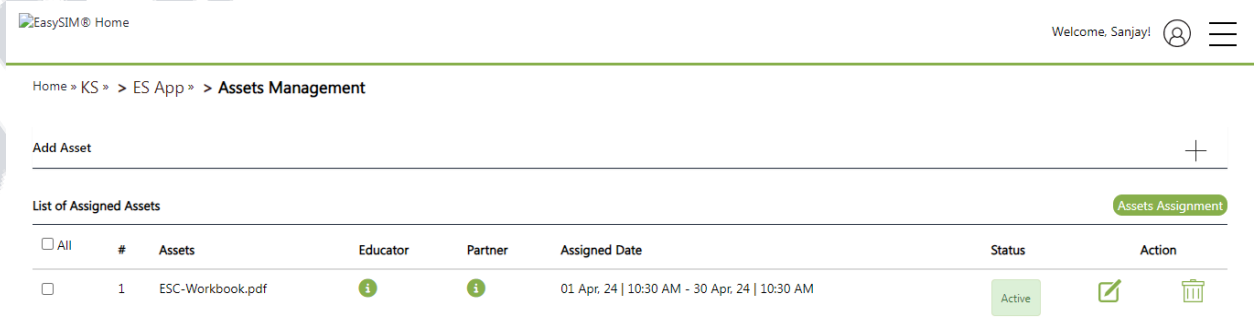
1. **const engAnalysisGraph = (dataPoint) => {}**
   This function creates a graph for "Engagement Analysis" data using CanvasJS.
2. **const ctrAnalysisGraph = (dataPoint) => {}**
   This function creates a graph for "CTA Performance" data using CanvasJS.
3. **const activeUsersGraph = (dataPoint) => {}**
   This function creates a graph for "Active Users" data using CanvasJS.
4. **const allocatedTimeGraph = (allocated_time, taken_time) => {}**
   This function creates a graph for "Allocated Time" data using CanvasJS.
5. **const assignGraph = (tassignment, cassignment) => {}**
   This function creates a graph for "Assignment Timeline" data using CanvasJS.
6. **const remedRecqGraph = (RProvided, RViewed) => {}**
   This function creates a graph for "Remediation Provided" data using CanvasJS.
7. **const bjAnalysis = (assign_path, attempted_path, completed_path, incompleted_path) => {}**
   This function creates a graph for "Learning Path Analysis" data using CanvasJS.
8. **const productivity = (dataPoint) => {}**
   This function creates a graph for "Performance Analysis - Productivity" data using CanvasJS.
9. **const topEducator = (data) => {}**
   This code defines a JavaScript function named topEducator that processes an array of data objects and updates the HTML content of a webpage to display information about the top three educators based on a specific criterion
10. **const trainingSatisfactionGraph = (tsDataPoint) => {}**
    This code defines a JavaScript function named trainingSatisfactionGraph that creates and renders a line chart using the CanvasJS library.

11. **const trainingEngagementRate = (terDataPoint) => {}**
    This code defines a JavaScript function "trainingEngagementRate" that creates and renders a line chart using the CanvasJS library.
12. **const trainingDropOutRate = (tdoDataPoint) => {}**
    This code defines a JavaScript function trainingDropOutRate that creates and renders a line chart using the CanvasJS library.
13. **const toolTipFormatter = (e) => {}**
    This code defines a JavaScript function toolTipFormatter that customizes the content of tooltips for a CanvasJS chart.
14. **const jobReadiness = (jrData) => {}**
    This code defines a JavaScript function "jobReadiness" that creates and renders an area chart using the CanvasJS library.
15. **const updateData = (data) => {}**
    This code defines a JavaScript function “updateData” that processes and updates a data object.
16. **const tirReq = (sid = null, date) => {}**
    This code defines a JavaScript function "tirReq" that performs a series of actions related to the analysis of training impact ratios, including updating engagement and drop-out rates, and displaying job readiness and attempts data.
17. **const countRatings = (data) => {}**
    This code defines a JavaScript function "countRatings" that counts the occurrences of each rating in an array of data objects.
18. **const getOverallRating = (ratings) => {}**
    This code defines a JavaScript function "getOverallRating" that calculates an overall rating based on weighted values of given ratings.
19. **const getHighestRatedSim = (data) => {}**
    This code defines a JavaScript function “getHighestRatedSim” that identifies the "sim" (simulation) with the highest total rating from a dataset. It then updates an HTML element with this highest-rated "sim".
20. **const getLowRatedSim = (data) => {}**
    This code defines a JavaScript function "getLowRatedSim" that identifies the "sim" (simulation) with the highest number of ratings from a dataset. It then updates an HTML element with this most-rated "sim".
21. **const showRaing = (rating) => {}**
    This code defines a JavaScript function "showRaing" that generates HTML to visually represent a star rating.
22. **const groupAndCalculateTotalRating = (data) => {}**
    This code defines a JavaScript function "groupAndCalculateTotalRating" that groups data by a date property and calculates the total count of entries for each date. It then returns an array of these grouped data points.
23. **const ratings = (data) => {}**
    This JavaScript code defines a function "ratings" that processes a dataset of ratings, updates the DOM with summarized rating information, and generates feedback sections based on rating categories.

24. **const showAskGptKeywords = (keywords) => {}**
    This JavaScript code defines a function "showAskGptKeywords" that takes an array of keyword objects and updates the DOM with the keywords formatted in table rows.
25. **const sendReminder = (date, action) => {}**
    This JavaScript code defines a function "sendReminder" that sends a reminder request to a server at *'includes/ajax.php'* using an AJAX POST request.
    The data sent with the request includes:
    a. reminder: Set to true to indicate a reminder request.
    b. action: The action parameter passed to the function.
    c. date: The date parameter passed to the function.
26. **const defaultData = (date) => {}**
    This JavaScript function, defaultData, is responsible for handling the response received after sending a POST request to a server with specific data. This function named "defaultData" takes a date parameter, presumably for fetching data related to that specific date.

Assets Management

Template Name: assets-management.php



PHP Functions Used

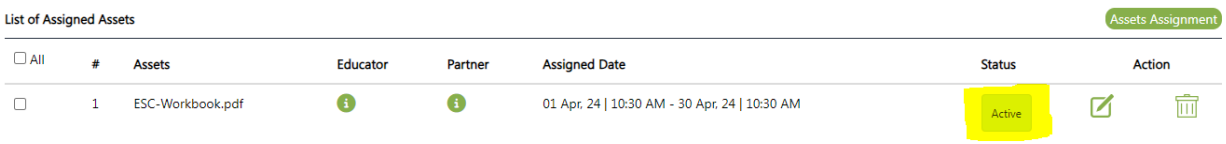
1. **List of Assigned Assets**
   *$sqlr = "SELECT *, DATE_FORMAT(start_date, '%d %b, %y | %H:%i %p') AS sdate, DATE_FORMAT(end_date, '%d %b, %y | %H:%i %p') AS edate FROM assigned_assets WHERE 1=1 ORDER BY id DESC";*
   This PHP code performs a database query to retrieve records from the "assigned_assets" table, formats the date fields, and manages pagination for displaying the results.

Third Party JS Library Used

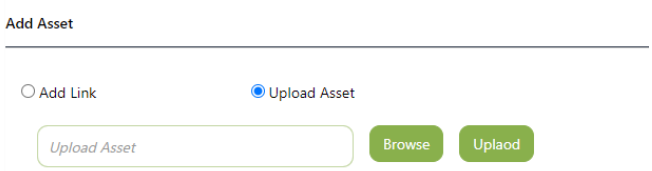
1. Bootstrap DateTimePicker
2. TooltipJS

JS Functions Used

1. **$('.form_datetime').datetimepicker({})**
   This code sets up a datetime picker for HTML elements with the class ".form_datetime".
2. **$('.checked_all').on('change', function() {})**
   This code adds an event listener to elements with the class ".checked_all", specifically listening for changes in their state (e.g., when a checkbox is checked or unchecked).
3. **$('.checkbox').change(function() {})**
   This code adds a change event listener to all elements with the class ".checkbox". It executes a function whenever the state of any of these checkboxes changes (i.e., when it is checked or unchecked).
4. **$('.assets_type').on('click', function() {})**
   This code adds a click event listener to elements with the class ".assets_type". This code dynamically shows or hides certain elements on the page and updates their required status based on the user's selection.
5. **$('.multiselect-ui').multiselect({})**
   This code initializes a multiselect dropdown menu for elements with the class ".multiselect-ui".
6. **const updateStatus = (status, id) => {}**

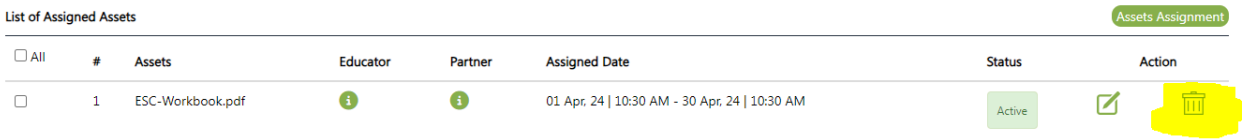


   This code defines a function "updateStatus" that is responsible for updating the status of an asset. It takes two parameters: "status" (either 0 for inactive or 1 for active) and "id" (the identifier of the asset).
7. **$.post(*'includes/ajax.php'*, { get_users: true, role: "'educator','partner'" }).done(function(response){})**
   This code snippet sends an AJAX request to fetch user data from the server based on specific roles (educator and partner). Upon success, it updates the options of dropdown elements for educators and partners.
8. **$('#upload').on('click', function (e) {})**



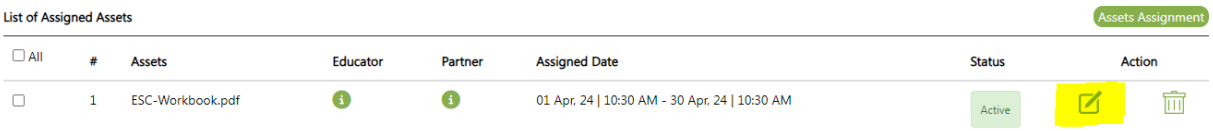
   This code handles the file upload process through AJAX which sends requests to *"includes/upload.php"*, validates the file type, and provides feedback to the user about the upload status.

9. **$('body').on('click', '.delete_assigned_assets', function (e) {})**



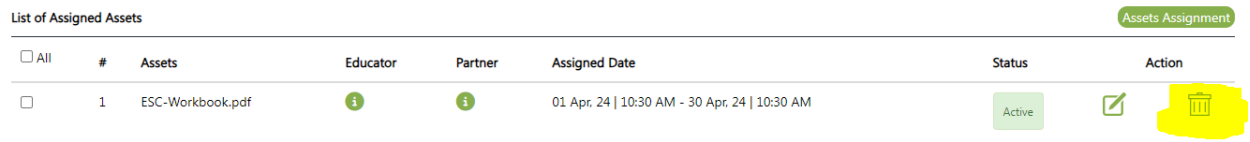
This event is triggered on click of Delete icon and send request to *'includes/delete-file.php'* with parameters “delete_page_assets” as true and file as file name.

10. **$('.edit_assets').click(function() {})**



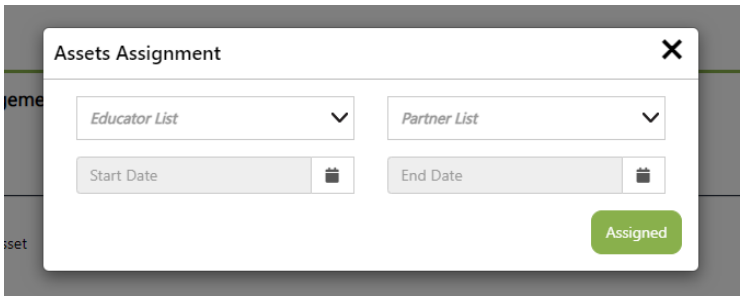
This event is triggered on click of Edit icon and send load URL “edit-assets-modal.php” with query string parameters data-id with id into modal popup.

11. **$('.delete_assets').click(function() {})**



The code handles the deletion of assets when a delete button with the class ".delete_assets" is clicked. It sends request to *“includes/process.php”* with parameters delete_assets as true and id as asset id.

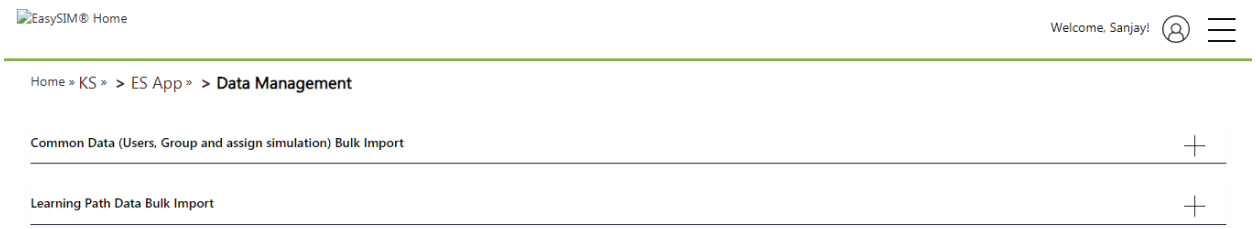
12. **$("#assigned_btn").on('click', (function(e) {})**



This code snippet handles the submission of an asset assignment form when a button with the ID "assigned_btn" is clicked. It sends POST requests to *“includes/process.php”* with Form Data for the Assets Assignment form.

Data Management

Template Name: data-management.php

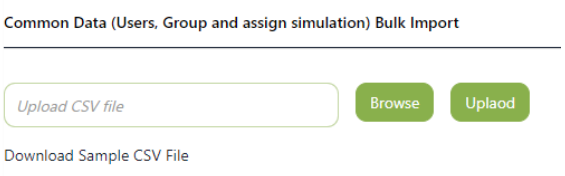


PHP Functions Used

1. **SQL Queries**
   *$sqlr = "SELECT *, REPLACE(CONCAT_WS(' ',fname,lname),' ',' ') AS fullname, DATE_FORMAT(date_of_join, '%e-%b-%y') AS dj FROM users WHERE 1=1 $norderby";*
   This query selects all columns from the "users" table, creates a concatenated "fullname" column, and formats the "date_of_join" column. It can be extended with additional conditions or ordering clauses depending on the value of "$norderby".

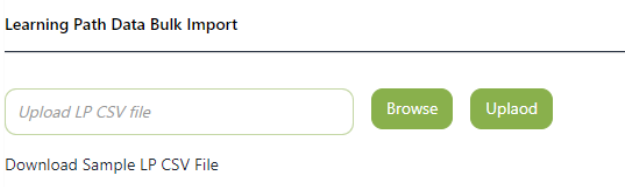
JS Functions Used

1. **$('#upload').click(function() {})**



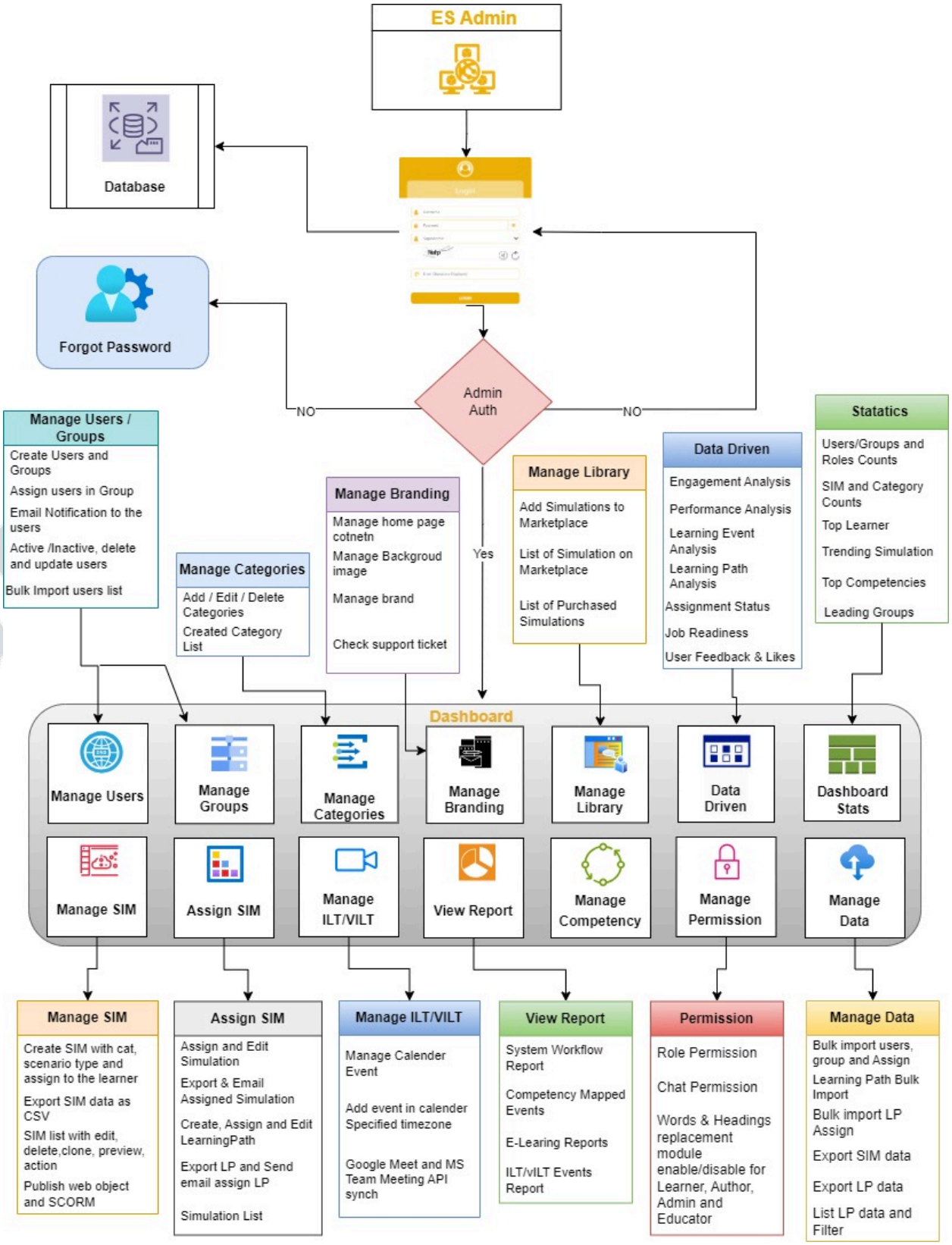
   This JavaScript code handles the file upload functionality triggered by a button click "Upload" for Common Data Bulk Upload form. It sends POST request to *"import-csv-data.php"*.

2. **$('#uploadlp').click(function() {})**



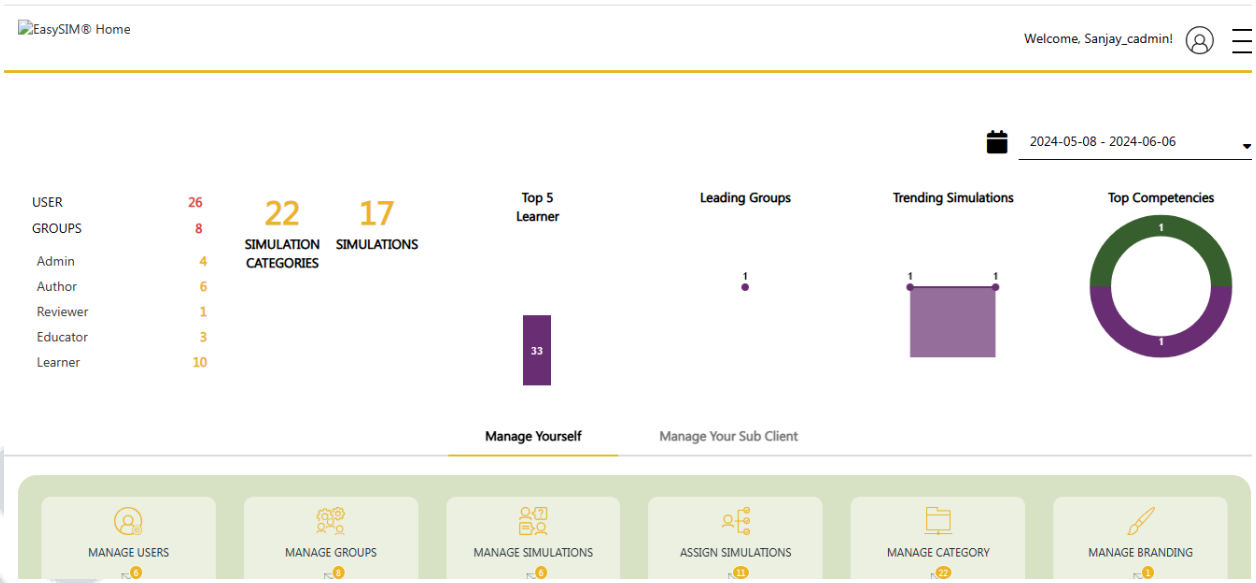
   This JavaScript code handles the file upload functionality triggered by a button click "Upload" for Learning Path Data Bulk Import form. It sends POST request to *"import-csv-data.php"*.

Admin Journey DFD



Client Admin

Overview



This domain will only be accessible to users with **ClientAdmin** credentials and will allow management of User, Group, Simulations, Category, Branding, Permission, Competency, Library, View Report, View Attention Report, and Assign Simulation. After authentication, users will be redirected to the dashboard page. The dashboard will feature static boxes containing details. Following this, users will have two options: "**Manage Yourself**" and "**Manage Your Subclients.**"

1. **Manage Yourself**
    a. Manage Yourself landing page same as **Admin role**.
2. **Manage Your Sub Client**
    a. **Add New Sub Client:-** Fill all details and then create a subclient.
    b. **View Report**

Code Stuff

Dashboard

Template Name: client-admin-dashboard.php

PHP Functions Used

1. **checkLogin()**
   Global function from "db.class.php". The checkLogin function ensures that a user is properly logged in by verifying that all essential session variables are set and not empty.
2. **checkAccountType()**
   Global function from "db.class.php". The checkAccountType function determines the type of account based on the user's role stored in the session. It returns a corresponding numerical value for each role or FALSE if the role is not recognized.
3. **getUsedHostingData($client_id)**
   Global function from "db.class.php". The getUsedHostingData function retrieves and calculates the used hosting space for a given client or all clients (excluding super admins) if no client ID is specified. It returns the total size used and detailed data points for each domain, formatted for display purposes. The function relies on several helper methods (getClientData, folderSize, and sizeFilter) to retrieve data, calculate folder sizes, and format the size output.
4. **getClientPermission($client_id)**
   Global function from "db.class.php". This function returns all the data for the permissions from table client_permissions for the give client_id.
5. **getTotalUsers('USER_TYPE')**
   Global function from "db.class.php". The getTotalUsers function retrieves the total number of users from the users table. If a specific role is provided, it filters the users by that role. The function constructs an appropriate SQL query based on whether the role parameter is given, executes the query, and returns the count of users. If no users are found, it returns 0.
6. **getTotalCategory()**
   Global function from "db.class.php". The getTotalCategory function retrieves the total number of categories from the category_tbl table.

7. **getTotalScenario()**
   Global function from "db.class.php". The getTotalScenario function retrieves the total number of scenarios from the scenario_master table.

Third Party JS Library

1. DateRangePicker
2. CanvasJS
3. OwlCarousel

JS Functions Used

1. **$('.owl-carousel').owlCarousel({})**
   This code initializes an Owl Carousel with custom settings on elements with the class owl-carousel. It specifies no looping, a 5-pixel margin between items, enabled navigation buttons, and disabled mouse dragging. Additionally, it configures responsive behavior to show a different number of items based on the viewport width.
2. **cb(start, end) {}**
   The function cb(start, end) is designed to update the content of a specific HTML element with the ID reportrange by setting the inner HTML of a span element within it to display a formatted date range.
3. **$('#reportrange').daterangepicker({})**
   This Javascript code initializes a date range picker on an input element with the ID daterange, sets its initial dates, formatting, alignment, and predefined ranges, and specifies a callback function to handle date changes.
4. **$.ajax({...url: *'includes/main-process.php'*,...})**
   This jQuery AJAX call sends a POST request to *'includes/main-process.php'* with the data 'getClientAdminDefaultData=true'. If the request is successful, it parses the JSON response. If the success property in the response is true, it updates various HTML elements with the data received. If not, or if there's an error in the request, it shows an error alert using SweetAlert.
5. **$('.multiselect-ui').multiselect({})**
   The provided jQuery script configures a multiselect dropdown using the multiselect plugin. On Change Counts selected options and disables unselected ones if the maximum count (maxCount) is reached. Sends an AJAX request to fetch client data when a selection is made, displaying a loading overlay during the request. Updates various HTML elements with the received data or shows error messages if the request fails.
6. **defaultGraphData(date){}**
   It gets data through AJAX requests to *'includes/client-admin-dash-grid-dataNew.php'* with different parameters for generating different graphs on the page. Graphs generated are Top Learners, Top Competence, Trending SIM and Leading Groups.

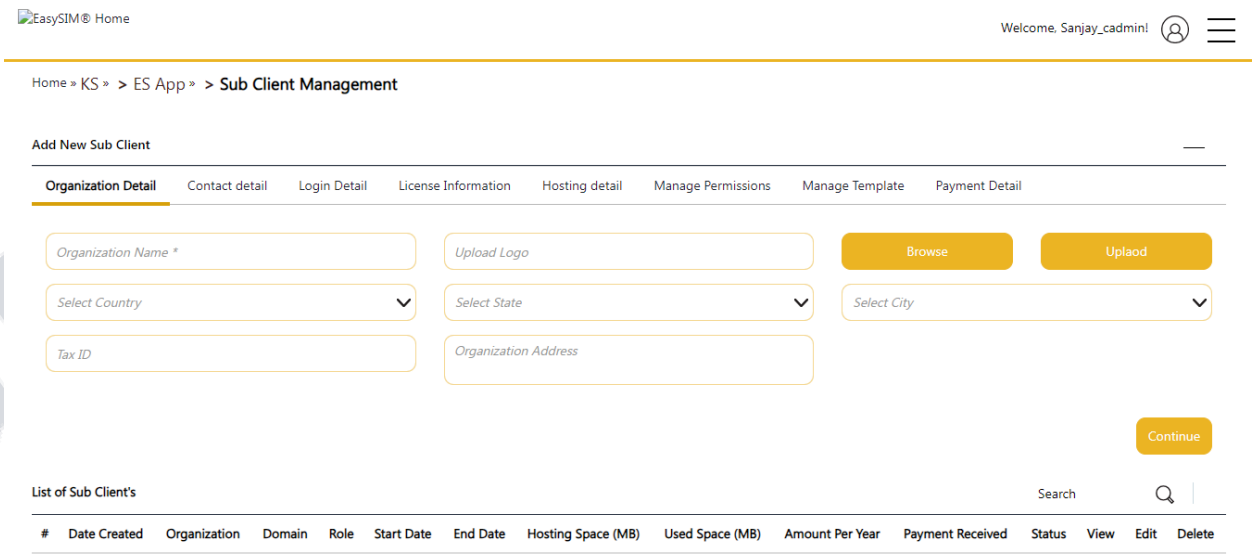
7. **UsedHostingData = new CanvasJS.Chart("UsedHostingData", {})**
   The provided JavaScript code creates and configures a doughnut chart using the CanvasJS library to visualize used hosting data.

Manage Yourself

This section has the same Coding Stuff as for Admin. Please check the Admin section for it.

Manage Your Sub Client > Manage Sub Client

Template Name: sub-client-management.php



PHP Functions Used

1. **checkLogin()**
   Global function from "db.class.php". The checkLogin function ensures that a user is properly logged in by verifying that all essential session variables are set and not empty.
2. **checkAccountType()**
   Global function from "db.class.php". The checkAccountType function determines the type of account based on the user's role stored in the session. It returns a corresponding numerical value for each role or FALSE if the role is not recognized.
3. **SQL Queries**
   $sqlr = "SELECT c., DATE_FORMAT(c.create_date_time, '%d-%m-%y %h:%i %p') AS create_date FROM mapping_tbl AS m LEFT JOIN client_tbl AS c ON m.sub_client_id = c.client_id WHERE m.client_id = ? ORDER BY c.client_id DESC";*
   This SQL query retrieves and formats data from the client_tbl table by joining it with the mapping_tbl table based on the sub_client_id and client_id fields. It selects all columns from the client_tbl table, formats the create_date_time column, filters records where

client_id matches a specific value, and orders the results by client_id in descending order.

Third Party JS Library

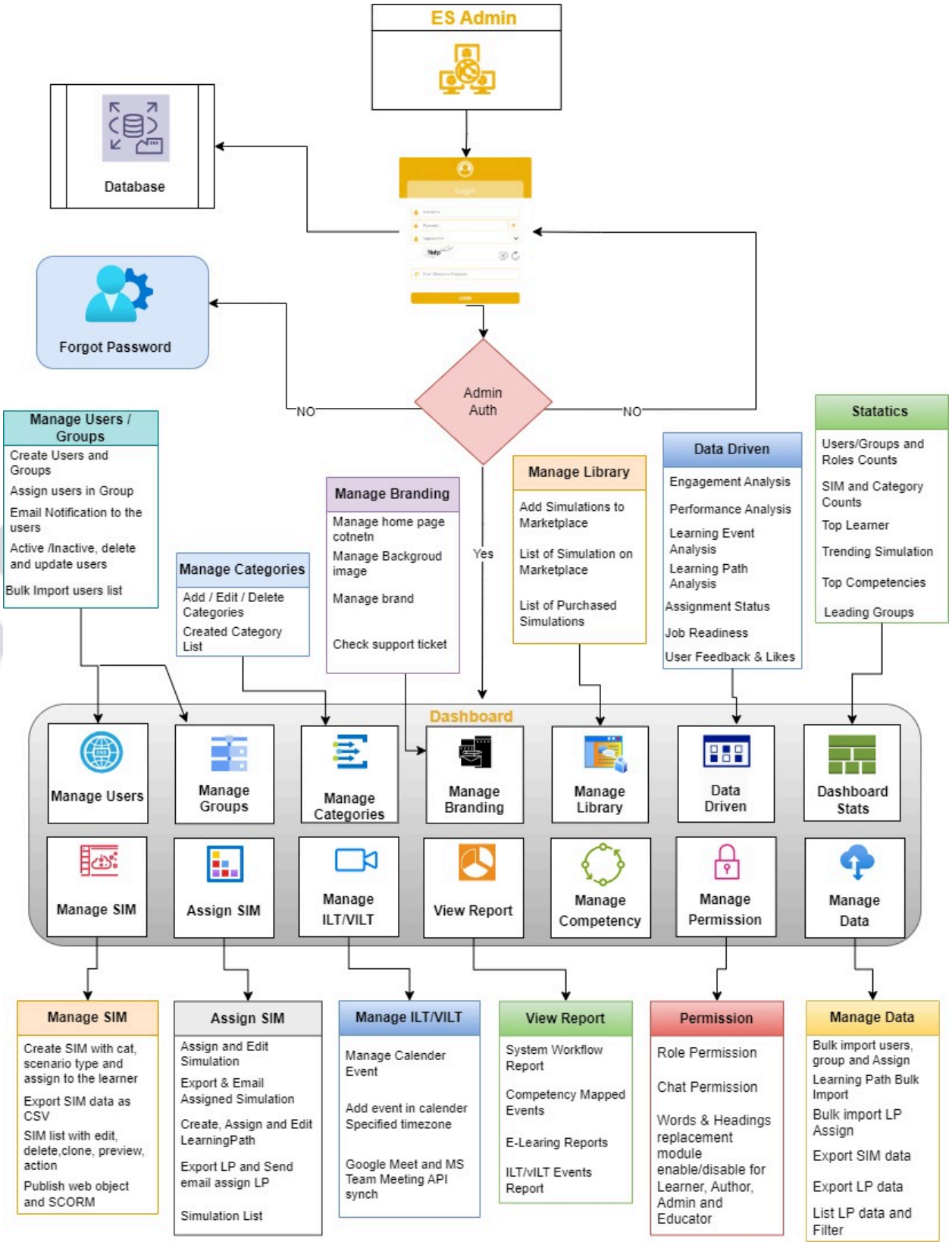
1. InputMask
2. Bootstrap-DateTimePicker

JS Functions Used

1. **$('input[name=mail_type]').on('click', function() {})**
   The jQuery code snippet sets up an event handler for click events on input elements with the name mail_type. Depending on the value of the clicked input, it toggles the visibility of certain elements and clears the values of specific input fields.
2. **$('.checked_all').on('change', function() {})**
   The jQuery code snippet sets up an event handler that synchronizes the checked state of a group of checkboxes based on the checked state of a master checkbox.
3. **$('.tmp_type').change(function() {})**
   The jQuery code sets up an event handler that monitors changes to a group of checkboxes with the class tmp_type. Depending on the state of these checkboxes, it updates the state of a master checkbox with the class checked_all.
4. **$.ajax({...url: *"includes/currencies.json"*,...})**
   This jQuery code snippet makes an AJAX request to fetch data from a JSON file (*includes/currencies.json*) containing currency information. Upon successful retrieval, it dynamically populates a <select> dropdown with the currency options.
5. **$.getJSON(*'includes/main-process.php*?type=getCountry', function(data) {})**
   This jQuery code snippet retrieves data from the server in JSON format using the $.getJSON() function. It fetches country information from a PHP script (*includes/main-process.php*) with the type parameter set to getCountry. Upon successful retrieval, it dynamically populates a <select> dropdown with country options.
6. **jQuery('body').on('change', '.country', function() {})**
   This jQuery code snippet sets up an event handler to handle changes in the selection of a <select> dropdown with the class country. When the user selects a country, it dynamically populates another dropdown with state options corresponding to the selected country.
7. **jQuery('body').on('change', '.state', function() {})**
   This jQuery code snippet sets up an event handler that handles changes in the selection of a <select> dropdown with the class state. When a state is selected, it dynamically populates another dropdown with city options corresponding to the selected state.
8. **$.getJSON(*'includes/main-process.php*?type=getCountry', function(data) {})**
   This jQuery code snippet retrieves data from the server in JSON format using the $.getJSON() function. It fetches country information from a PHP script (*includes/main-process.php*) with the type parameter set to getCountry. Upon successful retrieval, it dynamically populates a <select> dropdown with country options.

9. **$('.form_datetime').datetimepicker({})**
   The provided jQuery code configures a datetime picker on elements with the class form_datetime using the datetimepicker plugin.
10. **$('#role').multiselect({})**
    The provided jQuery code configures a multiselect dropdown menu on an HTML element with the ID role using the multiselect plugin.
11. **$('#db_name').keydown(function(event){})**
    The provided jQuery code sets up an event handler for the keydown event on an HTML element with the ID db_name.
12. **$("#domain_name").on('change', function(){})**
    This code validates the value of the input field with ID domain_name whenever it changes. If the value is not between 3 and 25 characters long, or contains spaces or special characters, the input field is cleared, and a warning message is displayed using SweetAlert. This ensures that the subdomain name adheres to the specified format.
13. **$('#upload').click(function() {})**
    This code handles the file upload process when the upload button is clicked. It disables the button and shows a loading message, gathers the file data, and sends it to the server via an AJAX POST request to *"includes/upload.php"*. Based on the server response, it updates the UI to show the uploaded image or an error message, and re-enables the upload button.
14. **$('.delete_logo').click(function() {})**
    This code handles the deletion of a logo file when a delete button is clicked. It retrieves the file name from the clicked element, displays a confirmation dialog, and if confirmed, sends an AJAX request to *"includes/delete-file.php"* to delete the file. Depending on the server response, it updates the UI and displays appropriate messages using SweetAlert. If the deletion is not confirmed, it informs the user that the file is safe.
15. **$('.delete').click(function () {})**
    This code handles the deletion of a client account when a delete button is clicked. It retrieves the client ID from the clicked element, displays a confirmation dialog, and if confirmed, sends an AJAX request to *"includes/main-process.php"* to delete the client account. Depending on the server response, it updates the UI, displays appropriate messages using SweetAlert, and reloads the page to reflect the changes. If the deletion is not confirmed, it informs the user that the client account is safe.
16. **$('.inactive_client').on('click', function(){})**
    The provided jQuery code sets up an event handler for the click event on elements with the class inactive_client. This code handles the process of marking a client account as inactive when the button is clicked through AJAX request to *'includes/main-process.php'*
17. **$('.active_client').on('click', function(){})**
    The provided jQuery code sets up an event handler for the click event on elements with the class active_client. This code handles the process of activating a client account when the button is clicked through AJAX request to *'includes/main-process.php'*..

Client Admin DFD



Learner

Overview

Learner Overview Dashboard

- All learner simulation data is represented on this dashboard, like all simulations "Syllabus Status", "Attempts Used", "Time Spent" and "Success Rate"
- Under the My Progress left section, it has three tabs.
  - My Assignments : Click on this tab showing all both attempted or incomplete simulations.
  - Attempted : Click on this tab showing all attempted simulations.
  - Incomplete: Click on this tab showing all incomplete simulations.
- After that tab has two boxes.
  - Event: Here, all events in the calendar and the event list where the admin assigns events to learners.
  - My Educator: Here, showing all educators where the admin assigns educators to learners
- After that tab has four boxes.
  - Score Result. All simulation score results are shown in this graph.
  - Attempts: All simulation attempts are shown in this graph.
  - Progress. All simulation progress is shown in this graph.
  - Productivity Rate. All simulation productivity rates are shown in this graph.

    Here there is one select and search event dropdown show. The learner can search for individual simulation and multiple simulation.

    Also, The learner can download the result by clicking on the "download result" button.
- The date of change for the simulation is shown in the dropdown box. The learner can select an individual simulation and show all the data in the right section.

- Click on "My Assignments" tab:

  It has two option:
  - Learning Paths:- All learning simulations are shown here.
  - Learning Events:- All learning simulations are shown here.
- Click on "Attempted" tab:

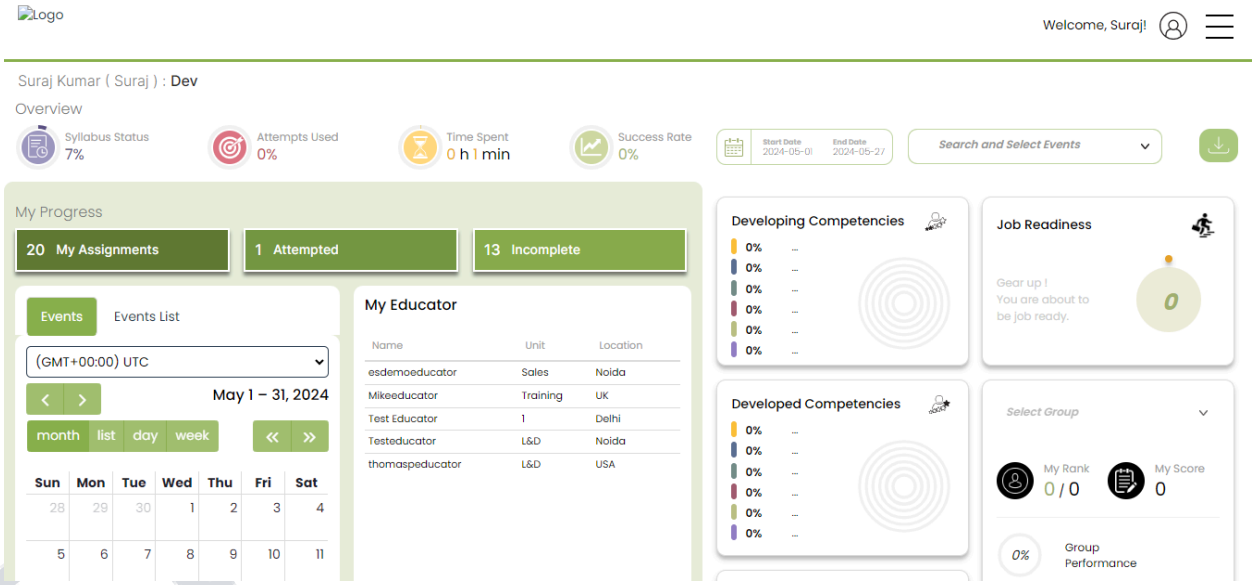
  It has two option:
  - Learning Paths:-All learning path simulations are shown here.
  - Learning Events:-All learning simulations are shown here.
- Click on "Incomplete" tab:
  - It has two option:
    - Learning Paths:-All learning path simulations are shown here.
    - Learning Events:-All learning simulations are shown here.

Easy SIM®

Code Stuff

Dashboard

Template Name: learner-overview-dashboard.php



PHP Function Used

1. **getTotalLearnerSIM($userid)**
   Global function from "db.class.php". This function will return the total number of Simulations for the user as per the ID passed as argument.
2. **getTotalAttemptedSim($userid)**
   Global function from "db.class.php". This function will return the total number of Simulations attempted by the user as per the ID passed as argument.
3. **getIncompleteLearnerSIM($userid)**
   Global function from "db.class.php". This function will return the total number of Simulations not completed by the user as per the ID passed as argument.
4. **getAssignSimData($userid)**
   Global function from "db.class.php". This function will return the total number of Simulations not completed by the user as per the ID passed as argument.
5. **getAssigneeName($userid)**
   Global function from "db.class.php". This function will return all the data for user as per the User ID passed in the argument.
6. **Events List**
   $sql = "SELECT s.scenario_id, s.Scenario_title, LEFT(s.Scenario_title, 20) AS title, c.category, LEFT(c.category, 20) AS lc FROM scenario_master AS s LEFT JOIN category_tbl AS c ON s.scenario_category = c.cat_id WHERE s.scenario_id IN ($sim_list)"

$asql = "SELECT DATE_FORMAT(a.start_date, '%e-%b-%y') AS sdate, DATE_FORMAT(a.start_date, '%l:%i %p') AS stime, DATE_FORMAT(a.end_date, '%e-%b-%y') AS edate, DATE_FORMAT(a.end_date, '%l:%i %p') AS etime, g.group_id FROM assignment_tbl a LEFT JOIN group_tbl g ON a.group_id = g.group_id WHERE FIND_IN_SET($sid, a.scenario_id) AND (FIND_IN_SET($userid, g.learner) OR FIND_IN_SET($userid, a.learner_id))"

First query selects data from *scenario_master* and *category* than in during the loop the 2nd sql query selects data as per *scenario_id* and *user_id* selected in the first query and returns the lists of events for the Users.

Third Party JS Library Used

1. CanvasJS
2. DateRangePicker
3. YearPicker
4. JSPDF
5. HTML2Canvas
6. Fancybox
7. FullCalender
8. Tooltip

JS File Used

File Path: content/js/learner-overview-dash.js

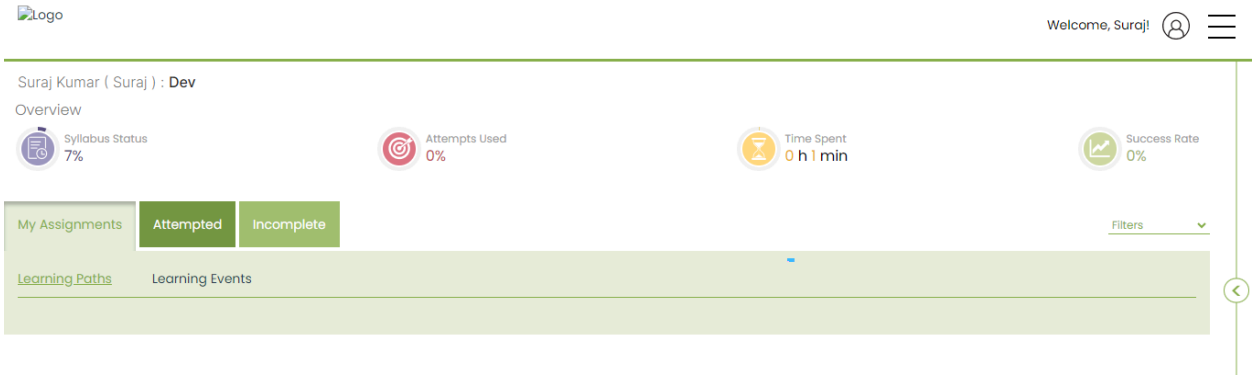
1. **const CircleBar = () => {}**
   This JavaScript code defines a function named "CircleBar" that dynamically updates the style of elements representing circular progress bars on a webpage.
2. **const calculateMinutes = (timeArray) => {}**
   This JavaScript code defines a function named "calculateMinutes" that converts a time string in the format "HH" into the total number of minutes.
3. **const getTotalSumTime = (timeArray, html = false) => {}**
   This JavaScript code defines a function named "getTotalSumTime" that takes an array of time strings and optionally returns the total time in a formatted HTML string or as a numerical value in minutes or a percentage of 10 seconds.
4. **const getPercentage = (x, y, html = false) => {}**
   This JavaScript code defines a function named "getPercentage" that calculates the percentage of x relative to y, optionally returning the result as a formatted string with a percentage sign.
5. **const progressBar = () => {}**
   This JavaScript code defines a function named "progressBar" that updates the width of progress bar elements on a webpage based on a data attribute.

6. **const parseTime = (time) => {}**
   This JavaScript code defines a function named "parseTime" that takes a time string in the format "HH:MM" and returns a formatted string displaying the hours and minutes with HTML bold tags.
7. **const overview = () => {}**
   This JavaScript code defines a function named overview that performs an AJAX POST request to retrieve learner overview data, processes the received data, updates the DOM with the calculated values, and handles any errors that may occur during the request.
8. **function cb(start, end) {}**
   This JavaScript code defines a function named "cb" that updates the values of specific input elements with formatted date strings based on the given start and end dates.
9. **$('#reportrange').daterangepicker({})**
   This Javascript code initializes a date range picker on an input element with the ID daterange, sets its initial dates, formatting, alignment, and predefined ranges, and specifies a callback function to handle date changes.
10. **$('.sim_list').multiselect({})**
    $('#sim_id2).multiselect({})
    $('#standing_groups).multiselect({})
    This JavaScript code configures a multi-select dropdown using the jQuery-based multiselect plugin for elements with the class "sim_list".
11. **const simList = (date, apply = false) => {}**
    This JavaScript code defines a function named "simList" that retrieves simulation data based on a given date, updates the options of multi-select dropdowns accordingly, and handles the loading overlay and error messages.
12. **const performanceGrowth = (dataPoint = []) => {}**
    This JavaScript code defines a function named "performanceGrowth" that generates a column chart using the CanvasJS library. This chart visualizes growth percentages against the number of attempts taken.
13. **const filterData = (date, sim_id) => {}**
    This JavaScript code defines a function named “filterData” that updates various elements and visual components on a webpage based on selected filter criteria, such as date and simulation ID.
14. **const groupsFilterData = (group_id, date, sim_id) => {}**
    This JavaScript code defines a function named "groupsFilterData" that updates various elements on a webpage based on selected group criteria, such as group ID, date, and simulation ID.
15. **const scoreResult = (pDataPoint = [], aDataPoint = []) => {}**
    const progress = (qDataPoint = [], aDataPoint = []) => {}
    const attempts = (qDataPoint = [], aDataPoint = []) => {}
    const productivityRate = (data = []) => {}
    These JavaScript code defines a function named “scoreResult”, “progress” and “attempts”, creates a line chart using the CanvasJS library to visualize respective results.

16. **const defaultGraphData = (date, sim_id = '') => {}**
    This JavaScript code defines a function named "defaultGraphData" that retrieves default graph data via an AJAX POST request to *"includes/learner/overview-dashboard-data.php"* and updates various charts with the received data.
17. **$('#reportrange').on('apply.daterangepicker', function(ev, picker) {})**
    This JavaScript code sets up an event handler for the "apply.daterangepicker" event on an HTML element with the ID "reportrange". This handler shows a loading overlay, processes the selected date range, updates a list of simulations, updates default graph data, and hides the loading overlay
18. **$("#yearpicker").yearpicker({})**
    This JavaScript code sets up a year picker widget on an HTML element with the ID "yearpicker". When the year picker is hidden, it triggers a function to show a loading overlay, make an AJAX POST request to fetch productivity data for the selected year, and update the productivity rate chart.
19. **const getUsername = (text) => {}**
    The "getUsername" function extracts a username from a given text using a regular expression. If the text does not match the expected pattern, it generates a random username.
20. **const getPDF = (id, title) => {}**
    The "getPDF" function converts a specified HTML element into a multi-page PDF document, adds a user-specific filename, and allows the user to download it. This function leverages "html2canvas" to capture the HTML element as an image and jsPDF to create and save the PDF.
21. **$.post(events, { getLearnerEvents: true }).done(function(response){})**
    This code is part of an AJAX success callback function that processes JSON response data from *"includes/learner/events.php"*, updates the count of assigned events, and dynamically appends event data to an HTML table. If the AJAX request fails, it shows an error message using Swal.fire.

My Assignments > Learning Path

Template Name: assigned-path.php



PHP Functions Used

1. **getClientPermission($client_id)**
   Global function from “db.class.php”. This function returns all the data for the permissions from table client_permissions for the give client_id.
2. **getAssigneeName((int)$userid)**
   Global function from “db.class.php”. This function will return all the data for user as per the User ID passed in the argument.
3. **getAssignPathData((int)$userid))**
   Global function from “db.class.php”. This function will return all the data for the learning path for the user.
4. *$catSql = "SELECT * FROM path_assignment_tbl WHERE path_id IN ($getPath) $corderby";*
   This SQL query retrieves data from the path_assignment_tbl table based on certain conditions. It assumes that $getPath contains a comma-separated list of path IDs (IDs of learning paths) and that $corderby specifies the ordering condition.

   $sql = "SELECT DISTINCT scenario_id, sim_ques_type, sim_temp_type, Scenario_title, LEFT(Scenario_title, 50) AS stitle, duration, image_url, video_url, audio_url, web_object_file, scenario_type, scenario_category, author_scenario_type, sim_cover_img, web_object FROM scenario_master WHERE scenario_id IN ($getSim) $orderby";
   This SQL query selects distinct rows from the "scenario_master" table. The query includes various fields and applies conditions based on scenario IDs and optional ordering.
5. **checkMandatory($sim_id)**
   Global function from “db.class.php”. This function will return all the data from assignment_tbl for the given Simulation ID.
6. **getScenarioTypes(md5($author_scenario_type))**
   Global function from “db.class.php”. This function will return all the different scenario_type for the author.
7. **getTotalAttempt($sim_id, $userid)**
   Global function from “db.class.php”. This function will return total attempts for the user and simulation passed as arguments.
8. **getAssignSimOtherData($sim_id, $userid)**
   Global function from “db.class.php”. This PHP function, "getAssignSimOtherData", is designed to retrieve assignment data related to a simulation ("$sim_id") for a particular user ("$user_id").
9. **getTotalQuestionsByScenario($en_sim_id)**
   Global function from “db.class.php”. This PHP function return total number of question for the particular simulation ID.
10. **getOpenResTotalTime($sim_id, $userid)**
    Global function from “db.class.php”. This PHP function calculates and returns the total time spent on a specific scenario attempt by a user.

11. **getLastAttemptOpenResp($sim_id, $userid)**
    Global function from “db.class.php”. This PHP function retrieves the latest attempt data for a specific scenario (scenario_id) and user (userid).
12. **getMultiTotalTime($sim_id, $userid)**
    Global function from “db.class.php”. This PHP function calculates and returns total duration of all attempts for a specific user ("userid") and scenario ("scenario_id") by summing the differences between the start and end times of each attempt.
13. **getLastAttemptMulti($sim_id, $userid)**
    Global function from “db.class.php”. This PHP function retrieves the most recent attempt made by a specific user (userid) on a particular scenario (scenario_id) where the attempt's unique identifier (uid) exists in the multi_score_tbl table.
14. **getTotalTime($sim_id, $userid)**
    Global function from “db.class.php”. This PHP function calculates and returns the total time spent on attempts for a specific scenario by a particular user.
15. **getLastAttempt($sim_id, $userid)**
    Global function from “db.class.php”. This PHP function returns information about the latest attempt made by a specific user on a scenario.
16. **getMultiTotalTime($sim_id, $userid)**
    Global function from “db.class.php”. This PHP function calculates and returns the total time difference between pairs of "startdate" and "enddate" values for a specific user ("userid") and simulation ("scenario_id").
17. **getLastAttemptELearning($sim_id, $userid)**
    Global function from “db.class.php”. This PHP function returns the latest attempt record for a specific simulation ("scenario_id") and user ("userid") from the scenario_attempt_tbl table, where the attempt ID (aid) also exists in the scorm_attempt_history table.
18. **getLastAttemptHybridMulti($sim_id, $userid)**
    Global function from “db.class.php”. This PHP function returns the latest attempt record for a specific simulation ("scenario_id") and user ("userid") from the scenario_attempt_tbl table, where the attempt's uid is present either in the multi_score_tbl or open_res_tbl tables.
19. **getLastAttemptHybrid($sim_id, $userid)**
    Global function from “db.class.php”. This PHP function returns the latest attempt record from the scenario_attempt_tbl table for a specific simulation ("scenario_id") and user ("userid"), ensuring that the attempt's uid is present in either the score_tbl or the open_res_tbl tables.
20. **getCustomerSatisfaction($user_id, $sim_id, $column, $table)**
    Global function from “db.class.php”. This PHP function returns data from specific column from a table defined previously, filtering the results based on user ($uid) and simulation ($scenario_id), and grouping the results by question_id.
21. **getTotalAttemptHybridQuesMulti($sim_id, $userid)**
    Global function from “db.class.php”. This PHP function, "getTotalAttemptHybridQuesMulti", is designed to retrieve and count the total number of attempts for hybrid questions (from both multi_score_tbl and open_res_tbl tables) for a specific simulation (sim_id) and user (user_id).

22. **get_percentage($totalQuestions, $satisfaction)**
    Global function from “db.class.php”. This function calculates the percentage of a part (number) relative to a total (total).
23. **isESimAttempted($sim_id, $userid)**
    Global function from “db.class.php”. The "isESimAttempted" function checks whether a user has attempted a specific simulation.
24. **isAttempted($sim_id, $userid)**
    Global function from “db.class.php”. The "isAttempted" function checks whether a user has attempted a specific simulation by querying a database to see if there are any relevant entries in several related tables.

Third Party JS Library

1. ChatApp
2. ChatGPT
3. OwlCarousel
4. DateRangePicker
5. CanvasJS
6. FancyBox
7. FullCalender
8. Tooltip
9. SweetAlert

JS File Used

File Path: content/js/learner-events.js

1. **const getUrlVars = () => {}**
   The "getUrlVars" function extracts query parameters from the current URL and returns them as an array where each parameter is both an item and a property of the array.
2. **const tooltip = () => {}**
   This triggers tooltip functionality on tags with attribute *data-toggle="tooltip".*
3. **const getPercentage = (x, y, html = false) => {}**
   This code defines a JavaScript function named "getPercentage". It calculates the percentage value of "x" in relation to "y", optionally formatting the result as either an integer or a string with a percentage symbol.
4. **const progressBar = () => {}**
   The “progressBar” function iterates over each element with the class progressBar, adds a span element inside it, and sets the width of the span to match the value of the data-progress attribute of the original element
5. **const CircleBar = () => {}**
   The "CircleBar" function initializes circular progress bars on elements with specific classes by setting the "stroke-dashoffset" CSS property based on a data attribute.

6. **const limit = (string = '', limit = 0) => {}**
   This JavaScript code defines a function named "limit". It truncates a given string to a specified limit and appends an ellipsis ("...") if the string exceeds the limit.
7. **const parseTime = (time) => {}**
   This JavaScript code defines a function named "parseTime". It parses a time string in the format "HH:MM" and returns a formatted string displaying the hours and minutes.
8. **const getTotalSumTime = (timeArray, html = false) => {}**
   This JavaScript code defines a function named "getTotalSumTime". It calculates the total sum of time durations from an array of time strings in the format "HH:MM" and returns the result.
9. **const calculateMinutes = (timeArray) => {}**
   This JavaScript code defines a function named "calculateMinutes". It calculates the total number of minutes from a time string in the format "HH".
10. **const getUsername = (text) => {}**
    This JavaScript code defines a function named "getUsername". It extracts a username from a text string that matches a specific pattern, or generates a random username if no match is found.
11. **const getPDF = (id, title) => {}**
    This JavaScript code defines a function named "getPDF". It generates a PDF file from the content of a specified HTML element and downloads it.
12. **function cb(start, end) {}**
    The provided code defines a function named cb that updates the values of specific input fields based on the start and end dates provided as arguments. This function is handling date ranges in a web application, using a library like jQuery and a date picker plugin.
13. **$('#reportrange').daterangepicker({})**
    This triggers a date range picker library for the selection of dates for reports generated.
14. **const setCookie = (name, value, days) => {}**
    This JavaScript code defines a function named "setCookie" used to set a cookie in the browser.
15. **const getCookie = (name) => {}**
    This JavaScript code defines a function named "getCookie" used to retrieve the value of a cookie by its name from the browser's cookies.
16. **$('.launch_sim').on('click', function(e){})**
    This jQuery code snippet sets up an event listener for clicks on elements with the class ".launch_sim". When clicked, it retrieves the value of a custom attribute "sim-id", stores it in a cookie named "'sim_id'", and sets the cookie to expire in 7 days.
17. **$('.sim_list').multiselect({})**
    $('#standing_groups').multiselect({})
    This code initializes a multiselect dropdown menu for elements for the specific class or ID as selector.
18. **const overview = () => {}**
    This JavaScript function overview is responsible for retrieving learner overview data via an AJAX POST request to *'includes/learner/overview-dashboard-data.php'* and updating various elements on the webpage with the received data.

19. **const performanceGrowth = (dataPoint = []) => {}**
    This JavaScript function, "performanceGrowth", is responsible for rendering a column chart using the CanvasJS library. It visualizes the growth percentage over attempts taken based on the provided data points.
20. **const groupsFilterData = (group_id, date, sim_id) => {}**
    This JavaScript function, "groupsFilterData", is responsible for filtering and updating data based on group, date, and simulation ID. It utilizes AJAX POST requests to retrieve data from the server and updates specific elements on the webpage accordingly.
21. **const filterData = (date, sim_id) => {}**
    The "filterData" function is responsible for filtering and updating various elements on a webpage based on the provided date and simulation ID. It makes use of AJAX POST requests to retrieve data from the server and dynamically updates the HTML content of different elements accordingly.
22. **const simList = (date, id, apply = false, clicked = false) => {}**
    The "simList" function is responsible for retrieving and populating a list of simulations based on the provided date and course ID. It utilizes AJAX GET requests to *'includes/learner/overview-dashboard-data.php'* to fetch data from the server and dynamically updates the options of a multiselect dropdown element accordingly.

File Path: content/js/scorm.js

1. **function setupScormApi() {}**
   The "setupScormApi" function is responsible for setting up the SCORM API functionality. It initializes SCORM communication and handles launching SCORM content within a popup window.
2. **function LMSInitialize(initializeInput) {}**
   The "LMSInitialize" function is a part of the SCORM (Sharable Content Object Reference Model) API. It is called to initialize communication between the SCORM content and the SCORM runtime environment (usually a learning management system or LMS).
3. **function LMSGetValue(varname) {}**
   The "LMSGetValue" function is another essential part of the SCORM API. It's called to retrieve the value of a specific data model element (variable) from the SCORM runtime environment (typically an LMS).
4. **function convertTimeString(timeString) {}**
   The "convertTimeString" function is designed to convert a time string into a standardized format of "HH:MM".
5. **function setStatusAndResult() {}**
   The "setStatusAndResult" function is responsible for setting the status and result of a SCORM lesson based on various conditions.
6. **function saveOrUpdate() {}**
   The "saveOrUpdate" function is responsible for saving or updating the SCORM attempt data based on certain conditions.

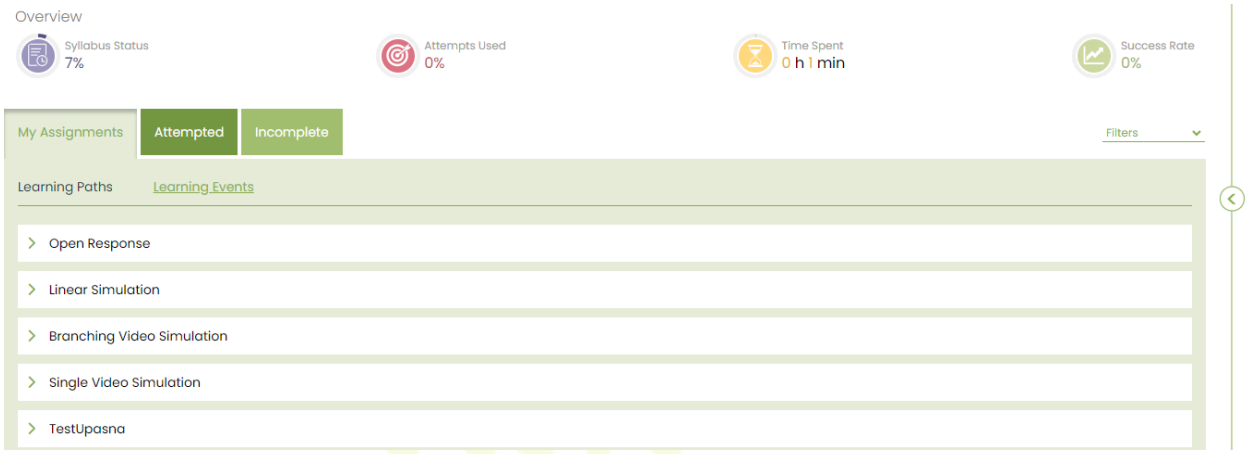
7. **function resetStatusVars() {}**
   The "resetStatusVars" function is used to reset the status-related variables to their initial values.
8. **function LMSSetValue(varname, varvalue) {}**
   The "LMSSetValue" function is responsible for setting values to SCORM data model elements based on the provided varname and varvalue parameters.
9. **function LMSFinish(finishInput) {}**
   The "LMSFinish" function is responsible for finalizing the SCORM session when the learner finishes interacting with the content.
10. **const base_url = (page = '') => {}**
    The "base_url" function is used to generate the base URL of the current page, optionally appending a specific page or path to it.
11. **const getUid = (sid) => {}**
    The "getUid" function sends AJAX request to *'includes/ajax.php'* with parameter user ID (uid) from the server and retrieves data, then creates an attempt entry in the "scenario_attempt_tbl" table with this "uid" by sending AJAX request to *'api-process.php'*, a given scenario ID (sid), and a user ID (userid). If successful, it sets the attempt ID (aid) and calls another function to handle additional actions.
12. **function getLaunchURL(path) {}**
    The "getLaunchURL" function retrieves the launch URL from an IMS manifest file (*imsmanifest.xml*) located at a given path. This function returns a promise that resolves with the full launch URL.
13. **function getOrgName(path, callback) {}**
    The "getOrgName" function retrieves the organization name and schema version from an IMS manifest file (*imsmanifest.xml*) located at a specified path. This function uses asynchronous operations (await and fetch) to handle the network request and parsing of the XML document.
14. **function msToTime(s) {}**
    The "msToTime" function converts milliseconds to a formatted time string in the format "HH:MM".
15. **function update_attempt_tbl() {}**
    The "update_attempt_tbl" function is responsible for updating the attempt table with relevant information.
16. **function toHoursAndMinutes(totalSeconds) {}**
    The "toHoursAndMinutes" function is used to convert a total number of seconds into hours, minutes, and seconds.
17. **function insert_attempt_tbl() {}**
    The "insert_attempt_tbl" function is responsible for inserting a new record into the attempt table.

File Path: content/js/e-learning.js

1. **const getTimestamp = (duration) => {}**
   The "getTimestamp" function converts a duration in seconds into a formatted timestamp string in the format mm:ss.
2. **const getBaseUrl = (page = '') => {}**
   The "getBaseUrl" function generates the base URL for a web page, optionally appending a specific page path.
3. **const getUID = (sid, userId) => {}**
   The "getUID" function is used to retrieve the unique identifier (UID) associated with a user by sending AJAX request to 'includes/ajax.php' with parameters "{ get_uid_check_attempts: true, sim_id: sid, uid: userId }" and create a new attempt record in the database by posting data through AJAX request to 'api-process.php' for a scenario.
4. **const getFileNameFromUrl = (url) => {}**
   The "getFileNameFromUrl" function is used to extract the file name from a given URL.
5. **const saveAttempt = (ev_aid, file, durationTime, currentTime, userId, type) => {}**
   The saveAttempt function is used to save attempt history related to e-learning activities by sending data through AJAX request to 'api-process.php'.
6. **$('.el_launch_mp4').on('click', function() {**
   This code handles the launching of MP4 videos, displays them in a modal, tracks user interactions, and saves relevant data about the user's interaction with the video.
7. **const videoModal = (preview, userId, simId, src) => {}**
   This function "videoModal" is responsible for dynamically generating and displaying a modal containing an HTML video player.
8. **$('body').on('click', '.el_video_close', function() {})**
   This listener ensures that when the close button within the video modal is clicked, it handles the necessary actions such as saving the attempt details and closing the modal.
9. **const pdfModal = (preview, userId, simId, src) => {}**
   This function sets up a modal window to display PDF documents with options for closing the modal and handling user interactions.
10. **$('.el_launch_pdf').on('click', function() {})**
    This event handler sets up functionality to open a PDF document in a modal window when the corresponding element is clicked.
11. **$('body').on('click', '.el_pdf_close', function() {})**
    This event handler provides functionality to close the PDF modal window and save the attempt details if necessary.

My Assignments > Learning Events

Template Name: assigned-events.php



PHP Functions Used

1. **getAssigneeName($userid)**
   Global function from "db.class.php". This function will return all the data for user as per the User ID passed in the argument.
2. **getAssignSimData($userid)**
   Global function from "db.class.php". This function will return the total number of Simulations not completed by the user as per the ID passed as argument.
3. **Learning Events List**
   $catSql = "SELECT * FROM category_tbl WHERE category != '' AND (cat_id IN (SELECT DISTINCT scenario_category FROM scenario_master WHERE scenario_id IN ($getSim)) OR cat_id IN (SELECT DISTINCT category FROM events AS ev LEFT JOIN group_tbl AS g1 ON FIND_IN_SET(g1.group_id, ev.groups) LEFT JOIN group_tbl AS g2 ON FIND_IN_SET($userid, g1.learner) WHERE FIND_IN_SET($userid, g2.learner) AND FIND_IN_SET($userid, g1.learner))) $corderby";
   This SQL query selects all rows from the category_tbl table where the category column is not empty (category != ''). It includes categories that meet either of the following conditions:
   a. The cat_id is found in the result of a subquery that selects distinct scenario_category values from the scenario_master table where the scenario_id is in a list of IDs provided by the variable $getSim.
   b. The cat_id is found in the result of a subquery that selects distinct category values from the events table, joining with the group_tbl table twice (g1 and g2). It selects events associated with groups where the user ID ($userid) is found in the learner field of both g1 and g2.
4. **Learning Events List SubQuery**
   $sql = "SELECT DISTINCT scenario_id, sim_ques_type, sim_temp_type, Scenario_title, LEFT(Scenario_title, 50) AS stitle, duration, image_url, video_url, audio_url, web_object_file, scenario_type, scenario_category, author_scenario_type, sim_cover_img, web_object FROM scenario_master WHERE scenario_category = ? AND scenario_id IN ($getSim) $orderby";

This SQL query retrieves distinct rows from the scenario_master table based on the following criteria:

Columns Selected:

a. scenario_id: The unique identifier for each scenario.
b. sim_ques_type: The type of questions associated with the scenario.
c. sim_temp_type: The template type of the scenario.
d. Scenario_title: The title of the scenario.
e. LEFT(Scenario_title, 50) AS stitle: The first 50 characters of the scenario title (truncated if longer).
f. duration: The duration of the scenario.
g. image_url: The URL of the image associated with the scenario.
h. video_url: The URL of the video associated with the scenario.
i. audio_url: The URL of the audio associated with the scenario.
j. web_object_file: The file associated with the web object.
k. scenario_type: The type of scenario.
l. scenario_category: The category of the scenario.
m. author_scenario_type: The type of scenario authored by the author.
n. sim_cover_img: The cover image associated with the scenario.
o. web_object: The web object associated with the scenario.

5. **checkMandatory($sim_id)**
   Global function from “db.class.php”. This function will return all the data from assignment_tbl for the given Simulation ID.
6. **getScenarioTypes(md5($author_scenario_type))**
   Global function from “db.class.php”. This function will return all the different scenario_type for the author.
7. **getTotalAttempt($sim_id, $userid)**
   Global function from “db.class.php”. This function will return total attempts for the user and simulation passed as arguments
8. **getAssignSimOtherData($sim_id, $userid)**
   Global function from “db.class.php”. This PHP function, "getAssignSimOtherData", is designed to retrieve assignment data related to a simulation ("$sim_id") for a particular user ("$user_id").
9. **getTotalQuestionsByScenario($en_sim_id)**
   Global function from “db.class.php”. This PHP function return total number of question for the particular simulation ID.
10. **getOpenResTotalTime($sim_id, $userid)**
    Global function from “db.class.php”. This PHP function calculates and returns the total time spent on a specific scenario attempt by a user.
11. **getLastAttemptOpenResp($sim_id, $userid)**
    Global function from “db.class.php”. This PHP function retrieves the latest attempt data for a specific scenario (scenario_id) and user (userid).

12. **getMultiTotalTime($sim_id, $userid)**
    Global function from “db.class.php”. This PHP function calculates and returns total duration of all attempts for a specific user ("userid") and scenario ("scenario_id") by summing the differences between the start and end times of each attempt.
13. **getLastAttemptMulti($sim_id, $userid)**
    Global function from “db.class.php”. This PHP function retrieves the most recent attempt made by a specific user (userid) on a particular scenario (scenario_id) where the attempt's unique identifier (uid) exists in the multi_score_tbl table.
14. **getTotalTime($sim_id, $userid)**
    Global function from “db.class.php”. This PHP function calculates and returns the total time spent on attempts for a specific scenario by a particular user.
15. **getLastAttempt($sim_id, $userid)**
    Global function from “db.class.php”. This PHP function returns information about the latest attempt made by a specific user on a scenario.
16. **getMultiTotalTime($sim_id, $userid)**
    Global function from “db.class.php”. This PHP function calculates and returns total duration of all attempts for a specific user ("userid") and scenario ("scenario_id") by summing the differences between the start and end times of each attempt.
17. **getLastAttemptELearning($sim_id, $userid)**
    Global function from “db.class.php”. This PHP function returns the latest attempt record for a specific simulation ("scenario_id") and user ("userid") from the scenario_attempt_tbl table, where the attempt ID (aid) also exists in the scorm_attempt_history table.
18. **getLastAttemptHybrid($sim_id, $userid)**
    Global function from “db.class.php”. This PHP function returns the latest attempt record from the scenario_attempt_tbl table for a specific simulation ("scenario_id") and user ("userid"), ensuring that the attempt's uid is present in either the score_tbl or the open_res_tbl tables.
19. **getCustomerSatisfaction($user_id, $sim_id, $column, $table)**
    Global function from “db.class.php”. This PHP function returns data from specific column from a table defined previously, filtering the results based on user ($uid) and simulation ($scenario_id), and grouping the results by question_id.
20. **getBranchingSatisfaction($user_id, $sim_id, $column, $table)**
    Global function from “db.class.php”. This PHP function selects and returns the choice_option_id from a specified table. The table name is represented by the variable $table.
21. **getTotalAttemptHybridQuesMulti($sim_id, $userid)**
    Global function from “db.class.php”. This PHP function, "getTotalAttemptHybridQuesMulti", is designed to retrieve and count the total number of attempts for hybrid questions (from both multi_score_tbl and open_res_tbl tables) for a specific simulation (sim_id) and user (user_id).
22. **get_percentage($totalQuestions, $satisfaction)**
    Global function from “db.class.php”. This function calculates the percentage of a part (number) relative to a total (total).

23. **isESimAttempted($sim_id, $userid)**
    Global function from “db.class.php”. The "isESimAttempted" function checks whether a user has attempted a specific simulation.
24. **isAttempted($sim_id, $userid)**
    Global function from “db.class.php”. The "isAttempted" function checks whether a user has attempted a specific simulation by querying a database to see if there are any relevant entries in several related tables.

Third Party JS Library

1. ChatApp
2. ChatGPT
3. OwlCarousel
4. DateRangePicker
5. CanvasJS
6. FancyBox
7. FullCalender
8. Tooltip
9. SweetAlert

JS File Used

File Path: content/js/learner-events.js

1. **const getUrlVars = () => {}**
   The "getUrlVars" function extracts query parameters from the current URL and returns them as an array where each parameter is both an item and a property of the array.
2. **const tooltip = () => {}**
   This triggers tooltip functionality on tags with attribute *data-toggle="tooltip".*
3. **const getPercentage = (x, y, html = false) => {}**
   This code defines a JavaScript function named "getPercentage". It calculates the percentage value of "x" in relation to "y", optionally formatting the result as either an integer or a string with a percentage symbol.
4. **const progressBar = () => {}**
   The “progressBar” function iterates over each element with the class progressBar, adds a span element inside it, and sets the width of the span to match the value of the data-progress attribute of the original element
5. **const CircleBar = () => {}**
   The "CircleBar" function initializes circular progress bars on elements with specific classes by setting the "stroke-dashoffset" CSS property based on a data attribute.
6. **const limit = (string = '', limit = 0) => {}**
   This JavaScript code defines a function named "limit". It truncates a given string to a specified limit and appends an ellipsis ("...") if the string exceeds the limit.

7. **const parseTime = (time) => {}**
   This JavaScript code defines a function named "parseTime". It parses a time string in the format "HH:MM" and returns a formatted string displaying the hours and minutes.
8. **const getTotalSumTime = (timeArray, html = false) => {}**
   This JavaScript code defines a function named "getTotalSumTime". It calculates the total sum of time durations from an array of time strings in the format "HH:MM" and returns the result.
9. **const calculateMinutes = (timeArray) => {}**
   This JavaScript code defines a function named "calculateMinutes". It calculates the total number of minutes from a time string in the format "HH".
10. **const getUsername = (text) => {}**
    This JavaScript code defines a function named "getUsername". It extracts a username from a text string that matches a specific pattern, or generates a random username if no match is found.
11. **const getPDF = (id, title) => {}**
    This JavaScript code defines a function named "getPDF". It generates a PDF file from the content of a specified HTML element and downloads it.
12. **function cb(start, end) {}**
    The provided code defines a function named cb that updates the values of specific input fields based on the start and end dates provided as arguments. This function is handling date ranges in a web application, using a library like jQuery and a date picker plugin.
13. **$('#reportrange').daterangepicker({})**
    This triggers date range picker library for the selection of dates for reports generated.
14. **const setCookie = (name, value, days) => {}**
    This JavaScript code defines a function named "setCookie" used to set a cookie in the browser.
15. **const getCookie = (name) => {}**
    This JavaScript code defines a function named "getCookie" used to retrieve the value of a cookie by its name from the browser's cookies.
16. **$('.launch_sim').on('click', function(e){})**
    This jQuery code snippet sets up an event listener for clicks on elements with the class ".launch_sim". When clicked, it retrieves the value of a custom attribute "sim-id", stores it in a cookie named "'sim_id'", and sets the cookie to expire in 7 days.
17. **$('.sim_list').multiselect({})**
    $('#standing_groups').multiselect({})
    This code initializes a multiselect dropdown menu for elements for the specific class or ID as selector.
18. **const overview = () => {}**
    This JavaScript function overview is responsible for retrieving learner overview data via an AJAX POST request to *'includes/learner/overview-dashboard-data.php'* and updating various elements on the webpage with the received data.
19. **const performanceGrowth = (dataPoint = []) => {}**
    This JavaScript function, "performanceGrowth", is responsible for rendering a column

chart using the CanvasJS library. It visualizes the growth percentage over attempts taken based on the provided data points.

20. **const groupsFilterData = (group_id, date, sim_id) => {}**
    This JavaScript function, "groupsFilterData", is responsible for filtering and updating data based on group, date, and simulation ID. It utilizes AJAX POST requests to retrieve data from the server and updates specific elements on the webpage accordingly.
21. **const filterData = (date, sim_id) => {}**
    The "filterData" function is responsible for filtering and updating various elements on a webpage based on the provided date and simulation ID. It makes use of AJAX POST requests to retrieve data from the server and dynamically updates the HTML content of different elements accordingly.
22. **const simList = (date, id, apply = false, clicked = false) => {}**
    The "simList" function is responsible for retrieving and populating a list of simulations based on the provided date and course ID. It utilizes AJAX GET requests to *'includes/learner/overview-dashboard-data.php'* to fetch data from the server and dynamically updates the options of a multiselect dropdown element accordingly.

File Path: content/js/scorm.js

1. **function setupScormApi() {}**
   The "setupScormApi" function is responsible for setting up the SCORM API functionality. It initializes SCORM communication and handles launching SCORM content within a popup window.
2. **function LMSInitialize(initializeInput) {}**
   The "LMSInitialize" function is a part of the SCORM (Sharable Content Object Reference Model) API. It is called to initialize communication between the SCORM content and the SCORM runtime environment (usually a learning management system or LMS).
3. **function LMSGetValue(varname) {}**
   The "LMSGetValue" function is another essential part of the SCORM API. It's called to retrieve the value of a specific data model element (variable) from the SCORM runtime environment (typically an LMS).
4. **function convertTimeString(timeString) {}**
   The "convertTimeString" function is designed to convert a time string into a standardized format of "HH:MM".
5. **function setStatusAndResult() {}**
   The "setStatusAndResult" function is responsible for setting the status and result of a SCORM lesson based on various conditions.
6. **function saveOrUpdate() {}**
   The "saveOrUpdate" function is responsible for saving or updating the SCORM attempt data based on certain conditions.

7. **function resetStatusVars() {}**
   The "resetStatusVars" function is used to reset the status-related variables to their initial values.
8. **function LMSSetValue(varname, varvalue) {}**
   The "LMSSetValue" function is responsible for setting values to SCORM data model elements based on the provided varname and varvalue parameters.
9. **function LMSFinish(finishInput) {}**
   The "LMSFinish" function is responsible for finalizing the SCORM session when the learner finishes interacting with the content.
10. **const base_url = (page = '') => {}**
    The "base_url" function is used to generate the base URL of the current page, optionally appending a specific page or path to it.
11. **const getUid = (sid) => {}**
    The "getUid" function sends AJAX request to *'includes/ajax.php'* with parameter user ID (uid) from the server and retrieves data, then creates an attempt entry in the "scenario_attempt_tbl" table with this "uid" by sending AJAX request to 'api-process.php', a given scenario ID (sid), and a user ID (userid). If successful, it sets the attempt ID (aid) and calls another function to handle additional actions.
12. **function getLaunchURL(path) {}**
    The "getLaunchURL" function retrieves the launch URL from an IMS manifest file (*imsmanifest.xml*) located at a given path. This function returns a promise that resolves with the full launch URL.
13. **function getOrgName(path, callback) {}**
    The "getOrgName" function retrieves the organization name and schema version from an IMS manifest file (*imsmanifest.xml*) located at a specified path. This function uses asynchronous operations (await and fetch) to handle the network request and parsing of the XML document.
14. **function msToTime(s) {}**
    The "msToTime" function converts milliseconds to a formatted time string in the format "HH:MM".
15. **function update_attempt_tbl() {}**
    The "update_attempt_tbl" function is responsible for updating the attempt table with relevant information.
16. **function toHoursAndMinutes(totalSeconds) {}**
    The "toHoursAndMinutes" function is used to convert a total number of seconds into hours, minutes, and seconds.
17. **function insert_attempt_tbl() {}**
    The "insert_attempt_tbl" function is responsible for inserting a new record into the attempt table.

File Path: content/js/e-learning.js

1. **const getTimestamp = (duration) => {}**
   The "getTimestamp" function converts a duration in seconds into a formatted timestamp string in the format mm:ss.
2. **const getBaseUrl = (page = '') => {}**
   The "getBaseUrl" function generates the base URL for a web page, optionally appending a specific page path.
3. **const getUID = (sid, userId) => {}**
   The "getUID" function is used to retrieve the unique identifier (UID) associated with a user by sending AJAX request to *'includes/ajax.php'* with parameters "{ get_uid_check_attempts: true, sim_id: sid, uid: userId }" and create a new attempt record in the database by posting data through AJAX request to 'api-process.php' for a scenario.
4. **const getFileNameFromUrl = (url) => {}**
   The "getFileNameFromUrl" function is used to extract the file name from a given URL.
5. **const saveAttempt = (ev_aid, file, durationTime, currentTime, userId, type) => {}**
   The saveAttempt function is used to save attempt history related to e-learning activities by sending data through AJAX request to *'api-process.php'*.
6. **$('.el_launch_mp4').on('click', function() {**
   This code handles the launching of MP4 videos, displays them in a modal, tracks user interactions, and saves relevant data about the user's interaction with the video.
7. **const videoModal = (preview, userId, simId, src) => {}**
   This function "videoModal" is responsible for dynamically generating and displaying a modal containing an HTML video player.
8. **$('body').on('click', '.el_video_close', function() {})**
   This listener ensures that when the close button within the video modal is clicked, it handles the necessary actions such as saving the attempt details and closing the modal.
9. **const pdfModal = (preview, userId, simId, src) => {}**
   This function sets up a modal window to display PDF documents with options for closing the modal and handling user interactions.
10. **$('.el_launch_pdf').on('click', function() {})**
    This event handler sets up functionality to open a PDF document in a modal window when the corresponding element is clicked.
11. **$('body').on('click', '.el_pdf_close', function() {})**
    This event handler provides functionality to close the PDF modal window and save the attempt details if necessary.

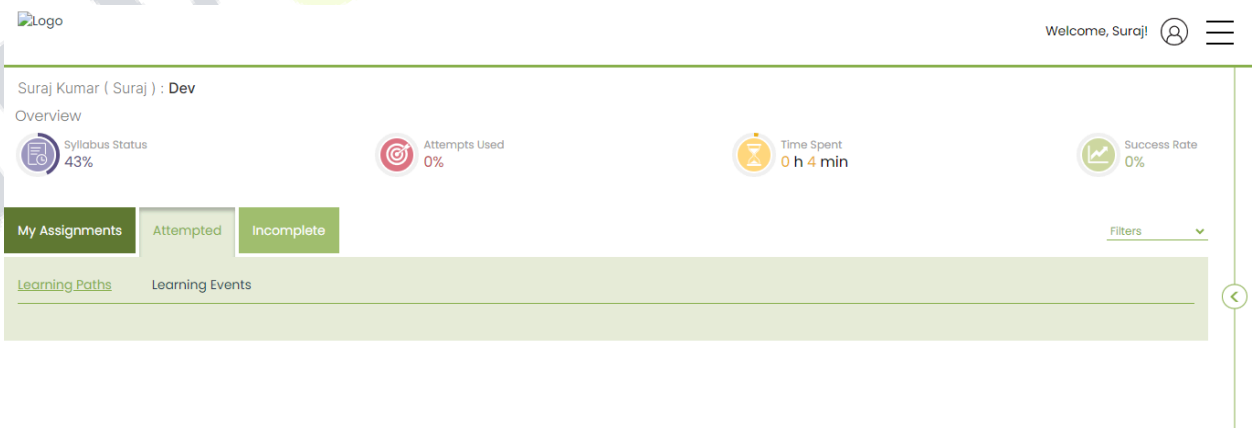
File Path: content/js/learner-assigned-events.js

1. **const getEvents = () => {}**
   This function "getEvents" is designed to retrieve events data from the server. It sends an AJAX POST request to a specified URL (*'includes/learner/events.php'*) with the intention of retrieving events categorized by category.

2. **const textLimit = (string = '', limit = 0) => {}**
   The "textLimit" function is designed to limit the length of a given string to a specified number of characters and append ellipsis (...) if the string exceeds the limit.
3. **const genrateItems = (data, key) => {}**
   The "genrateItems" function generates HTML elements based on the provided data and appends them to the DOM.
4. **const genrateItemsNew = (data, key, id) => {}**
   The "genrateItemsNew" function is similar to genrateItems, but it includes additional functionality to remove an empty event element before appending new items.
5. **$('body').on('click', '.view_event_attachment', function(e){})**
   This code is an event handler for opening a FancyBox modal with an iframe.
6. **$('body').on('click', '.attended', function(e){})**
   This JavaScript code is an event handler for handling the click event on elements with the class .attended.

Attempted > Learning Path

Template Name: attempted-path.php



PHP Functions Used

1. **getAssigneeName($userid)**
   Global function from "db.class.php". This function will return all the data for user as per the User ID passed in the argument.
2. **getAssigneeName($userid)**
   Global function from "db.class.php". This function will return all the data for user as per the User ID passed in the argument.
3. **getAssignPathData((int)$userid))**
   Global function from "db.class.php". This function will return all the data for the learning path for the user.
4. **SQL Queries**
   $catSql = "SELECT * FROM path_assignment_tbl WHERE path_id IN ($getPath) $corderby";

This SQL query retrieves records from the "path_assignment_tbl" table where the "path_id" matches any of the values specified in the $getPath variable, and optionally applies additional conditions or sorting specified by the $corderby variable.

$sql = "SELECT DISTINCT s.scenario_id, s.sim_ques_type, s.sim_temp_type, s.Scenario_title, LEFT(s.Scenario_title, 50) AS stitle, s.duration, s.image_url, s.video_url, s.audio_url, s.web_object_file, s.scenario_type, s.scenario_category, s.author_scenario_type, s.sim_cover_img, s.web_object FROM scenario_master AS s LEFT JOIN scenario_attempt_tbl AS st ON s.scenario_id = st.scenario_id WHERE s.scenario_id IN ($getSim) AND st.userid = ? AND (st.aid IN (SELECT aid FROM score_tbl) OR st.aid IN (SELECT aid FROM multi_score_tbl) OR st.aid IN (SELECT aid FROM open_res_tbl) OR st.aid IN (SELECT aid FROM scorm_attempt_history))";
This SQL query retrieves distinct records from the "scenario_master" table where the "scenario_id" matches any of the values specified in the $getSim variable, and where there are associated records in the "scenario_attempt_tbl" table for the specified user ID and with matching "aid" values in one of the specified related tables

5. **checkMandatory($sim_id)**
   Global function from “db.class.php”. This function will return all the data from assignment_tbl for the given Simulation ID.
6. **getScenarioTypes(md5($author_scenario_type))**
   Global function from “db.class.php”. This function will return all the different scenario_type for the author.
7. **getTotalAttempt($sim_id, $userid)**
   Global function from “db.class.php”. This function will return total attempts for the user and simulation passed as arguments.
8. **getAssignSimOtherData($sim_id, $userid)**
   Global function from “db.class.php”. This PHP function, "getAssignSimOtherData", is designed to retrieve assignment data related to a simulation ("$sim_id") for a particular user ("$user_id").
9. **getTotalQuestionsByScenario($en_sim_id)**
   Global function from “db.class.php”. This PHP function return total number of question for the particular simulation ID.
10. **getOpenResTotalTime($sim_id, $userid)**
    Global function from “db.class.php”. This PHP function calculates and returns the total time spent on a specific scenario attempt by a user.
11. **getLastAttemptOpenResp($sim_id, $userid)**
    Global function from “db.class.php”. This PHP function retrieves the latest attempt data for a specific scenario (scenario_id) and user (userid).
12. **getMultiTotalTime($sim_id, $userid)**
    Global function from “db.class.php”. This PHP function calculates and returns total duration of all attempts for a specific user ("userid") and scenario ("scenario_id") by summing the differences between the start and end times of each attempt.
13. **getLastAttemptMulti($sim_id, $userid)**
    Global function from “db.class.php”. This PHP function retrieves the most recent attempt

made by a specific user (userid) on a particular scenario (scenario_id) where the attempt's unique identifier (uid) exists in the multi_score_tbl table.

14. **getTotalTime($sim_id, $userid)**
    Global function from “db.class.php”. This PHP function calculates and returns the total time spent on attempts for a specific scenario by a particular user.
15. **getLastAttempt($sim_id, $userid)**
    Global function from “db.class.php”. This PHP function returns information about the latest attempt made by a specific user on a scenario.
16. **getMultiTotalTime($sim_id, $userid)**
    Global function from “db.class.php”. This PHP function calculates and returns the total time difference between pairs of "startdate" and "enddate" values for a specific user ("userid") and simulation ("scenario_id").
17. **getLastAttemptELearning($sim_id, $userid)**
    Global function from “db.class.php”. This PHP function returns the latest attempt record for a specific simulation ("scenario_id") and user ("userid") from the scenario_attempt_tbl table, where the attempt ID (aid) also exists in the scorm_attempt_history table.
18. **getLastAttemptHybridMulti($sim_id, $userid)**
    Global function from “db.class.php”. This PHP function returns the latest attempt record for a specific simulation ("scenario_id") and user ("userid") from the scenario_attempt_tbl table, where the attempt's uid is present either in the multi_score_tbl or open_res_tbl tables.
19. **getLastAttemptHybrid($sim_id, $userid)**
    Global function from “db.class.php”. This PHP function returns the latest attempt record from the scenario_attempt_tbl table for a specific simulation ("scenario_id") and user ("userid"), ensuring that the attempt's uid is present in either the score_tbl or the open_res_tbl tables.
20. **getCustomerSatisfaction($user_id, $sim_id, $column, $table)**
    Global function from “db.class.php”. This PHP function returns data from specific column from a table defined previously, filtering the results based on user ($uid) and simulation ($scenario_id), and grouping the results by question_id.
21. **getTotalAttemptHybridQues($sim_id, $userid)**
    Global function from “db.class.php”. This PHP function "getTotalAttemptHybridQues", retrieves the total number of hybrid questions attempted by a specific user for a given simulation.
22. **getTotalAttemptHybridQuesMulti($sim_id, $userid)**
    Global function from “db.class.php”. This PHP function, "getTotalAttemptHybridQuesMulti", is designed to retrieve and count the total number of attempts for hybrid questions (from both multi_score_tbl and open_res_tbl tables) for a specific simulation (sim_id) and user (user_id).
23. **get_percentage($totalQuestions, $satisfaction)**
    Global function from “db.class.php”. This function calculates the percentage of a part (number) relative to a total (total).

24. **isESimAttempted($sim_id, $userid)**
    Global function from “db.class.php”. The "isESimAttempted" function checks whether a user has attempted a specific simulation.
25. **isAttempted($sim_id, $userid)**
    Global function from “db.class.php”. The "isAttempted" function checks whether a user has attempted a specific simulation by querying a database to see if there are any relevant entries in several related tables.

Third Party JS Library

1. ChatApp
2. ChatGPT
3. OwlCarousel
4. DateRangePicker
5. CanvasJS
6. FancyBox
7. FullCalender
8. Tooltip
9. SweetAlert

JS File Used

1. **const getUrlVars = () => {}**
   The "getUrlVars" function extracts query parameters from the current URL and returns them as an array where each parameter is both an item and a property of the array.
2. **const tooltip = () => {}**
   This triggers tooltip functionality on tags with attribute *data-toggle="tooltip".*
3. **const getPercentage = (x, y, html = false) => {}**
   This code defines a JavaScript function named "getPercentage". It calculates the percentage value of "x" in relation to "y", optionally formatting the result as either an integer or a string with a percentage symbol.
4. **const progressBar = () => {}**
   The “progressBar” function iterates over each element with the class progressBar, adds a span element inside it, and sets the width of the span to match the value of the data-progress attribute of the original element
5. **const CircleBar = () => {}**
   The "CircleBar" function initializes circular progress bars on elements with specific classes by setting the "stroke-dashoffset" CSS property based on a data attribute.
6. **const limit = (string = '', limit = 0) => {}**
   This JavaScript code defines a function named "limit". It truncates a given string to a specified limit and appends an ellipsis ("...") if the string exceeds the limit.
7. **const parseTime = (time) => {}**
   This JavaScript code defines a function named "parseTime". It parses a time string in the format "HH:MM" and returns a formatted string displaying the hours and minutes.

8. **const getTotalSumTime = (timeArray, html = false) => {}**
   This JavaScript code defines a function named "getTotalSumTime". It calculates the total sum of time durations from an array of time strings in the format "HH:MM" and returns the result.
9. **const calculateMinutes = (timeArray) => {}**
   This JavaScript code defines a function named "calculateMinutes". It calculates the total number of minutes from a time string in the format "HH".
10. **const getUsername = (text) => {}**
    This JavaScript code defines a function named "getUsername". It extracts a username from a text string that matches a specific pattern, or generates a random username if no match is found.
11. **const getPDF = (id, title) => {}**
    This JavaScript code defines a function named "getPDF". It generates a PDF file from the content of a specified HTML element and downloads it.
12. **function cb(start, end) {}**
    The provided code defines a function named cb that updates the values of specific input fields based on the start and end dates provided as arguments. This function is handling date ranges in a web application, using a library like jQuery and a date picker plugin.
13. **$('#reportrange').daterangepicker({})**
    This triggers date range picker library for the selection of dates for reports generated.
14. **const setCookie = (name, value, days) => {}**
    This JavaScript code defines a function named "setCookie" used to set a cookie in the browser.
15. **const getCookie = (name) => {}**
    This JavaScript code defines a function named "getCookie" used to retrieve the value of a cookie by its name from the browser's cookies.
16. **$('.launch_sim').on('click', function(e){})**
    This jQuery code snippet sets up an event listener for clicks on elements with the class ".launch_sim". When clicked, it retrieves the value of a custom attribute "sim-id", stores it in a cookie named "'sim_id'", and sets the cookie to expire in 7 days.
17. **$('.sim_list').multiselect({})**
    $('#standing_groups').multiselect({})
    This code initializes a multiselect dropdown menu for elements for the specific class or ID as selector.
18. **const overview = () => {}**
    This JavaScript function overview is responsible for retrieving learner overview data via an AJAX POST request to *'includes/learner/overview-dashboard-data.php'* and updating various elements on the webpage with the received data.
19. **const performanceGrowth = (dataPoint = []) => {}**
    This JavaScript function, "performanceGrowth", is responsible for rendering a column chart using the CanvasJS library. It visualizes the growth percentage over attempts taken based on the provided data points.
20. **const groupsFilterData = (group_id, date, sim_id) => {}**
    This JavaScript function, "groupsFilterData", is responsible for filtering and updating data

based on group, date, and simulation ID. It utilizes AJAX POST requests to retrieve data from the server and updates specific elements on the webpage accordingly.

21. **const filterData = (date, sim_id) => {}**
    The "filterData" function is responsible for filtering and updating various elements on a webpage based on the provided date and simulation ID. It makes use of AJAX POST requests to retrieve data from the server and dynamically updates the HTML content of different elements accordingly.
22. **const simList = (date, id, apply = false, clicked = false) => {}**
    The "simList" function is responsible for retrieving and populating a list of simulations based on the provided date and course ID. It utilizes AJAX GET requests to *'includes/learner/overview-dashboard-data.php'* to fetch data from the server and dynamically updates the options of a multiselect dropdown element accordingly.

File Path: content/js/scorm.js

1. **function setupScormApi() {}**
   The "setupScormApi" function is responsible for setting up the SCORM API functionality. It initializes SCORM communication and handles launching SCORM content within a popup window.
2. **function LMSInitialize(initializeInput) {}**
   The "LMSInitialize" function is a part of the SCORM (Sharable Content Object Reference Model) API. It is called to initialize communication between the SCORM content and the SCORM runtime environment (usually a learning management system or LMS).
3. **function LMSGetValue(varname) {}**
   The "LMSGetValue" function is another essential part of the SCORM API. It's called to retrieve the value of a specific data model element (variable) from the SCORM runtime environment (typically an LMS).
4. **function convertTimeString(timeString) {}**
   The "convertTimeString" function is designed to convert a time string into a standardized format of "HH:MM".
5. **function setStatusAndResult() {}**
   The "setStatusAndResult" function is responsible for setting the status and result of a SCORM lesson based on various conditions.
6. **function saveOrUpdate() {}**
   The "saveOrUpdate" function is responsible for saving or updating the SCORM attempt data based on certain conditions.
7. **function resetStatusVars() {}**
   The "resetStatusVars" function is used to reset the status-related variables to their initial values.
8. **function LMSSetValue(varname, varvalue) {}**
   The "LMSSetValue" function is responsible for setting values to SCORM data model elements based on the provided varname and varvalue parameters.

9. **function LMSFinish(finishInput) {}**
   The "LMSFinish" function is responsible for finalizing the SCORM session when the learner finishes interacting with the content.
10. **const base_url = (page = '') => {}**
    The "base_url" function is used to generate the base URL of the current page, optionally appending a specific page or path to it.
11. **const getUid = (sid) => {}**
    The "getUid" function sends AJAX request to 'includes/ajax.php' with parameter user ID (uid) from the server and retrieves data, then creates an attempt entry in the "scenario_attempt_tbl" table with this "uid" by sending AJAX request to *'api-process.php'*, a given scenario ID (sid), and a user ID (userid). If successful, it sets the attempt ID (aid) and calls another function to handle additional actions.
12. **function getLaunchURL(path) {}**
    The "getLaunchURL" function retrieves the launch URL from an IMS manifest file (*imsmanifest.xml*) located at a given path. This function returns a promise that resolves with the full launch URL.
13. **function getOrgName(path, callback) {}**
    The "getOrgName" function retrieves the organization name and schema version from an IMS manifest file (*imsmanifest.xml*) located at a specified path. This function uses asynchronous operations (await and fetch) to handle the network request and parsing of the XML document.
14. **function msToTime(s) {}**
    The "msToTime" function converts milliseconds to a formatted time string in the format "HH:MM".
15. **function update_attempt_tbl() {}**
    The "update_attempt_tbl" function is responsible for updating the attempt table with relevant information.
16. **function toHoursAndMinutes(totalSeconds) {}**
    The "toHoursAndMinutes" function is used to convert a total number of seconds into hours, minutes, and seconds.
17. **function insert_attempt_tbl() {}**
    The "insert_attempt_tbl" function is responsible for inserting a new record into the attempt table.

File Path: content/js/e-learning.js

1. **const getTimestamp = (duration) => {}**
   The "getTimestamp" function converts a duration in seconds into a formatted timestamp string in the format mm:ss.
2. **const getBaseUrl = (page = '') => {}**
   The "getBaseUrl" function generates the base URL for a web page, optionally appending a specific page path.
3. **const getUID = (sid, userId) => {}**
   The "getUID" function is used to retrieve the unique identifier (UID) associated with a

user by sending AJAX request to *'includes/ajax.php'* with parameters "{ get_uid_check_attempts: true, sim_id: sid, uid: userId }" and create a new attempt record in the database by posting data through AJAX request to 'api-process.php' for a scenario.

4. **const getFileNameFromUrl = (url) => {}**
   The "getFileNameFromUrl" function is used to extract the file name from a given URL.
5. **const saveAttempt = (ev_aid, file, durationTime, currentTime, userId, type) => {}**
   The saveAttempt function is used to save attempt history related to e-learning activities by sending data through AJAX request to *'api-process.php'*.
6. **$('.el_launch_mp4').on('click', function() {**
   This code handles the launching of MP4 videos, displays them in a modal, tracks user interactions, and saves relevant data about the user's interaction with the video.
7. **const videoModal = (preview, userId, simId, src) => {}**
   This function "videoModal" is responsible for dynamically generating and displaying a modal containing an HTML video player.
8. **$('body').on('click', '.el_video_close', function() {})**
   This listener ensures that when the close button within the video modal is clicked, it handles the necessary actions such as saving the attempt details and closing the modal.
9. **const pdfModal = (preview, userId, simId, src) => {}**
   This function sets up a modal window to display PDF documents with options for closing the modal and handling user interactions.
10. **$('.el_launch_pdf').on('click', function() {})**
    This event handler sets up functionality to open a PDF document in a modal window when the corresponding element is clicked.
11. **$('body').on('click', '.el_pdf_close', function() {})**
    This event handler provides functionality to close the PDF modal window and save the attempt details if necessary.

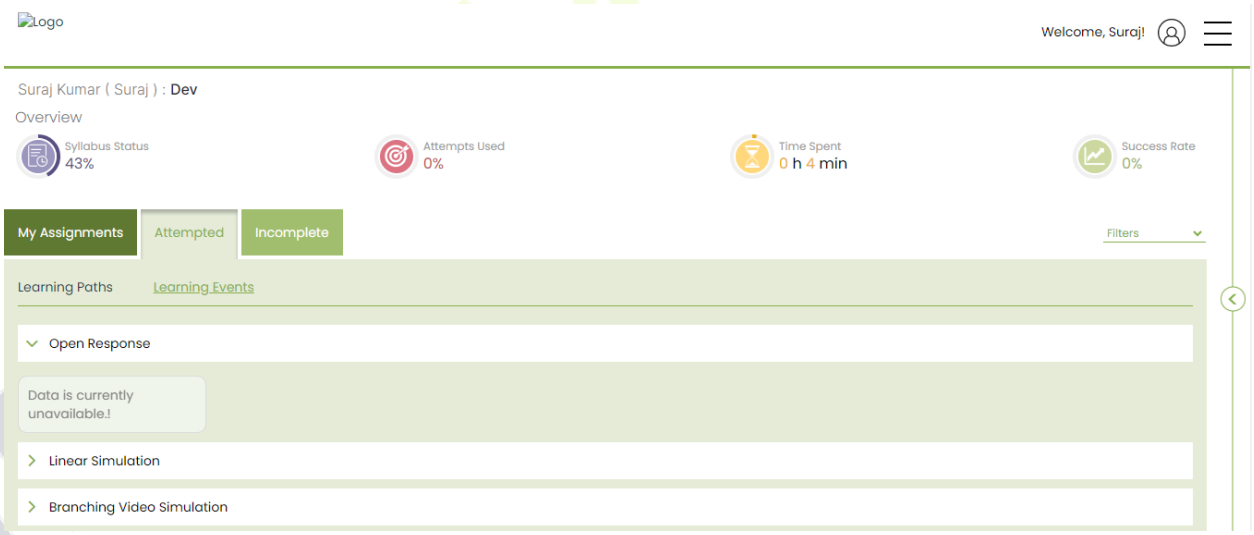
File Path: content/js/learner-assigned-events.js

1. **const getEvents = () => {}**
   This function "getEvents" is designed to retrieve events data from the server. It sends an AJAX POST request to a specified URL (*'includes/learner/events.php'*) with the intention of retrieving events categorized by category.
2. **const textLimit = (string = '', limit = 0) => {}**
   The "textLimit" function is designed to limit the length of a given string to a specified number of characters and append ellipsis (...) if the string exceeds the limit.
3. **const genrateItems = (data, key) => {}**
   The "genrateItems" function generates HTML elements based on the provided data and appends them to the DOM.
4. **const genrateItemsNew = (data, key, id) => {}**
   The "genrateItemsNew" function is similar to genrateItems, but it includes additional functionality to remove an empty event element before appending new items.

5. **$('body').on('click', '.view_event_attachment', function(e){})**
   This code is an event handler for opening a FancyBox modal with an iframe.
6. **$('body').on('click', '.attended', function(e){})**
   This JavaScript code is an event handler for handling the click event on elements with the class .attended.

Attempted > Learning Events

Template Name: attempted-events.php



PHP Functions Used

1. **getClientPermission($client_id)**
   Global function from "db.class.php". This function returns all the data for the permissions from table client_permissions for the give client_id.
2. **getAssigneeName($userid)**
   Global function from "db.class.php". This function will return all the data for user as per the User ID passed in the argument.
3. **getAssignPathData((int)$userid))**
   Global function from "db.class.php". This function will return all the data for the learning path for the user.
4. **SQL Queries**
   $catSql = "SELECT * FROM category_tbl WHERE category != '' AND (cat_id IN (SELECT DISTINCT scenario_category FROM scenario_master WHERE scenario_id IN ($getSim)) OR cat_id IN (SELECT DISTINCT category FROM events AS ev LEFT JOIN group_tbl AS g1 ON FIND_IN_SET(g1.group_id, ev.groups) LEFT JOIN group_tbl AS g2 ON FIND_IN_SET($userid, g1.learner) WHERE FIND_IN_SET($userid, g2.learner) AND FIND_IN_SET($userid, g1.learner))) $corderby";
   The query retrieves all categories from category_tbl that are non-empty and meet one of the following criteria:

A. The cat_id is associated with a scenario_category from scenario_master where the scenario_id is in $getSim.
B. The cat_id is associated with a category from events where complex conditions involving user ID ($userid) and group membership in group_tbl are satisfied.

Finally, the results are ordered according to the $corderby clause. This complex query is likely used in a scenario-based application where categories are filtered based on specific scenarios and user group associations.

$sql = "SELECT DISTINCT s.scenario_id, s.sim_ques_type, s.sim_temp_type, s.Scenario_title, LEFT(s.Scenario_title, 50) AS stitle, s.duration, s.image_url, s.video_url, s.audio_url, s.web_object_file, s.scenario_type, s.scenario_category, s.author_scenario_type, s.sim_cover_img, s.web_object FROM scenario_master AS s LEFT JOIN scenario_attempt_tbl AS st ON s.scenario_id = st.scenario_id WHERE s.scenario_category = ? AND s.scenario_id IN ($getSim) AND st.userid = ? AND (st.aid IN (SELECT aid FROM score_tbl) OR st.aid IN (SELECT aid FROM multi_score_tbl) OR st.aid IN (SELECT aid FROM open_res_tbl) OR st.aid IN (SELECT aid FROM scorm_attempt_history))";
The query retrieves distinct scenario details from scenario_master where:
a. The scenario_category matches a given category.
b. The scenario_id is within a specified list ($getSim).
c. The userid in scenario_attempt_tbl matches a specified user.
d. The aid in scenario_attempt_tbl matches an aid in one of the specified score tables.
The fields selected include scenario ID, question type, template type, scenario title (both full and truncated to 50 characters), duration, URLs for image, video, audio, web object file, scenario type, scenario category, author scenario type, cover image, and web object.

5. **checkMandatory($sim_id)**
   Global function from “db.class.php”. This function will return all the data from assignment_tbl for the given Simulation ID.
6. **getScenarioTypes(md5($author_scenario_type))**
   Global function from “db.class.php”. This function will return all the different scenario_type for the author.
7. **getTotalAttempt($sim_id, $userid)**
   Global function from “db.class.php”. This function will return total attempts for the user and simulation passed as arguments.
8. **getAssignSimOtherData($sim_id, $userid)**
   Global function from “db.class.php”. This PHP function, "getAssignSimOtherData", is designed to retrieve assignment data related to a simulation ("$sim_id") for a particular user ("$user_id").
9. **getTotalQuestionsByScenario($en_sim_id)**
   Global function from “db.class.php”. This PHP function return total number of question for the particular simulation ID.

10. **getOpenResTotalTime($sim_id, $userid)**
    Global function from “db.class.php”. This PHP function calculates and returns the total time spent on a specific scenario attempt by a user.
11. **getLastAttemptOpenResp($sim_id, $userid)**
    Global function from “db.class.php”. This PHP function retrieves the latest attempt data for a specific scenario (scenario_id) and user (userid).
12. **getMultiTotalTime($sim_id, $userid)**
    Global function from “db.class.php”. This PHP function calculates and returns total duration of all attempts for a specific user ("userid") and scenario ("scenario_id") by summing the differences between the start and end times of each attempt.
13. **getLastAttemptMulti($sim_id, $userid)**
    Global function from “db.class.php”. This PHP function retrieves the most recent attempt made by a specific user (userid) on a particular scenario (scenario_id) where the attempt's unique identifier (uid) exists in the multi_score_tbl table.
14. **getTotalTime($sim_id, $userid)**
    Global function from “db.class.php”. This PHP function calculates and returns the total time spent on attempts for a specific scenario by a particular user.
15. **getLastAttempt($sim_id, $userid)**
    Global function from “db.class.php”. This PHP function returns information about the latest attempt made by a specific user on a scenario.
16. **getMultiTotalTime($sim_id, $userid)**
    Global function from “db.class.php”. This PHP function calculates and returns the total time difference between pairs of "startdate" and "enddate" values for a specific user ("userid") and simulation ("scenario_id").
17. **getLastAttemptELearning($sim_id, $userid)**
    Global function from “db.class.php”. This PHP function returns the latest attempt record for a specific simulation ("scenario_id") and user ("userid") from the scenario_attempt_tbl table, where the attempt ID (aid) also exists in the scorm_attempt_history table.
18. **getLastAttemptHybridMulti($sim_id, $userid)**
    Global function from “db.class.php”. This PHP function returns the latest attempt record for a specific simulation ("scenario_id") and user ("userid") from the scenario_attempt_tbl table, where the attempt's uid is present either in the multi_score_tbl or open_res_tbl tables.
19. **getLastAttemptHybrid($sim_id, $userid)**
    Global function from “db.class.php”. This PHP function returns the latest attempt record from the scenario_attempt_tbl table for a specific simulation ("scenario_id") and user ("userid"), ensuring that the attempt's uid is present in either the score_tbl or the open_res_tbl tables.
20. **getCustomerSatisfaction($user_id, $sim_id, $column, $table)**
    Global function from “db.class.php”. This PHP function returns data from specific column from a table defined previously, filtering the results based on user ($uid) and simulation ($scenario_id), and grouping the results by question_id.
21. **getBranchingSatisfaction($user_id, $sim_id, $column, $table)**
    Global function from “db.class.php”. This PHP function selects and returns the

choice_option_id from a specified table. The table name is represented by the variable $table.

22. **getTotalAttemptHybridQues($sim_id, $userid)**
    Global function from “db.class.php”. This PHP function "getTotalAttemptHybridQues", retrieves the total number of hybrid questions attempted by a specific user for a given simulation.
23. **getTotalAttemptHybridQuesMulti($sim_id, $userid)**
    Global function from “db.class.php”. This PHP function, "getTotalAttemptHybridQuesMulti", is designed to retrieve and count the total number of attempts for hybrid questions (from both multi_score_tbl and open_res_tbl tables) for a specific simulation (sim_id) and user (user_id).
24. **get_percentage($totalQuestions, $satisfaction)**
    Global function from “db.class.php”. This function calculates the percentage of a part (number) relative to a total (total).
25. **isESimAttempted($sim_id, $userid)**
    Global function from “db.class.php”. The "isESimAttempted" function checks whether a user has attempted a specific simulation.
26. **isAttempted($sim_id, $userid)**
    Global function from “db.class.php”. The "isAttempted" function checks whether a user has attempted a specific simulation by querying a database to see if there are any relevant entries in several related tables.

Third Party JS Library

1. ChatApp
2. ChatGPT
3. OwlCarousel
4. DateRangePicker
5. CanvasJS
6. FancyBox
7. FullCalender
8. Tooltip
9. SweetAlert

JS File Used

File Path: content/js/learner-events.js

1. **const getUrlVars = () => {}**
   The "getUrlVars" function extracts query parameters from the current URL and returns them as an array where each parameter is both an item and a property of the array.
2. **const tooltip = () => {}**
   This triggers tooltip functionality on tags with attribute *data-toggle="tooltip".*

3. **const getPercentage = (x, y, html = false) => {}**
   This code defines a JavaScript function named "getPercentage". It calculates the percentage value of "x" in relation to "y", optionally formatting the result as either an integer or a string with a percentage symbol.
4. **const progressBar = () => {}**
   The “progressBar” function iterates over each element with the class progressBar, adds a span element inside it, and sets the width of the span to match the value of the data-progress attribute of the original element
5. **const CircleBar = () => {}**
   The "CircleBar" function initializes circular progress bars on elements with specific classes by setting the "stroke-dashoffset" CSS property based on a data attribute.
6. **const limit = (string = '', limit = 0) => {}**
   This JavaScript code defines a function named "limit". It truncates a given string to a specified limit and appends an ellipsis ("...") if the string exceeds the limit.
7. **const parseTime = (time) => {}**
   This JavaScript code defines a function named "parseTime". It parses a time string in the format "HH:MM" and returns a formatted string displaying the hours and minutes.
8. **const getTotalSumTime = (timeArray, html = false) => {}**
   This JavaScript code defines a function named "getTotalSumTime". It calculates the total sum of time durations from an array of time strings in the format "HH:MM" and returns the result.
9. **const calculateMinutes = (timeArray) => {}**
   This JavaScript code defines a function named "calculateMinutes". It calculates the total number of minutes from a time string in the format "HH".
10. **const getUsername = (text) => {}**
    This JavaScript code defines a function named "getUsername". It extracts a username from a text string that matches a specific pattern, or generates a random username if no match is found.
11. **const getPDF = (id, title) => {}**
    This JavaScript code defines a function named "getPDF". It generates a PDF file from the content of a specified HTML element and downloads it.
12. **function cb(start, end) {}**
    The provided code defines a function named cb that updates the values of specific input fields based on the start and end dates provided as arguments. This function is handling date ranges in a web application, using a library like jQuery and a date picker plugin.
13. **$('#reportrange').daterangepicker({})**
    This triggers date range picker library for the selection of dates for reports generated.
14. **const setCookie = (name, value, days) => {}**
    This JavaScript code defines a function named "setCookie" used to set a cookie in the browser.
15. **const getCookie = (name) => {}**
    This JavaScript code defines a function named "getCookie" used to retrieve the value of a cookie by its name from the browser's cookies.

16. **$('.launch_sim').on('click', function(e){})**
    This jQuery code snippet sets up an event listener for clicks on elements with the class ".launch_sim". When clicked, it retrieves the value of a custom attribute "sim-id", stores it in a cookie named "'sim_id'", and sets the cookie to expire in 7 days.
17. **$('.sim_list').multiselect({})**
    $('#standing_groups').multiselect({})
    This code initializes a multiselect dropdown menu for elements for the specific class or ID as selector.
18. **const overview = () => {}**
    This JavaScript function overview is responsible for retrieving learner overview data via an AJAX POST request to *'includes/learner/overview-dashboard-data.php'* and updating various elements on the webpage with the received data.
19. **const performanceGrowth = (dataPoint = []) => {}**
    This JavaScript function, "performanceGrowth", is responsible for rendering a column chart using the CanvasJS library. It visualizes the growth percentage over attempts taken based on the provided data points.
20. **const groupsFilterData = (group_id, date, sim_id) => {}**
    This JavaScript function, "groupsFilterData", is responsible for filtering and updating data based on group, date, and simulation ID. It utilizes AJAX POST requests to retrieve data from the server and updates specific elements on the webpage accordingly.
21. **const filterData = (date, sim_id) => {}**
    The "filterData" function is responsible for filtering and updating various elements on a webpage based on the provided date and simulation ID. It makes use of AJAX POST requests to retrieve data from the server and dynamically updates the HTML content of different elements accordingly.
22. **const simList = (date, id, apply = false, clicked = false) => {}**
    The "simList" function is responsible for retrieving and populating a list of simulations based on the provided date and course ID. It utilizes AJAX GET requests to *'includes/learner/overview-dashboard-data.php'* to fetch data from the server and dynamically updates the options of a multiselect dropdown element accordingly.

File Path: content/js/scorm.js

1. **function setupScormApi() {}**
   The "setupScormApi" function is responsible for setting up the SCORM API functionality. It initializes SCORM communication and handles launching SCORM content within a popup window.
2. **function LMSInitialize(initializeInput) {}**
   The "LMSInitialize" function is a part of the SCORM (Sharable Content Object Reference Model) API. It is called to initialize communication between the SCORM content and the SCORM runtime environment (usually a learning management system or LMS).
3. **function LMSGetValue(varname) {}**
   The "LMSGetValue" function is another essential part of the SCORM API. It's called to

retrieve the value of a specific data model element (variable) from the SCORM runtime environment (typically an LMS).

4. **function convertTimeString(timeString) {}**
   The "convertTimeString" function is designed to convert a time string into a standardized format of "HH:MM".
5. **function setStatusAndResult() {}**
   The "setStatusAndResult" function is responsible for setting the status and result of a SCORM lesson based on various conditions.
6. **function saveOrUpdate() {}**
   The "saveOrUpdate" function is responsible for saving or updating the SCORM attempt data based on certain conditions.
7. **function resetStatusVars() {}**
   The "resetStatusVars" function is used to reset the status-related variables to their initial values.
8. **function LMSSetValue(varname, varvalue) {}**
   The "LMSSetValue" function is responsible for setting values to SCORM data model elements based on the provided varname and varvalue parameters.
9. **function LMSFinish(finishInput) {}**
   The "LMSFinish" function is responsible for finalizing the SCORM session when the learner finishes interacting with the content.
10. **const base_url = (page = '') => {}**
    The "base_url" function is used to generate the base URL of the current page, optionally appending a specific page or path to it.
11. **const getUid = (sid) => {}**
    The "getUid" function sends AJAX request to *'includes/ajax.php'* with parameter user ID (uid) from the server and retrieves data, then creates an attempt entry in the "scenario_attempt_tbl" table with this "uid" by sending AJAX request to 'api-process.php', a given scenario ID (sid), and a user ID (userid). If successful, it sets the attempt ID (aid) and calls another function to handle additional actions.
12. **function getLaunchURL(path) {}**
    The "getLaunchURL" function retrieves the launch URL from an IMS manifest file (*imsmanifest.xml*) located at a given path. This function returns a promise that resolves with the full launch URL.
13. **function getOrgName(path, callback) {}**
    The "getOrgName" function retrieves the organization name and schema version from an IMS manifest file (*imsmanifest.xml*) located at a specified path. This function uses asynchronous operations (await and fetch) to handle the network request and parsing of the XML document.
14. **function msToTime(s) {}**
    The "msToTime" function converts milliseconds to a formatted time string in the format "HH:MM".
15. **function update_attempt_tbl() {}**
    The "update_attempt_tbl" function is responsible for updating the attempt table with relevant information.

16. **function toHoursAndMinutes(totalSeconds) {}**
    The "toHoursAndMinutes" function is used to convert a total number of seconds into hours, minutes, and seconds.
17. **function insert_attempt_tbl() {}**
    The "insert_attempt_tbl" function is responsible for inserting a new record into the attempt table.

File Path: content/js/e-learning.js

1. **const getTimestamp = (duration) => {}**
   The "getTimestamp" function converts a duration in seconds into a formatted timestamp string in the format mm:ss.
2. **const getBaseUrl = (page = '') => {}**
   The "getBaseUrl" function generates the base URL for a web page, optionally appending a specific page path.
3. **const getUID = (sid, userId) => {}**
   The "getUID" function is used to retrieve the unique identifier (UID) associated with a user by sending AJAX request to *'includes/ajax.php'* with parameters "{ get_uid_check_attempts: true, sim_id: sid, uid: userId }" and create a new attempt record in the database by posting data through AJAX request to 'api-process.php' for a scenario.
4. **const getFileNameFromUrl = (url) => {}**
   The "getFileNameFromUrl" function is used to extract the file name from a given URL.
5. **const saveAttempt = (ev_aid, file, durationTime, currentTime, userId, type) => {}**
   The saveAttempt function is used to save attempt history related to e-learning activities by sending data through AJAX request to *'api-process.php'*.
6. **$('.el_launch_mp4').on('click', function() {**
   This code handles the launching of MP4 videos, displays them in a modal, tracks user interactions, and saves relevant data about the user's interaction with the video.
7. **const videoModal = (preview, userId, simId, src) => {}**
   This function "videoModal" is responsible for dynamically generating and displaying a modal containing an HTML video player.
8. **$('body').on('click', '.el_video_close', function() {})**
   This listener ensures that when the close button within the video modal is clicked, it handles the necessary actions such as saving the attempt details and closing the modal.
9. **const pdfModal = (preview, userId, simId, src) => {}**
   This function sets up a modal window to display PDF documents with options for closing the modal and handling user interactions.
10. **$('.el_launch_pdf').on('click', function() {})**
    This event handler sets up functionality to open a PDF document in a modal window when the corresponding element is clicked.
11. **$('body').on('click', '.el_pdf_close', function() {})**
    This event handler provides functionality to close the PDF modal window and save the attempt details if necessary.

File Path: content/js/learner-assigned-events.js

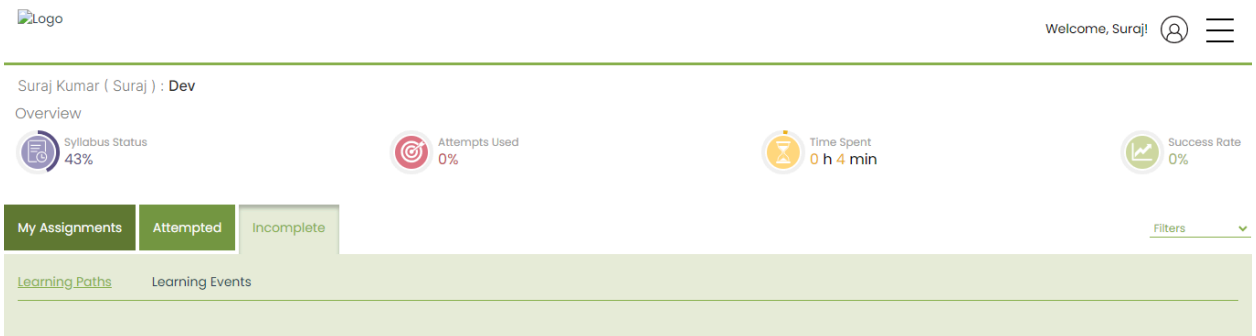
1. **const getEvents = () => {}**
   This function "getEvents" is designed to retrieve events data from the server. It sends an AJAX POST request to a specified URL (*'includes/learner/events.php'*) with the intention of retrieving events categorized by category.
2. **const textLimit = (string = '', limit = 0) => {}**
   The "textLimit" function is designed to limit the length of a given string to a specified number of characters and append ellipsis (...) if the string exceeds the limit.
3. **const genrateItems = (data, key) => {}**
   The "genrateItems" function generates HTML elements based on the provided data and appends them to the DOM.
4. **const genrateItemsNew = (data, key, id) => {}**
   The "genrateItemsNew" function is similar to genrateItems, but it includes additional functionality to remove an empty event element before appending new items.
5. **$('body').on('click', '.view_event_attachment', function(e){})**
   This code is an event handler for opening a FancyBox modal with an iframe.
6. **$('body').on('click', '.attended', function(e){})**
   This JavaScript code is an event handler for handling the click event on elements with the class .attended.

File Path: content/js/learner-events-schedule.js

1. **const getLatLong = () => {}**
   The getLatLong function attempts to retrieve the user's current geographical coordinates (latitude and longitude) using the browser's geolocation API.
2. **function viewEventTemp(info) {}**
   The viewEventTemp function generates an HTML string that displays detailed information about an event, to be inserted into a web page, where it will display the formatted event details to the user.
3. **$('body').on('change', '#time-zone-selector', function (e) {})**
   This script is a part of a feature where users can change the time zone of a calendar view, and the change is also sent to the server to update the user's settings or preferences.
4. **$('body').on('click', '.view_event_attachment', function (e) {})**
   This script is used to display external content (e.g., a webpage or document) within a modal popup on a webpage, providing a user-friendly way to view the content without navigating away from the current page.

Incomplete > Learning Path

Template Name: incomplete-path.php



PHP Functions Used

1. **getAssigneeName($userid)**
   Global function from "db.class.php". This function will return all the data for user as per the User ID passed in the argument.
2. **getAssigneeName($userid)**
   Global function from "db.class.php". This function will return all the data for user as per the User ID passed in the argument.
3. **getAssignPathData((int)$userid))**
   Global function from "db.class.php". This function will return all the data for the learning path for the user.
4. **SQL Queries**
   $catSql = "SELECT * FROM path_assignment_tbl WHERE path_id IN ($getPath) $corderby";
   This SQL query retrieves records from the "path_assignment_tbl" table where the "path_id" matches any of the values specified in the $getPath variable, and optionally applies additional conditions or sorting specified by the $corderby variable.

   $sql = "SELECT DISTINCT scenario_id, sim_ques_type, sim_temp_type, Scenario_title, LEFT(Scenario_title, 50) AS stitle, duration, image_url, video_url, audio_url, web_object_file, scenario_type, scenario_category, author_scenario_type, sim_cover_img, web_object FROM scenario_master WHERE scenario_id IN ($getSim) AND scenario_id NOT IN (SELECT scenario_id FROM scenario_attempt_tbl WHERE userid = ?)";
   This SQL query retrieves distinct scenarios from the scenario_master table based on certain criteria like only include scenarios whose IDs are in the list provided by $getSim and exclude scenarios that have already been attempted by a specific user, identified by their user ID.
5. **checkMandatory($sim_id)**
   Global function from "db.class.php". This function will return all the data from assignment_tbl for the given Simulation ID.

6. **getScenarioTypes(md5($author_scenario_type))**
   Global function from “db.class.php”. This function will return all the different scenario_type for the author.
7. **getTotalAttempt($sim_id, $userid)**
   Global function from “db.class.php”. This function will return total attempts for the user and simulation passed as arguments.
8. **getAssignSimOtherData($sim_id, $userid)**
   Global function from “db.class.php”. This PHP function, "getAssignSimOtherData", is designed to retrieve assignment data related to a simulation ("$sim_id") for a particular user ("$user_id").
9. **getTotalQuestionsByScenario($en_sim_id)**
   Global function from “db.class.php”. This PHP function return total number of question for the particular simulation ID.

Third Party JS Library

1. ChatApp
2. ChatGPT
3. OwlCarousel
4. DateRangePicker
5. CanvasJS
6. FancyBox
7. FullCalender
8. Tooltip
9. SweetAlert

JS File Used

File Path: content/js/learner-events.js

1. **const getUrlVars = () => {}**
   The "getUrlVars" function extracts query parameters from the current URL and returns them as an array where each parameter is both an item and a property of the array.
2. **const tooltip = () => {}**
   This triggers tooltip functionality on tags with attribute *data-toggle="tooltip".*
3. **const getPercentage = (x, y, html = false) => {}**
   This code defines a JavaScript function named "getPercentage". It calculates the percentage value of "x" in relation to "y", optionally formatting the result as either an integer or a string with a percentage symbol.
4. **const progressBar = () => {}**
   The “progressBar” function iterates over each element with the class progressBar, adds a span element inside it, and sets the width of the span to match the value of the data-progress attribute of the original element

5. **const CircleBar = () => {}**
   The "CircleBar" function initializes circular progress bars on elements with specific classes by setting the "stroke-dashoffset" CSS property based on a data attribute.
6. **const limit = (string = '', limit = 0) => {}**
   This JavaScript code defines a function named "limit". It truncates a given string to a specified limit and appends an ellipsis ("...") if the string exceeds the limit.
7. **const parseTime = (time) => {}**
   This JavaScript code defines a function named "parseTime". It parses a time string in the format "HH:MM" and returns a formatted string displaying the hours and minutes.
8. **const getTotalSumTime = (timeArray, html = false) => {}**
   This JavaScript code defines a function named "getTotalSumTime". It calculates the total sum of time durations from an array of time strings in the format "HH:MM" and returns the result.
9. **const calculateMinutes = (timeArray) => {}**
   This JavaScript code defines a function named "calculateMinutes". It calculates the total number of minutes from a time string in the format "HH".
10. **const getUsername = (text) => {}**
    This JavaScript code defines a function named "getUsername". It extracts a username from a text string that matches a specific pattern, or generates a random username if no match is found.
11. **const getPDF = (id, title) => {}**
    This JavaScript code defines a function named "getPDF". It generates a PDF file from the content of a specified HTML element and downloads it.
12. **function cb(start, end) {}**
    The provided code defines a function named cb that updates the values of specific input fields based on the start and end dates provided as arguments. This function is handling date ranges in a web application, using a library like jQuery and a date picker plugin.
13. **$('#reportrange').daterangepicker({})**
    This triggers date range picker library for the selection of dates for reports generated.
14. **const setCookie = (name, value, days) => {}**
    This JavaScript code defines a function named "setCookie" used to set a cookie in the browser.
15. **const getCookie = (name) => {}**
    This JavaScript code defines a function named "getCookie" used to retrieve the value of a cookie by its name from the browser's cookies.
16. **$('.launch_sim').on('click', function(e){})**
    This jQuery code snippet sets up an event listener for clicks on elements with the class ".launch_sim". When clicked, it retrieves the value of a custom attribute "sim-id", stores it in a cookie named "'sim_id'", and sets the cookie to expire in 7 days.
17. **$('.sim_list').multiselect({})**
    $('#standing_groups').multiselect({})
    This code initializes a multiselect dropdown menu for elements for the specific class or ID as selector.

18. **const overview = () => {}**
    This JavaScript function overview is responsible for retrieving learner overview data via an AJAX POST request to *'includes/learner/overview-dashboard-data.php'* and updating various elements on the webpage with the received data.
19. **const performanceGrowth = (dataPoint = []) => {}**
    This JavaScript function, "performanceGrowth", is responsible for rendering a column chart using the CanvasJS library. It visualizes the growth percentage over attempts taken based on the provided data points.
20. **const groupsFilterData = (group_id, date, sim_id) => {}**
    This JavaScript function, "groupsFilterData", is responsible for filtering and updating data based on group, date, and simulation ID. It utilizes AJAX POST requests to retrieve data from the server and updates specific elements on the webpage accordingly.
21. **const filterData = (date, sim_id) => {}**
    The "filterData" function is responsible for filtering and updating various elements on a webpage based on the provided date and simulation ID. It makes use of AJAX POST requests to retrieve data from the server and dynamically updates the HTML content of different elements accordingly.
22. **const simList = (date, id, apply = false, clicked = false) => {}**
    The "simList" function is responsible for retrieving and populating a list of simulations based on the provided date and course ID. It utilizes AJAX GET requests to *'includes/learner/overview-dashboard-data.php'* to fetch data from the server and dynamically updates the options of a multiselect dropdown element accordingly.

File Path: content/js/scorm.js

1. **function setupScormApi() {}**
   The "setupScormApi" function is responsible for setting up the SCORM API functionality. It initializes SCORM communication and handles launching SCORM content within a popup window.
2. **function LMSInitialize(initializeInput) {}**
   The "LMSInitialize" function is a part of the SCORM (Sharable Content Object Reference Model) API. It is called to initialize communication between the SCORM content and the SCORM runtime environment (usually a learning management system or LMS).
3. **function LMSGetValue(varname) {}**
   The "LMSGetValue" function is another essential part of the SCORM API. It's called to retrieve the value of a specific data model element (variable) from the SCORM runtime environment (typically an LMS).
4. **function convertTimeString(timeString) {}**
   The "convertTimeString" function is designed to convert a time string into a standardized format of "HH:MM".
5. **function setStatusAndResult() {}**
   The "setStatusAndResult" function is responsible for setting the status and result of a SCORM lesson based on various conditions.

6. **function saveOrUpdate() {}**
   The "saveOrUpdate" function is responsible for saving or updating the SCORM attempt data based on certain conditions.
7. **function resetStatusVars() {}**
   The "resetStatusVars" function is used to reset the status-related variables to their initial values.
8. **function LMSSetValue(varname, varvalue) {}**
   The "LMSSetValue" function is responsible for setting values to SCORM data model elements based on the provided varname and varvalue parameters.
9. **function LMSFinish(finishInput) {}**
   The "LMSFinish" function is responsible for finalizing the SCORM session when the learner finishes interacting with the content.
10. **const base_url = (page = '') => {}**
    The "base_url" function is used to generate the base URL of the current page, optionally appending a specific page or path to it.
11. **const getUid = (sid) => {}**
    The "getUid" function sends AJAX request to *'includes/ajax.php'* with parameter user ID (uid) from the server and retrieves data, then creates an attempt entry in the "scenario_attempt_tbl" table with this "uid" by sending AJAX request to *'api-process.php'*, a given scenario ID (sid), and a user ID (userid). If successful, it sets the attempt ID (aid) and calls another function to handle additional actions.
12. **function getLaunchURL(path) {}**
    The "getLaunchURL" function retrieves the launch URL from an IMS manifest file (*imsmanifest.xml*) located at a given path. This function returns a promise that resolves with the full launch URL.
13. **function getOrgName(path, callback) {}**
    The "getOrgName" function retrieves the organization name and schema version from an IMS manifest file (*imsmanifest.xml*) located at a specified path. This function uses asynchronous operations (await and fetch) to handle the network request and parsing of the XML document.
14. **function msToTime(s) {}**
    The "msToTime" function converts milliseconds to a formatted time string in the format "HH:MM".
15. **function update_attempt_tbl() {}**
    The "update_attempt_tbl" function is responsible for updating the attempt table with relevant information.
16. **function toHoursAndMinutes(totalSeconds) {}**
    The "toHoursAndMinutes" function is used to convert a total number of seconds into hours, minutes, and seconds.
17. **function insert_attempt_tbl() {}**
    The "insert_attempt_tbl" function is responsible for inserting a new record into the attempt table.

File Path: content/js/e-learning.js

1. **const getTimestamp = (duration) => {}**
   The "getTimestamp" function converts a duration in seconds into a formatted timestamp string in the format mm:ss.
2. **const getBaseUrl = (page = '') => {}**
   The "getBaseUrl" function generates the base URL for a web page, optionally appending a specific page path.
3. **const getUID = (sid, userId) => {}**
   The "getUID" function is used to retrieve the unique identifier (UID) associated with a user by sending AJAX request to *'includes/ajax.php'* with parameters "{ get_uid_check_attempts: true, sim_id: sid, uid: userId }" and create a new attempt record in the database by posting data through AJAX request to 'api-process.php' for a scenario.
4. **const getFileNameFromUrl = (url) => {}**
   The "getFileNameFromUrl" function is used to extract the file name from a given URL.
5. **const saveAttempt = (ev_aid, file, durationTime, currentTime, userId, type) => {}**
   The saveAttempt function is used to save attempt history related to e-learning activities by sending data through AJAX request to *'api-process.php'*.
6. **$('.el_launch_mp4').on('click', function() {**
   This code handles the launching of MP4 videos, displays them in a modal, tracks user interactions, and saves relevant data about the user's interaction with the video.
7. **const videoModal = (preview, userId, simId, src) => {}**
   This function "videoModal" is responsible for dynamically generating and displaying a modal containing an HTML video player.
8. **$('body').on('click', '.el_video_close', function() {})**
   This listener ensures that when the close button within the video modal is clicked, it handles the necessary actions such as saving the attempt details and closing the modal.
9. **const pdfModal = (preview, userId, simId, src) => {}**
   This function sets up a modal window to display PDF documents with options for closing the modal and handling user interactions.
10. **$('.el_launch_pdf').on('click', function() {})**
    This event handler sets up functionality to open a PDF document in a modal window when the corresponding element is clicked.
11. **$('body').on('click', '.el_pdf_close', function() {})**
    This event handler provides functionality to close the PDF modal window and save the attempt details if necessary.

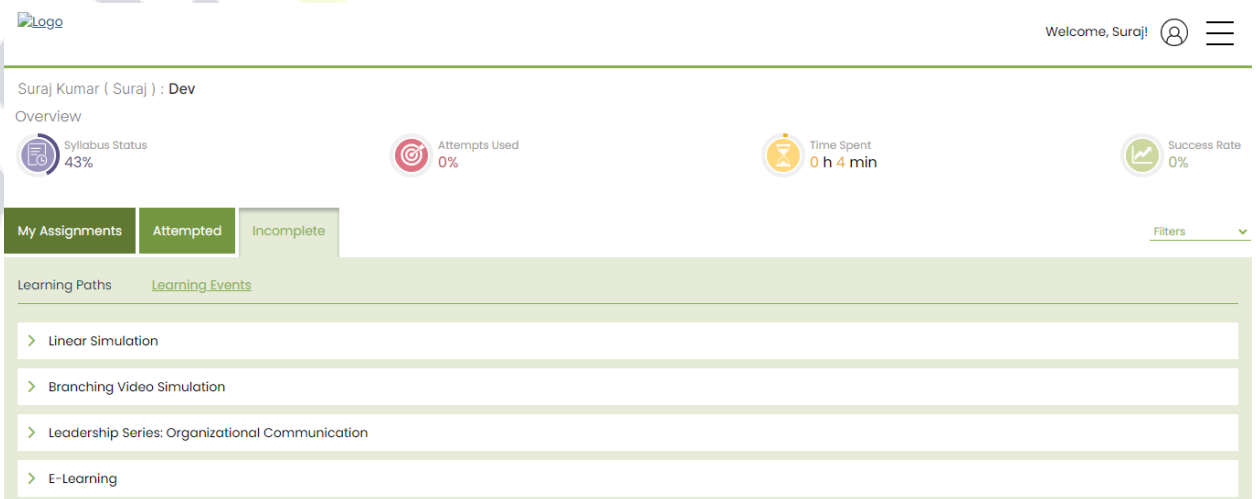
File Path: content/js/learner-assigned-events.js

1. **const getEvents = () => {}**
   This function "getEvents" is designed to retrieve events data from the server. It sends an AJAX POST request to a specified URL (*'includes/learner/events.php'*) with the intention of retrieving events categorized by category.

2. **const textLimit = (string = '', limit = 0) => {}**
   The "textLimit" function is designed to limit the length of a given string to a specified number of characters and append ellipsis (...) if the string exceeds the limit.
3. **const genrateItems = (data, key) => {}**
   The "genrateItems" function generates HTML elements based on the provided data and appends them to the DOM.
4. **const genrateItemsNew = (data, key, id) => {}**
   The "genrateItemsNew" function is similar to genrateItems, but it includes additional functionality to remove an empty event element before appending new items.
5. **$('body').on('click', '.view_event_attachment', function(e){})**
   This code is an event handler for opening a FancyBox modal with an iframe.
6. **$('body').on('click', '.attended', function(e){})**
   This JavaScript code is an event handler for handling the click event on elements with the class .attended.

Incomplete > Learning Events

Template Name: incomplete-events.php



PHP Functions Used

1. **getClientPermission($client_id)**
   Global function from "db.class.php". This function returns all the data for the permissions from table client_permissions for the give client_id.
2. **getAssigneeName($userid)**
   Global function from "db.class.php". This function will return all the data for user as per the User ID passed in the argument.
3. **getAssignPathData((int)$userid))**
   Global function from "db.class.php". This function will return all the data for the learning path for the user.

4. **SQL Queries**
   $catSql = "SELECT * FROM category_tbl WHERE category != '' AND (cat_id IN (SELECT DISTINCT scenario_category FROM scenario_master WHERE scenario_id IN ($getSim) AND scenario_id NOT IN (SELECT scenario_id FROM scenario_attempt_tbl WHERE userid = ?)) OR cat_id IN (SELECT DISTINCT category FROM events AS ev LEFT JOIN group_tbl AS g1 ON FIND_IN_SET(g1.group_id, ev.groups) LEFT JOIN group_tbl AS g2 ON FIND_IN_SET($userid, g1.learner) WHERE FIND_IN_SET($userid, g2.learner) AND FIND_IN_SET($userid, g1.learner) AND ev.id NOT IN (SELECT ev_id FROM events_attend WHERE user_id = $userid))) $corderby";
   The full query retrieves all rows from the category_tbl table where the category column is not empty and meets one of the following conditions:
   A. The cat_id corresponds to a scenario_category in the scenario_master table for scenarios listed in $getSim that the user has not attempted (based on the scenario_attempt_tbl).
   B. The cat_id corresponds to a category in the events table, where the event groups include the user and the event has not been attended by the user (based on the events_attend table).
   Finally, the results can be ordered based on the $corderby clause. This query is likely used to filter and display categories relevant to the user based on their participation and interaction history in scenarios and events.

   $sql = "SELECT DISTINCT scenario_id, sim_ques_type, sim_temp_type, Scenario_title, LEFT(Scenario_title, 50) AS stitle, duration, image_url, video_url, audio_url, web_object_file, scenario_type, scenario_category, author_scenario_type, sim_cover_img, web_object FROM scenario_master WHERE scenario_category = ? AND scenario_id IN ($getSim) AND scenario_id NOT IN (SELECT scenario_id FROM scenario_attempt_tbl WHERE userid = ?)";
   The query retrieves distinct scenarios from the scenario_master table that:
   A. Belong to a specific category.
   B. Have an ID included in a specified list ($getSim).
   C. Have not been attempted by a particular user (specified by the userid).
   It selects various details about each scenario, including its title (both full and truncated to 50 characters), media URLs, type, category, and other attributes.
5. **checkMandatory($sim_id)**
   Global function from “db.class.php”. This function will return all the data from assignment_tbl for the given Simulation ID.
6. **getScenarioTypes(md5($author_scenario_type))**
   Global function from “db.class.php”. This function will return all the different scenario_type for the author.
7. **getTotalAttempt($sim_id, $userid)**
   Global function from “db.class.php”. This function will return total attempts for the user and simulation passed as arguments.
8. **getAssignSimOtherData($sim_id, $userid)**
   Global function from “db.class.php”. This PHP function, "getAssignSimOtherData", is

designed to retrieve assignment data related to a simulation ("$sim_id") for a particular user ("$user_id").

Third Party JS Library

1. ChatApp
2. ChatGPT
3. OwlCarousel
4. DateRangePicker
5. CanvasJS
6. FancyBox
7. FullCalender
8. Tooltip
9. SweetAlert

JS File Used

File Path: content/js/learner-events.js

1. **const getUrlVars = () => {}**
   The "getUrlVars" function extracts query parameters from the current URL and returns them as an array where each parameter is both an item and a property of the array.
2. **const tooltip = () => {}**
   This triggers tooltip functionality on tags with attribute *data-toggle="tooltip".*
3. **const getPercentage = (x, y, html = false) => {}**
   This code defines a JavaScript function named "getPercentage". It calculates the percentage value of "x" in relation to "y", optionally formatting the result as either an integer or a string with a percentage symbol.
4. **const progressBar = () => {}**
   The “progressBar” function iterates over each element with the class progressBar, adds a span element inside it, and sets the width of the span to match the value of the data-progress attribute of the original element
5. **const CircleBar = () => {}**
   The "CircleBar" function initializes circular progress bars on elements with specific classes by setting the "stroke-dashoffset" CSS property based on a data attribute.
6. **const limit = (string = '', limit = 0) => {}**
   This JavaScript code defines a function named "limit". It truncates a given string to a specified limit and appends an ellipsis ("...") if the string exceeds the limit.
7. **const parseTime = (time) => {}**
   This JavaScript code defines a function named "parseTime". It parses a time string in the format "HH:MM" and returns a formatted string displaying the hours and minutes.
8. **const getTotalSumTime = (timeArray, html = false) => {}**
   This JavaScript code defines a function named "getTotalSumTime". It calculates the total

sum of time durations from an array of time strings in the format "HH:MM" and returns the result.

9. **const calculateMinutes = (timeArray) => {}**
   This JavaScript code defines a function named "calculateMinutes". It calculates the total number of minutes from a time string in the format "HH".
10. **const getUsername = (text) => {}**
    This JavaScript code defines a function named "getUsername". It extracts a username from a text string that matches a specific pattern, or generates a random username if no match is found.
11. **const getPDF = (id, title) => {}**
    This JavaScript code defines a function named "getPDF". It generates a PDF file from the content of a specified HTML element and downloads it.
12. **function cb(start, end) {}**
    The provided code defines a function named cb that updates the values of specific input fields based on the start and end dates provided as arguments. This function is handling date ranges in a web application, using a library like jQuery and a date picker plugin.
13. **$('#reportrange').daterangepicker({})**
    This triggers date range picker library for the selection of dates for reports generated.
14. **const setCookie = (name, value, days) => {}**
    This JavaScript code defines a function named "setCookie" used to set a cookie in the browser.
15. **const getCookie = (name) => {}**
    This JavaScript code defines a function named "getCookie" used to retrieve the value of a cookie by its name from the browser's cookies.
16. **$('.launch_sim').on('click', function(e){})**
    This jQuery code snippet sets up an event listener for clicks on elements with the class ".launch_sim". When clicked, it retrieves the value of a custom attribute "sim-id", stores it in a cookie named "'sim_id'", and sets the cookie to expire in 7 days.
17. **$('.sim_list').multiselect({})**
    $('#standing_groups').multiselect({})
    This code initializes a multiselect dropdown menu for elements for the specific class or ID as selector.
18. **const overview = () => {}**
    This JavaScript function overview is responsible for retrieving learner overview data via an AJAX POST request to *'includes/learner/overview-dashboard-data.php'* and updating various elements on the webpage with the received data.
19. **const performanceGrowth = (dataPoint = []) => {}**
    This JavaScript function, "performanceGrowth", is responsible for rendering a column chart using the CanvasJS library. It visualizes the growth percentage over attempts taken based on the provided data points.
20. **const groupsFilterData = (group_id, date, sim_id) => {}**
    This JavaScript function, "groupsFilterData", is responsible for filtering and updating data based on group, date, and simulation ID. It utilizes AJAX POST requests to retrieve data from the server and updates specific elements on the webpage accordingly.

21. **const filterData = (date, sim_id) => {}**
    The "filterData" function is responsible for filtering and updating various elements on a webpage based on the provided date and simulation ID. It makes use of AJAX POST requests to retrieve data from the server and dynamically updates the HTML content of different elements accordingly.
22. **const simList = (date, id, apply = false, clicked = false) => {}**
    The "simList" function is responsible for retrieving and populating a list of simulations based on the provided date and course ID. It utilizes AJAX GET requests to *'includes/learner/overview-dashboard-data.php'* to fetch data from the server and dynamically updates the options of a multiselect dropdown element accordingly.

File Path: content/js/scorm.js

1. **function setupScormApi() {}**
   The "setupScormApi" function is responsible for setting up the SCORM API functionality. It initializes SCORM communication and handles launching SCORM content within a popup window.
2. **function LMSInitialize(initializeInput) {}**
   The "LMSInitialize" function is a part of the SCORM (Sharable Content Object Reference Model) API. It is called to initialize communication between the SCORM content and the SCORM runtime environment (usually a learning management system or LMS).
3. **function LMSGetValue(varname) {}**
   The "LMSGetValue" function is another essential part of the SCORM API. It's called to retrieve the value of a specific data model element (variable) from the SCORM runtime environment (typically an LMS).
4. **function convertTimeString(timeString) {}**
   The "convertTimeString" function is designed to convert a time string into a standardized format of "HH:MM".
5. **function setStatusAndResult() {}**
   The "setStatusAndResult" function is responsible for setting the status and result of a SCORM lesson based on various conditions.
6. **function saveOrUpdate() {}**
   The "saveOrUpdate" function is responsible for saving or updating the SCORM attempt data based on certain conditions.
7. **function resetStatusVars() {}**
   The "resetStatusVars" function is used to reset the status-related variables to their initial values.
8. **function LMSSetValue(varname, varvalue) {}**
   The "LMSSetValue" function is responsible for setting values to SCORM data model elements based on the provided varname and varvalue parameters.
9. **function LMSFinish(finishInput) {}**
   The "LMSFinish" function is responsible for finalizing the SCORM session when the learner finishes interacting with the content.

10. **const base_url = (page = '') => {}**
    The "base_url" function is used to generate the base URL of the current page, optionally appending a specific page or path to it.
11. **const getUid = (sid) => {}**
    The "getUid" function sends AJAX request to *'includes/ajax.php'* with parameter user ID (uid) from the server and retrieves data, then creates an attempt entry in the "scenario_attempt_tbl" table with this "uid" by sending AJAX request to *'api-process.php'*, a given scenario ID (sid), and a user ID (userid). If successful, it sets the attempt ID (aid) and calls another function to handle additional actions.
12. **function getLaunchURL(path) {}**
    The "getLaunchURL" function retrieves the launch URL from an IMS manifest file (*imsmanifest.xml*) located at a given path. This function returns a promise that resolves with the full launch URL.
13. **function getOrgName(path, callback) {}**
    The "getOrgName" function retrieves the organization name and schema version from an IMS manifest file (*imsmanifest.xml*) located at a specified path. This function uses asynchronous operations (await and fetch) to handle the network request and parsing of the XML document.
14. **function msToTime(s) {}**
    The "msToTime" function converts milliseconds to a formatted time string in the format "HH:MM".
15. **function update_attempt_tbl() {}**
    The "update_attempt_tbl" function is responsible for updating the attempt table with relevant information.
16. **function toHoursAndMinutes(totalSeconds) {}**
    The "toHoursAndMinutes" function is used to convert a total number of seconds into hours, minutes, and seconds.
17. **function insert_attempt_tbl() {}**
    The "insert_attempt_tbl" function is responsible for inserting a new record into the attempt table.

File Path: content/js/e-learning.js

1. **const getTimestamp = (duration) => {}**
   The "getTimestamp" function converts a duration in seconds into a formatted timestamp string in the format mm:ss.
2. **const getBaseUrl = (page = '') => {}**
   The "getBaseUrl" function generates the base URL for a web page, optionally appending a specific page path.
3. **const getUID = (sid, userId) => {}**
   The "getUID" function is used to retrieve the unique identifier (UID) associated with a user by sending AJAX request to *'includes/ajax.php'* with parameters "{ get_uid_check_attempts: true, sim_id: sid, uid: userId }" and create a new attempt record

in the database by posting data through AJAX request to *'api-process.php'* for a scenario.

4. **const getFileNameFromUrl = (url) => {}**
   The "getFileNameFromUrl" function is used to extract the file name from a given URL.
5. **const saveAttempt = (ev_aid, file, durationTime, currentTime, userId, type) => {}**
   The saveAttempt function is used to save attempt history related to e-learning activities by sending data through AJAX request to *'api-process.php'*.
6. **$('.el_launch_mp4').on('click', function() {**
   This code handles the launching of MP4 videos, displays them in a modal, tracks user interactions, and saves relevant data about the user's interaction with the video.
7. **const videoModal = (preview, userId, simId, src) => {}**
   This function "videoModal" is responsible for dynamically generating and displaying a modal containing an HTML video player.
8. **$('body').on('click', '.el_video_close', function() {})**
   This listener ensures that when the close button within the video modal is clicked, it handles the necessary actions such as saving the attempt details and closing the modal.
9. **const pdfModal = (preview, userId, simId, src) => {}**
   This function sets up a modal window to display PDF documents with options for closing the modal and handling user interactions.
10. **$('.el_launch_pdf').on('click', function() {})**
    This event handler sets up functionality to open a PDF document in a modal window when the corresponding element is clicked.
11. **$('body').on('click', '.el_pdf_close', function() {})**
    This event handler provides functionality to close the PDF modal window and save the attempt details if necessary.

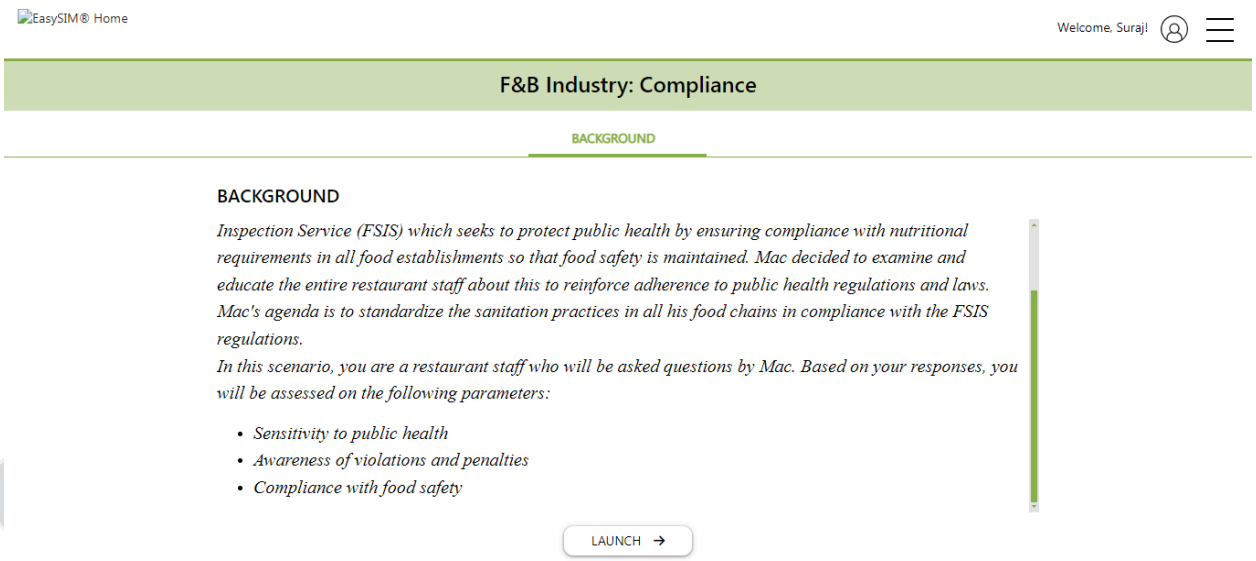
File Path: content/js/learner-assigned-events.js

1. **const getEvents = () => {}**
   This function "getEvents" is designed to retrieve events data from the server. It sends an AJAX POST request to a specified URL (*'includes/learner/events.php'*) with the intention of retrieving events categorized by category.
2. **const textLimit = (string = '', limit = 0) => {}**
   The "textLimit" function is designed to limit the length of a given string to a specified number of characters and append ellipsis (...) if the string exceeds the limit.
3. **const genrateItems = (data, key) => {}**
   The "genrateItems" function generates HTML elements based on the provided data and appends them to the DOM.
4. **const genrateItemsNew = (data, key, id) => {}**
   The "genrateItemsNew" function is similar to genrateItems, but it includes additional functionality to remove an empty event element before appending new items.
5. **$('body').on('click', '.view_event_attachment', function(e){})**
   This code is an event handler for opening a FancyBox modal with an iframe.

6. **$('body').on('click', '.attended', function(e){})**
   This JavaScript code is an event handler for handling the click event on elements with the class .attended.

Simulation > Launch

Template Name: launch.php



This is the intro page for the start of the simulation. It shows all the details/background for the simulation with Launch CTA at the bottom.

PHP Functions Used

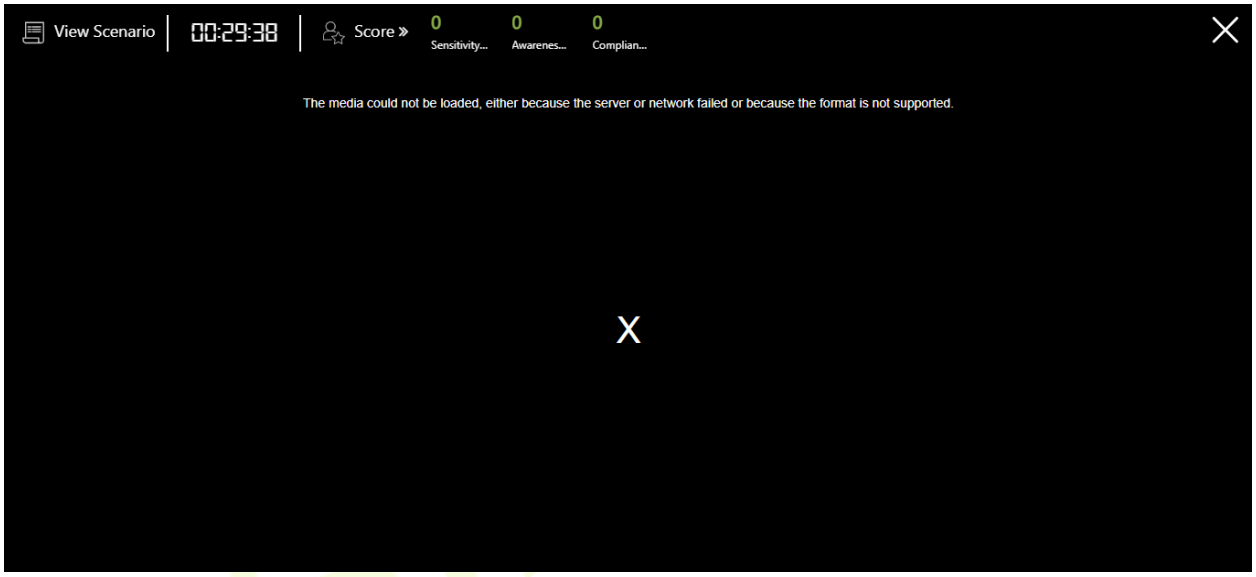
1. **getScenario($sid)**
   Global function from "db.class.php". This PHP function, "getScenario", retrieves scenario data from the database based on the provided scenario ID ($sim_id).
2. **getSimPage($sim_id)**
   Global function from "db.class.php". This PHP function, "getSimPage", retrieves simulation pages associated with a specific scenario ID from the sim_pages table in the database.

JS Function Used

1. **$('body').on('click', '#Verify', function(){})**
   This JavaScript code segment presents an AJAX request to *'face-verification-modal.php'* for loading a face verification modal.

Simulation > Branching Video Classic

Template Name: launch-branching-video-classic.php



PHP Functions Used

1. **getSimBranding($simID)**
   Global function from “db.class.php”. This function, “getSimBranding”, retrieves branding information associated with a specific scenario from the sim_branding_tbl database table.
2. **getScenarioCompetency($simID)**
   Global function from “db.class.php”. This function, "getScenarioCompetency", retrieves competency information associated with a specific scenario from the competency_tbl database table.
3. **webBranding($client_id)**
   Global function from “db.class.php”. The function "webBranding" generates CSS styles and JavaScript for branding a web page based on a client's preferences stored in a database.
4. **UID()**
   Global function from “db.class.php”. The "UID" function generates a unique identifier (UUID) using MySQL's UUID() function. If the query does not return a result, it falls back to generating a unique identifier using md5(time()).
5. **getTotalAttemptedSim($userid)**
   Global function from “db.class.php”. This function will return the total number of Simulations attempted by the user as per the ID passed as argument.
6. **getAssignSimOtherData($simID, $userid)**
   Global function from “db.class.php”. This PHP function, "getAssignSimOtherData", is designed to retrieve assignment data related to a simulation ("$sim_id") for a particular user ("$user_id").
7. **hmsToSeconds($start_time)**
   The "hmsToSeconds" function converts a time string in "minutes" format into the total number of seconds.

8. **Questions Data**
   $dnd_choice_sql = "SELECT question_id, sub_option, image FROM sub_options_tbl WHERE question_id = ?";
   It retrieves data from a database table named sub_options_tbl for options for the question based on a specified question_id.
9. **JS CODE for handling play/pause and display of questions**
   ```
   var vid = '<?= $qid; ?>';
   ....
                    $('.MatchingQusBanner_'+ vid +' #option_'+ box_id).fadeIn(1000);
            });
   }
   ```
   This JavaScript code snippet uses the video.js library to manage video playback and display questions or options at specific times during or after the video.

Third Party JS Library

1. Articulate
2. OwlCarousel

JS File UsedJS File Used

File Path: content/js/launch-sim.js

1. **$('body').on('click', '#view_sim', function () {})**
   This JavaScript code uses jQuery to handle a click event on an element with the ID view_sim. When this element is clicked, it performs an AJAX request to *'view-sim-modal.php'* to load content into a modal window.
2. **$('.close-banner').click(function(){})**
   This JavaScript function using "SweetAlert" and jQuery to confirm the user's intention to close a simulation. If the user confirms, a loading overlay is shown, and after a delay, the user is redirected to a URL specified in the "href" attribute of an element with the ID "close-ban".
3. **$('body').on('click', 'a.qassets_icon', function(e){})**
   This JavaScript code is a jQuery event handler that uses Fancybox to display content when an anchor element with the class "qassets_icon" is clicked. It customizes Fancybox to prevent closing when clicking outside the modal or pressing the ESC key and applies specific transition effects.
4. **$('body').on('click', '.document', function(e){})**
   This JavaScript code is a jQuery event handler that uses Fancybox to display content when an element with the class document is clicked. It customizes Fancybox to display the content in an iframe with specific open and close effects, and ensures the content auto-sizes within the modal.
5. **speak(obj) {}**
   This JavaScript function "speak", controls text-to-speech functionality using the Articulate

plugin. It toggles between speaking, pausing, and resuming speech, and updates the display of play and pause icons accordingly.

6. **speak_feedback(obj) {}**
   The "speak_feedback" function manages text-to-speech functionality specifically for feedback, using the Articulate plugin. It toggles the speaking state (play/pause/resume) and updates the visibility of feedback-specific play and pause icons.

File Path: content/js/sim-branding.js

This JS file simply updates Background color on various elements on the page as per the JSON data fetched from the Server.

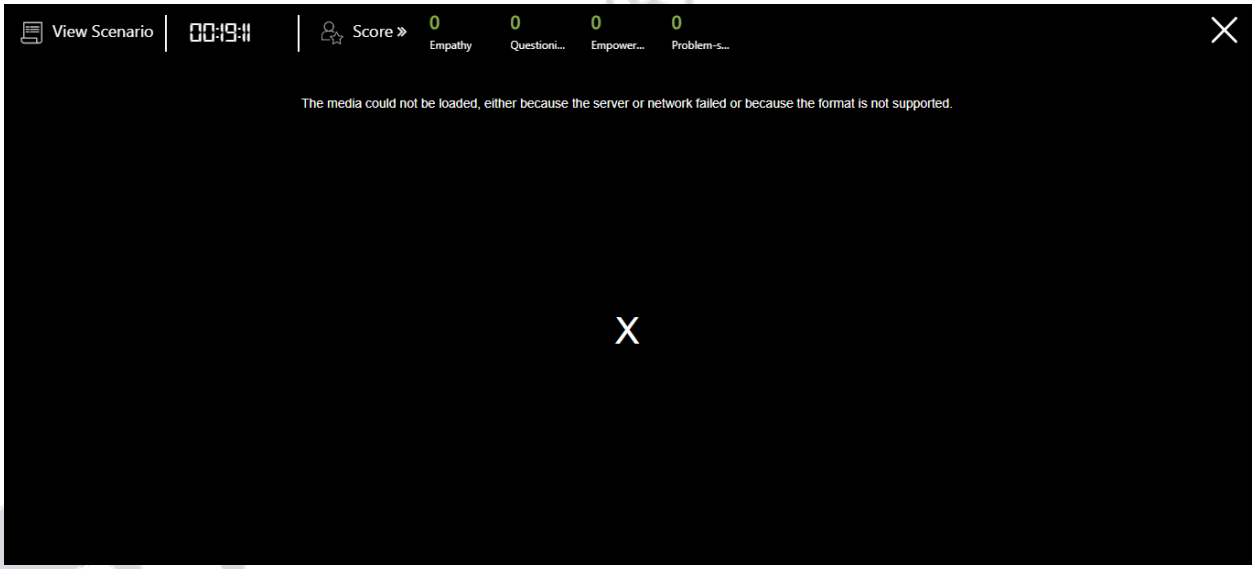
File Path: content/js/sim-temp.js

1. **const QuesTemp = (type, qid, aid = null, next = null, tabindex = null) => {}**
   The "QuesTemp" function is an asynchronous JavaScript function that fetches question template data from a server by sending AJAX request to *'includes/process.php'* and displays it in a modal. It uses jQuery for making AJAX requests and manipulating the DOM.
2. **$('body').on('click', '.launchAddData .naxtPreNav .preBtn', function(e){})**
   This jQuery script listens for a click event on the .preBtn button inside the .naxtPreNav navigation container within the .launchAddData section. When the button is clicked, it decreases the actSlNo variable by 1 to navigate to the previous slide. It also ensures actSlNo does not become negative, and then it calls the MakeSlideActive function to activate the corresponding slide.
3. **$('body').on('click', '.launchAddData .naxtPreNav .nextBtn', function(e){})**
   This jQuery script listens for a click event on an element and handles the navigation to the next slide in a series. It increments the actSlNo variable to move to the next slide, ensures it does not exceed the maximum number of slides, and calls the MakeSlideActive function to activate the corresponding slide.
4. **function MakeSlideActive(index, next = null) {}**
   The "MakeSlideActive" function manages the state of the slides within a modal by adding and removing classes to highlight the active slide. It also enables or disables navigation buttons based on the current slide index and handles special behavior when the last slide is reached.
5. **$('body').on('click', '.feedclose', function(e){})**
   The code defines an event handler for the ".feedclose" element. When this element is clicked, the function performs several actions, including preventing the default action, retrieving data attributes, hiding a modal, resetting a slide counter, and calling a function "QuesTemp" to load new content.
6. **$('body').on('click', '.qftemp_close', function(e){})**
   The code defines an event handler for the ".qftemp_close" element. When this element is clicked, the function performs several actions, including preventing the default action,

retrieving data attributes, hiding a modal, resetting a slide counter, and calling a function "QuesTemp" to load new content.

Simulation > Single Video

Template Name: launch-single-video.php



PHP Functions Used

1. **getScenario($sim_id)**
   Global function from “db.class.php”. This PHP function, "getScenario", retrieves scenario data from the database based on the provided scenario ID ($sim_id).
2. **getSimBranding($simID)**
   Global function from “db.class.php”. This function, “getSimBranding”, retrieves branding information associated with a specific scenario from the sim_branding_tbl database table.
3. **getScenarioCompetency($simID)**
   Global function from “db.class.php”. This function, "getScenarioCompetency", retrieves competency information associated with a specific scenario from the competency_tbl database table.
4. **webBranding($client_id)**
   Global function from “db.class.php”. The function "webBranding" generates CSS styles and JavaScript for branding a web page based on a client's preferences stored in a database.
5. **UID()**
   Global function from “db.class.php”. The "UID" function generates a unique identifier (UUID) using MySQL's UUID() function. If the query does not return a result, it falls back to generating a unique identifier using md5(time()).

6. **getTotalAttempt($simID, $userid)**
   Global function from “db.class.php”. This function will return total attempts for the user and simulation passed as arguments.
7. **getAssignSimOtherData($simID, $userid)**
   Global function from “db.class.php”. This PHP function, "getAssignSimOtherData", is designed to retrieve assignment data related to a simulation ("$sim_id") for a particular user ("$user_id").
8. **Getting Questions and Options**
   *$qSql = $db->prepare("SELECT * FROM question_tbl WHERE scenario_id = ? ORDER BY qorder = 0, qorder");*
   The line of code prepares an SQL statement to select all records from the "question_tbl" table where the "scenario_id" matches a specified value. The results are ordered such that rows with qorder equal to 0 appear first, followed by sorting based on the "qorder" column. This prepared statement can be executed with a specific "scenario_id" value later in the code.
9. **getAnswersByQuestionId(md5($qid), $shuffle)**
   Global function from “db.class.php”. The "getAnswersByQuestionId" function retrieves answers from the answer_tbl table based on a given question ID. It uses the MD5 hash of the question ID to perform the lookup. If the $shuffle parameter is TRUE, the results are returned in a random order. If there are results, they are fetched and returned as an array; otherwise, an empty array is returned.

JS Functions Used

1. **$('body').on('click', '.submitbtn', function() {})**
   a. The script listens for clicks on .submitbtn elements and determines the type of question (MCQ, Swiping, Drag and Drop).
   b. It validates if an option is selected for each type.
   c. Shows a loading overlay during the process.
   d. Loads the feedback modal with appropriate data or displays an error message if no option is selected.

Third Party JS Library

1. ArticulateJS
2. OwlCarousel

JS File Used

File Path: content/js/launch-sim.js

1. **$('body').on('click', '#view_sim', function () {})**
   This JavaScript code uses jQuery to handle a click event on an element with the ID

view_sim. When this element is clicked, it performs an AJAX request to *'view-sim-modal.php'* to load content into a modal window.

2. **$('.close-banner').click(function(){})**
   This JavaScript function using "SweetAlert" and jQuery to confirm the user's intention to close a simulation. If the user confirms, a loading overlay is shown, and after a delay, the user is redirected to a URL specified in the "href" attribute of an element with the ID "close-ban".
3. **$('body').on('click', 'a.qassets_icon', function(e){})**
   This JavaScript code is a jQuery event handler that uses Fancybox to display content when an anchor element with the class "qassets_icon" is clicked. It customizes Fancybox to prevent closing when clicking outside the modal or pressing the ESC key and applies specific transition effects.
4. **$('body').on('click', '.document', function(e){})**
   This JavaScript code is a jQuery event handler that uses Fancybox to display content when an element with the class document is clicked. It customizes Fancybox to display the content in an iframe with specific open and close effects, and ensures the content auto-sizes within the modal.
5. **speak(obj) {}**
   This JavaScript function "speak", controls text-to-speech functionality using the Articulate plugin. It toggles between speaking, pausing, and resuming speech, and updates the display of play and pause icons accordingly.
6. **speak_feedback(obj) {}**
   The "speak_feedback" function manages text-to-speech functionality specifically for feedback, using the Articulate plugin. It toggles the speaking state (play/pause/resume) and updates the visibility of feedback-specific play and pause icons.

File Path: content/js/sim-branding.js

This JS file simply updates Background color on various elements on the page as per the JSON data fetched from the Server.

File Path: content/js/sim-temp.js

1. **const QuesTemp = (type, qid, aid = null, next = null, tabindex = null) => {}**
   The "QuesTemp" function is an asynchronous JavaScript function that fetches question template data from a server by sending AJAX request to *'includes/process.php'* and displays it in a modal. It uses jQuery for making AJAX requests and manipulating the DOM.
2. **$('body').on('click', '.launchAddData .naxtPreNav .preBtn', function(e){})**
   This jQuery script listens for a click event on the .preBtn button inside the .naxtPreNav navigation container within the .launchAddData section. When the button is clicked, it decreases the actSlNo variable by 1 to navigate to the previous slide. It also ensures actSlNo does not become negative, and then it calls the MakeSlideActive function to activate the corresponding slide.

3. **$('body').on('click', '.launchAddData .naxtPreNav .nextBtn', function(e){})**
   This jQuery script listens for a click event on an element and handles the navigation to the next slide in a series. It increments the actSlNo variable to move to the next slide, ensures it does not exceed the maximum number of slides, and calls the MakeSlideActive function to activate the corresponding slide.
4. **function MakeSlideActive(index, next = null) {}**
   The "MakeSlideActive" function manages the state of the slides within a modal by adding and removing classes to highlight the active slide. It also enables or disables navigation buttons based on the current slide index and handles special behavior when the last slide is reached.
5. **$('body').on('click', '.feedclose', function(e){})**
   The code defines an event handler for the ".feedclose" element. When this element is clicked, the function performs several actions, including preventing the default action, retrieving data attributes, hiding a modal, resetting a slide counter, and calling a function "QuesTemp" to load new content.
6. **$('body').on('click', '.qftemp_close', function(e){})**
   The code defines an event handler for the ".qftemp_close" element. When this element is clicked, the function performs several actions, including preventing the default action, retrieving data attributes, hiding a modal, resetting a slide counter, and calling a function "QuesTemp" to load new content.

Simulation > Branching Video Multiple Classic

Template Name: launch-branching-video-multiple.php

PHP Functions Used

1. **getScenario($sim_id)**
   Global function from “db.class.php”. This PHP function, "getScenario", retrieves scenario data from the database based on the provided scenario ID ($sim_id).

2. **getSimBranding($simID)**
   Global function from “db.class.php”. This function, “getSimBranding”, retrieves branding information associated with a specific scenario from the sim_branding_tbl database table.
3. **getScenarioCompetency($simID)**
   Global function from “db.class.php”. This function, "getScenarioCompetency", retrieves competency information associated with a specific scenario from the competency_tbl database table.
4. **webBranding($client_id)**
   Global function from “db.class.php”. The function "webBranding" generates CSS styles and JavaScript for branding a web page based on a client's preferences stored in a database.
5. **UID()**
   Global function from “db.class.php”. The "UID" function generates a unique identifier (UUID) using MySQL's UUID() function. If the query does not return a result, it falls back to generating a unique identifier using md5(time()).
6. **getTotalAttempt($simID, $userid)**
   Global function from “db.class.php”. This function will return total attempts for the user and simulation passed as arguments.
7. **getAssignSimOtherData($simID, $userid)**
   Global function from “db.class.php”. This PHP function, "getAssignSimOtherData", is designed to retrieve assignment data related to a simulation ("$sim_id") for a particular user ("$user_id").
8. **Getting Questions and Options**
   *$qSql = $db->prepare("SELECT * FROM question_tbl WHERE scenario_id = ? ORDER BY qorder = 0, qorder");*
   The line of code prepares an SQL statement to select all records from the "question_tbl" table where the "scenario_id" matches a specified value. The results are ordered such that rows with qorder equal to 0 appear first, followed by sorting based on the "qorder" column. This prepared statement can be executed with a specific "scenario_id" value later in the code.

   $mtf_choice_sql = "SELECT answer_id, question_id, choice_option, match_option FROM answer_tbl WHERE question_id = ? ORDER BY (CASE WHEN choice_order_no != 0 THEN choice_order_no WHEN choice_order_no = 0 THEN RAND() END)";
   This SQL query retrieves information from the answer_tbl table related to multiple-choice matching type questions (often abbreviated as MTF or Matching Type Format).

   $mtf_match_sql = "SELECT answer_id, match_option FROM answer_tbl WHERE question_id = ? ORDER BY (CASE WHEN match_order_no != 0 THEN match_order_no WHEN match_order_no = 0 THEN RAND() END)";
   The query retrieves the answer_id and match_option columns from the answer_tbl table for rows where the question_id matches a provided value. The results are ordered such that rows with a non-zero match_order_no are sorted by that value, while rows with a

match_order_no of 0 are placed in a random order. This approach ensures that certain matches appear in a specific order, while others appear randomly, which can be useful in scenarios where some items need a fixed order while others do not.

$sq_sql = "SELECT answer_id, question_id, choice_option FROM answer_tbl WHERE question_id = ? ORDER BY RAND()";
The query retrieves the answer_id, question_id, and choice_option columns from the answer_tbl table for rows where the question_id matches a provided value. The results are then randomly ordered each time the query is executed. This approach is useful when you want to retrieve random rows from a table, such as in situations where you want to display random quiz questions or random choices to users.

$sort_choice_sql = "SELECT answer_id, question_id, choice_option, image FROM answer_tbl WHERE question_id = ?";
The query selects the answer_id, question_id, choice_option, and image columns from the answer_tbl table for rows where the question_id matches a provided value. This allows you to retrieve information about the choices for a specific question.

$sort_option_sql = "SELECT sub.sub_options_id, sub.sub_option FROM sub_options_tbl AS sub LEFT JOIN answer_tbl AS ans ON sub.answer_id = ans.answer_id WHERE ans.question_id = ? ".((! empty($shuffle)) ? "ORDER BY RAND() ": "");
The query retrieves the sub_options_id and sub_option columns from the sub_options_tbl table, joined with the answer_tbl table to retrieve options related to a specific question. If the $shuffle variable is not empty, the results are randomly ordered; otherwise, they are returned without any specific order.

$dnd_choice_sql = "SELECT question_id, sub_option, image FROM sub_options_tbl WHERE question_id = ?";
It retrieves data from a database table named sub_options_tbl for options for the question based on a specified question_id.

$dnd2_choice_sql = "SELECT answer_id, question_id, image, choice_option FROM answer_tbl WHERE question_id = ? ".(($shuffle == TRUE) ? "ORDER BY RAND()" : "");
The query selects the answer_id, question_id, image, and choice_option columns from the answer_tbl table for rows where the question_id matches a provided value. If the $shuffle variable is TRUE, the results are randomly ordered; otherwise, they are returned without any specific order. This approach allows for flexibility in sorting the results based on a condition.

$sql2dnd = "SELECT answer_id, image2, match_option FROM answer_tbl WHERE question_id = ? ".(($shuffle == TRUE) ? "ORDER BY RAND()" : "");
The query selects the answer_id, image2, and match_option columns from the answer_tbl table for rows where the question_id matches a provided value. If the

$shuffle variable is TRUE, the results are randomly ordered; otherwise, they are returned without any specific order. This approach provides flexibility in sorting the results based on a condition.

9. **getAnswersByQuestionId(md5($qid), $shuffle)**
   Global function from "db.class.php". The "getAnswersByQuestionId" function retrieves answers from the answer_tbl table based on a given question ID. It uses the MD5 hash of the question ID to perform the lookup. If the $shuffle parameter is TRUE, the results are returned in a random order. If there are results, they are fetched and returned as an array; otherwise, an empty array is returned.

Third Party JS Library

1. Articulate
2. OwlCarousel

JS Functions

1. **$('body').on('click', '.submitbtn', function(){})**
   This JavaScript/jQuery code is designed to handle the click event on elements with the class .submitbtn within the body. The code processes different types of questions and interacts with a modal to display the results or feedback.
2. **$('.dragdrop').draggable({})**
   This JavaScript/jQuery code makes elements with the class .dragdrop draggable and defines the behavior for dropping these elements onto designated drop targets.
3. **$('.dragdrop1').draggable({})**
   This JavaScript/jQuery code makes elements with the class .dragdrop1 draggable and defines their behavior when dropped onto designated drop targets.

JS File Used

File Path: content/js/launch-sim.js

1. **$('body').on('click', '#view_sim', function () {})**
   This JavaScript code uses jQuery to handle a click event on an element with the ID view_sim. When this element is clicked, it performs an AJAX request to *'view-sim-modal.php'* to load content into a modal window.
2. **$('.close-banner').click(function(){})**
   This JavaScript function using "SweetAlert" and jQuery to confirm the user's intention to close a simulation. If the user confirms, a loading overlay is shown, and after a delay, the user is redirected to a URL specified in the "href" attribute of an element with the ID "close-ban".
3. **$('body').on('click', 'a.qassets_icon', function(e){})**
   This JavaScript code is a jQuery event handler that uses Fancybox to display content when an anchor element with the class "qassets_icon" is clicked. It customizes

Fancybox to prevent closing when clicking outside the modal or pressing the ESC key and applies specific transition effects.

4. **$('body').on('click', '.document', function(e){})**
   This JavaScript code is a jQuery event handler that uses Fancybox to display content when an element with the class document is clicked. It customizes Fancybox to display the content in an iframe with specific open and close effects, and ensures the content auto-sizes within the modal.
5. **speak(obj) {}**
   This JavaScript function "speak", controls text-to-speech functionality using the Articulate plugin. It toggles between speaking, pausing, and resuming speech, and updates the display of play and pause icons accordingly.
6. **speak_feedback(obj) {}**
   The "speak_feedback" function manages text-to-speech functionality specifically for feedback, using the Articulate plugin. It toggles the speaking state (play/pause/resume) and updates the visibility of feedback-specific play and pause icons.

File Path: content/js/sim-branding.js

This JS file simply updates Background color on various elements on the page as per the JSON data fetched from the Server.

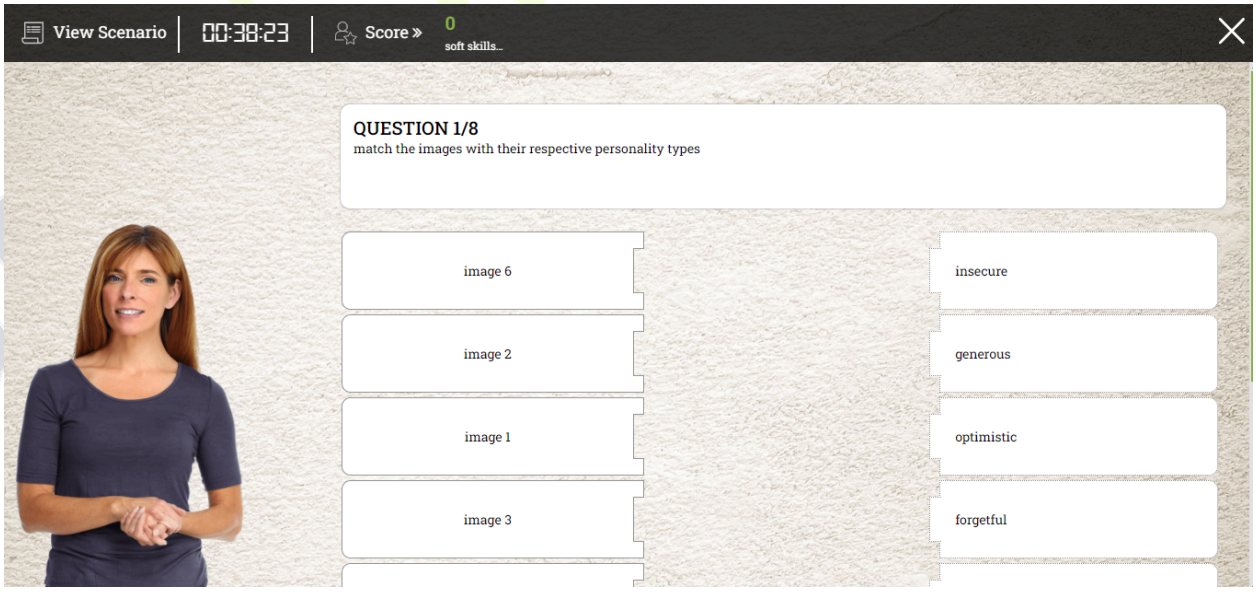
File Path: content/js/sim-temp.js

1. **const QuesTemp = (type, qid, aid = null, next = null, tabindex = null) => {}**
   The "QuesTemp" function is an asynchronous JavaScript function that fetches question template data from a server by sending AJAX request to *'includes/process.php'* and displays it in a modal. It uses jQuery for making AJAX requests and manipulating the DOM.
2. **$('body').on('click', '.launchAddData .naxtPreNav .preBtn', function(e){})**
   This jQuery script listens for a click event on the .preBtn button inside the .naxtPreNav navigation container within the .launchAddData section. When the button is clicked, it decreases the actSlNo variable by 1 to navigate to the previous slide. It also ensures actSlNo does not become negative, and then it calls the MakeSlideActive function to activate the corresponding slide.
3. **$('body').on('click', '.launchAddData .naxtPreNav .nextBtn', function(e){})**
   This jQuery script listens for a click event on an element and handles the navigation to the next slide in a series. It increments the actSlNo variable to move to the next slide, ensures it does not exceed the maximum number of slides, and calls the MakeSlideActive function to activate the corresponding slide.
4. **function MakeSlideActive(index, next = null) {}**
   The "MakeSlideActive" function manages the state of the slides within a modal by adding and removing classes to highlight the active slide. It also enables or disables navigation buttons based on the current slide index and handles special behavior when the last slide is reached.

5. **$('body').on('click', '.feedclose', function(e){})**
   The code defines an event handler for the ".feedclose" element. When this element is clicked, the function performs several actions, including preventing the default action, retrieving data attributes, hiding a modal, resetting a slide counter, and calling a function "QuesTemp" to load new content.
6. **$('body').on('click', '.qftemp_close', function(e){})**
   The code defines an event handler for the ".qftemp_close" element. When this element is clicked, the function performs several actions, including preventing the default action, retrieving data attributes, hiding a modal, resetting a slide counter, and calling a function "QuesTemp" to load new content.

Simulation > Linear Multiple

Template Name: launch-sim-linear-multiple.php



PHP Functions Used

1. **getScenario($sim_id)**
   Global function from "db.class.php". This PHP function, "getScenario", retrieves scenario data from the database based on the provided scenario ID ($sim_id).
2. **getSimBranding($simID)**
   Global function from "db.class.php". This function, "getSimBranding", retrieves branding information associated with a specific scenario from the sim_branding_tbl database table.
3. **getScenarioCompetency($simID)**
   Global function from "db.class.php". This function, "getScenarioCompetency", retrieves competency information associated with a specific scenario from the competency_tbl database table.

4. **webBranding($client_id)**
   Global function from “db.class.php”. The function "webBranding" generates CSS styles and JavaScript for branding a web page based on a client's preferences stored in a database.
5. **UID()**
   Global function from “db.class.php”. The "UID" function generates a unique identifier (UUID) using MySQL's UUID() function. If the query does not return a result, it falls back to generating a unique identifier using md5(time()).
6. **getTotalAttempt($simID, $userid)**
   Global function from “db.class.php”. This function will return total attempts for the user and simulation passed as arguments.
7. **getAssignSimOtherData($simID, $userid)**
   Global function from “db.class.php”. This PHP function, "getAssignSimOtherData", is designed to retrieve assignment data related to a simulation ("$sim_id") for a particular user ("$user_id").
8. **Getting Questions and Options**
   *$qSql = $db->prepare("SELECT * FROM question_tbl WHERE scenario_id = ? ORDER BY qorder = 0, qorder");*
   The line of code prepares an SQL statement to select all records from the "question_tbl" table where the "scenario_id" matches a specified value. The results are ordered such that rows with qorder equal to 0 appear first, followed by sorting based on the "qorder" column. This prepared statement can be executed with a specific "scenario_id" value later in the code.

   $mtf_choice_sql = "SELECT answer_id, question_id, choice_option, match_option FROM answer_tbl WHERE question_id = ? ORDER BY (CASE WHEN choice_order_no != 0 THEN choice_order_no WHEN choice_order_no = 0 THEN RAND() END)";
   The query retrieves specific columns from the "answer_tbl" table for rows where the "question_id" matches a provided value. The results are ordered such that rows with a non-zero "choice_order_no" are sorted by that value, while rows with a "choice_order_no" of 0 are placed in a random order. This is useful in scenarios where some choices should appear in a specific order, and others should appear randomly.

   $mtf_match_sql = "SELECT answer_id, match_option FROM answer_tbl WHERE question_id = ? ORDER BY (CASE WHEN match_order_no != 0 THEN match_order_no WHEN match_order_no = 0 THEN RAND() END)";
   The query retrieves the answer_id and match_option columns from the answer_tbl table for rows where the question_id matches a provided value. The results are ordered such that rows with a non-zero match_order_no are sorted by that value, while rows with a match_order_no of 0 are placed in a random order. This approach ensures that certain matches appear in a specific order, while others appear randomly, which can be useful in scenarios where some items need a fixed order while others do not.

   $sq_sql = "SELECT answer_id, question_id, choice_option FROM answer_tbl WHERE

question_id = ? ORDER BY RAND()";
The query retrieves the answer_id, question_id, and choice_option columns from the answer_tbl table for rows where the question_id matches a provided value. The results are then randomly ordered each time the query is executed. This approach is useful when you want to retrieve random rows from a table, such as in situations where you want to display random quiz questions or random choices to users.

$sort_choice_sql = "SELECT answer_id, question_id, choice_option, image FROM answer_tbl WHERE question_id = ?";
The query selects the answer_id, question_id, choice_option, and image columns from the answer_tbl table for rows where the question_id matches a provided value. This allows you to retrieve information about the choices for a specific question.

$sort_option_sql = "SELECT sub.sub_options_id, sub.sub_option FROM sub_options_tbl AS sub LEFT JOIN answer_tbl AS ans ON sub.answer_id = ans.answer_id WHERE ans.question_id = ? ".((! empty($shuffle)) ? "ORDER BY RAND() ": "");
The query retrieves the sub_options_id and sub_option columns from the sub_options_tbl table, joined with the answer_tbl table to retrieve options related to a specific question. If the $shuffle variable is not empty, the results are randomly ordered; otherwise, they are returned without any specific order.

$dnd_choice_sql = "SELECT question_id, sub_option, image FROM sub_options_tbl WHERE question_id = ?";
It retrieves data from a database table named sub_options_tbl for options for the question based on a specified question_id.

$dnd2_choice_sql = "SELECT answer_id, question_id, image, choice_option FROM answer_tbl WHERE question_id = ? ".(($shuffle == TRUE) ? "ORDER BY RAND()" : "");
The query selects the answer_id, question_id, image, and choice_option columns from the answer_tbl table for rows where the question_id matches a provided value. If the $shuffle variable is TRUE, the results are randomly ordered; otherwise, they are returned without any specific order. This approach allows for flexibility in sorting the results based on a condition.

"SELECT answer_id, image2, match_option FROM answer_tbl WHERE question_id = ? ".(($shuffle == TRUE) ? "ORDER BY RAND()" : "");
The query selects the answer_id, image2, and match_option columns from the answer_tbl table for rows where the question_id matches a provided value. If the $shuffle variable is TRUE, the results are randomly ordered; otherwise, they are returned without any specific order. This approach provides flexibility in sorting the results based on a condition.

Third Party JS Library

1. Articulate
2. OwlCarousel

JS Functions Used

1. **$('.dragdrop').draggable({})**
   The provided JavaScript/jQuery code snippet sets up draggable functionality for elements with the class dragdrop and defines their behavior when dropped onto elements with the class drop.
2. **$('.dragdrop1').draggable({})**
   This code sets up draggable functionality for elements with the class dragdrop1, allowing them to be dragged and dropped onto elements with the class drop1. Upon dropping, it appends the draggable item to the droptarget, makes it draggable again, removes any empty .owl-item elements, and refreshes the Owl Carousel with the class COURSALbtnSty and swipecoursal.
3. **$('body').on('click', '.submitbtn', function() {})**
   a. The script listens for clicks on .submitbtn elements and determines the type of question (MCQ, Swiping, Drag and Drop).
   b. It validates if an option is selected for each type.
   c. Shows a loading overlay during the process.
   d. Loads the feedback modal with appropriate data or displays an error message if no option is selected.

JS File Used

File Path: content/js/launch-sim.js

1. **$('body').on('click', '#view_sim', function () {})**
   This JavaScript code uses jQuery to handle a click event on an element with the ID view_sim. When this element is clicked, it performs an AJAX request to *'view-sim-modal.php'* to load content into a modal window.
2. **$('.close-banner').click(function(){})**
   This JavaScript function using "SweetAlert" and jQuery to confirm the user's intention to close a simulation. If the user confirms, a loading overlay is shown, and after a delay, the user is redirected to a URL specified in the "href" attribute of an element with the ID "close-ban".
3. **$('body').on('click', 'a.qassets_icon', function(e){})**
   This JavaScript code is a jQuery event handler that uses Fancybox to display content when an anchor element with the class "qassets_icon" is clicked. It customizes Fancybox to prevent closing when clicking outside the modal or pressing the ESC key and applies specific transition effects.
4. **$('body').on('click', '.document', function(e){})**
   This JavaScript code is a jQuery event handler that uses Fancybox to display content

when an element with the class document is clicked. It customizes Fancybox to display the content in an iframe with specific open and close effects, and ensures the content auto-sizes within the modal.

5. **speak(obj) {}**
   This JavaScript function "speak", controls text-to-speech functionality using the Articulate plugin. It toggles between speaking, pausing, and resuming speech, and updates the display of play and pause icons accordingly.
6. **speak_feedback(obj) {}**
   The "speak_feedback" function manages text-to-speech functionality specifically for feedback, using the Articulate plugin. It toggles the speaking state (play/pause/resume) and updates the visibility of feedback-specific play and pause icons.

File Path: content/js/sim-branding.js

This JS file simply updates Background color on various elements on the page as per the JSON data fetched from the Server.

File Path: content/js/sim-temp.js

1. **const QuesTemp = (type, qid, aid = null, next = null, tabindex = null) => {}**
   The "QuesTemp" function is an asynchronous JavaScript function that fetches question template data from a server by sending AJAX request to *'includes/process.php'* and displays it in a modal. It uses jQuery for making AJAX requests and manipulating the DOM.
2. **$('body').on('click', '.launchAddData .naxtPreNav .preBtn', function(e){})**
   This jQuery script listens for a click event on the .preBtn button inside the .naxtPreNav navigation container within the .launchAddData section. When the button is clicked, it decreases the actSlNo variable by 1 to navigate to the previous slide. It also ensures actSlNo does not become negative, and then it calls the MakeSlideActive function to activate the corresponding slide.
3. **$('body').on('click', '.launchAddData .naxtPreNav .nextBtn', function(e){})**
   This jQuery script listens for a click event on an element and handles the navigation to the next slide in a series. It increments the actSlNo variable to move to the next slide, ensures it does not exceed the maximum number of slides, and calls the MakeSlideActive function to activate the corresponding slide.
4. **function MakeSlideActive(index, next = null) {}**
   The "MakeSlideActive" function manages the state of the slides within a modal by adding and removing classes to highlight the active slide. It also enables or disables navigation buttons based on the current slide index and handles special behavior when the last slide is reached.
5. **$('body').on('click', '.feedclose', function(e){})**
   The code defines an event handler for the ".feedclose" element. When this element is clicked, the function performs several actions, including preventing the default action,

retrieving data attributes, hiding a modal, resetting a slide counter, and calling a function "QuesTemp" to load new content.

6. **$('body').on('click', '.qftemp_close', function(e){})**
   The code defines an event handler for the ".qftemp_close" element. When this element is clicked, the function performs several actions, including preventing the default action, retrieving data attributes, hiding a modal, resetting a slide counter, and calling a function "QuesTemp" to load new content.

Simulation > Linear Classic

Template Name: launch-sim-linear-classic.php

ImageToBePlacedHere

PHP Functions Used

1. **getScenario($sim_id)**
   Global function from "db.class.php". This PHP function, "getScenario", retrieves scenario data from the database based on the provided scenario ID ($sim_id).
2. **getSimBranding($simID)**
   Global function from "db.class.php". This function, "getSimBranding", retrieves branding information associated with a specific scenario from the sim_branding_tbl database table.
3. **getScenarioCompetency($simID)**
   Global function from "db.class.php". This function, "getScenarioCompetency", retrieves competency information associated with a specific scenario from the competency_tbl database table.
4. **webBranding($client_id)**
   Global function from "db.class.php". The function "webBranding" generates CSS styles and JavaScript for branding a web page based on a client's preferences stored in a database.
5. **UID()**
   Global function from "db.class.php". The "UID" function generates a unique identifier (UUID) using MySQL's UUID() function. If the query does not return a result, it falls back to generating a unique identifier using md5(time()).
6. **getTotalAttempt($simID, $userid)**
   Global function from "db.class.php". This function will return total attempts for the user and simulation passed as arguments.
7. **getAssignSimOtherData($simID, $userid)**
   Global function from "db.class.php". This PHP function, "getAssignSimOtherData", is designed to retrieve assignment data related to a simulation ("$sim_id") for a particular user ("$user_id").
8. **Getting Questions and Options**
   *$qSql = $db->prepare("SELECT * FROM question_tbl WHERE scenario_id = ? ORDER BY qorder = 0, qorder");*

The line of code prepares an SQL statement to select all records from the "question_tbl" table where the "scenario_id" matches a specified value. The results are ordered such that rows with qorder equal to 0 appear first, followed by sorting based on the "qorder" column. This prepared statement can be executed with a specific "scenario_id" value later in the code.

$dnd_choice_sql = "SELECT question_id, sub_option, image FROM sub_options_tbl WHERE question_id = ?";
The SQL query is designed to fetch specific data (question_id, sub_option, image) from the sub_options_tbl table, filtered by a question_id.

9. **getAnswersByQuestionId(md5($qid), $shuffle)**
   Global function from "db.class.php". The "getAnswersByQuestionId" function retrieves answers from the answer_tbl table based on a given question ID. It uses the MD5 hash of the question ID to perform the lookup. If the $shuffle parameter is TRUE, the results are returned in a random order. If there are results, they are fetched and returned as an array; otherwise, an empty array is returned.

Third Party JS Library

1. Articulate
2. OwlCarousel

JS Functions Used

1. **$('.dragdrop').draggable({})**
   This JavaScript/jQuery code snippet enables elements with the class .dragdrop to be draggable and defines their behavior when dropped onto designated drop targets.
2. **$('body').on('click', '.submitbtn', function() {})**
   This JavaScript/jQuery code snippet adds a click event listener to elements with the class .submitbtn and processes user interactions based on the type of question (MCQ, swiping, or drag-and-drop) present in an educational or quiz-like interface.

JS File Used

File Path: content/js/launch-sim.js

1. **$('body').on('click', '#view_sim', function () {})**
   This JavaScript code uses jQuery to handle a click event on an element with the ID view_sim. When this element is clicked, it performs an AJAX request to *'view-sim-modal.php'* to load content into a modal window.
2. **$('.close-banner').click(function(){})**
   This JavaScript function using "SweetAlert" and jQuery to confirm the user's intention to close a simulation. If the user confirms, a loading overlay is shown, and after a delay, the

user is redirected to a URL specified in the "href" attribute of an element with the ID "close-ban".

3. **$('body').on('click', 'a.qassets_icon', function(e){})**
   This JavaScript code is a jQuery event handler that uses Fancybox to display content when an anchor element with the class "qassets_icon" is clicked. It customizes Fancybox to prevent closing when clicking outside the modal or pressing the ESC key and applies specific transition effects.
4. **$('body').on('click', '.document', function(e){})**
   This JavaScript code is a jQuery event handler that uses Fancybox to display content when an element with the class document is clicked. It customizes Fancybox to display the content in an iframe with specific open and close effects, and ensures the content auto-sizes within the modal.
5. **speak(obj) {}**
   This JavaScript function "speak", controls text-to-speech functionality using the Articulate plugin. It toggles between speaking, pausing, and resuming speech, and updates the display of play and pause icons accordingly.
6. **speak_feedback(obj) {}**
   The "speak_feedback" function manages text-to-speech functionality specifically for feedback, using the Articulate plugin. It toggles the speaking state (play/pause/resume) and updates the visibility of feedback-specific play and pause icons.

File Path: content/js/sim-branding.js

This JS file simply updates Background color on various elements on the page as per the JSON data fetched from the Server.

File Path: content/js/sim-temp.js

1. **const QuesTemp = (type, qid, aid = null, next = null, tabindex = null) => {}**
   The "QuesTemp" function is an asynchronous JavaScript function that fetches question template data from a server by sending AJAX request to *'includes/process.php'* and displays it in a modal. It uses jQuery for making AJAX requests and manipulating the DOM.
2. **$('body').on('click', '.launchAddData .naxtPreNav .preBtn', function(e){})**
   This jQuery script listens for a click event on the .preBtn button inside the .naxtPreNav navigation container within the .launchAddData section. When the button is clicked, it decreases the actSlNo variable by 1 to navigate to the previous slide. It also ensures actSlNo does not become negative, and then it calls the MakeSlideActive function to activate the corresponding slide.
3. **$('body').on('click', '.launchAddData .naxtPreNav .nextBtn', function(e){})**
   This jQuery script listens for a click event on an element and handles the navigation to the next slide in a series. It increments the actSlNo variable to move to the next slide, ensures it does not exceed the maximum number of slides, and calls the MakeSlideActive function to activate the corresponding slide.

4. **function MakeSlideActive(index, next = null) {}**
   The "MakeSlideActive" function manages the state of the slides within a modal by adding and removing classes to highlight the active slide. It also enables or disables navigation buttons based on the current slide index and handles special behavior when the last slide is reached.
5. **$('body').on('click', '.feedclose', function(e){})**
   The code defines an event handler for the ".feedclose" element. When this element is clicked, the function performs several actions, including preventing the default action, retrieving data attributes, hiding a modal, resetting a slide counter, and calling a function "QuesTemp" to load new content.
6. **$('body').on('click', '.qftemp_close', function(e){})**
   The code defines an event handler for the ".qftemp_close" element. When this element is clicked, the function performs several actions, including preventing the default action, retrieving data attributes, hiding a modal, resetting a slide counter, and calling a function "QuesTemp" to load new content.

File Path: content/js/track-assets.js

1. **const getTimestamp = (duration) => {}**
   The “getTimestamp” function converts a duration in seconds into a formatted string representing minutes and seconds.
2. **$('body').on('click', '.track_assets', function(e) {})**
   This jQuery code snippet sets up a click event listener on elements with the class .track_assets. Upon clicking, it tracks various types of assets (such as images, audio, video, iframes, and text-to-speech) in an educational or interactive context.

File Path: content/js/proctor-api.js

1. **function snapshot_valid(profile_url, snapshot_url) {}**
   This function conducts a snapshot validation process by sending a POST request to a specified endpoint (*profile_url*) with the provided profile and snapshot URLs. It handles different scenarios such as successful validation, validation failure, and errors encountered during the process, displaying appropriate messages to the user using a library like swal (possibly SweetAlert).
2. **function matchProgress(param) {}**
   The matchProgress function is used to visually represent the progress of a match operation, such as facial recognition. It dynamically updates a progress bar element on the page to reflect the current progress percentage. The function runs periodically until the progress reaches the specified percentage (param). This feature provides users with real-time feedback on the progress of the match operation.
3. **function attention_analysis(filename, video_url, profile_url) {}**
   The attention_analysis function facilitates the analysis of attention-related aspects in video content. It sends an AJAX request to a specified endpoint (*video_url*) with the necessary data, including the video's filename, URL, and the user's profile picture URL.

Upon successful analysis, it logs the response data to the console, and in case of an error, it displays an error message to the user.

4. **function attention_analysis_report(filename) {}**
   The attention_analysis_report function fetches the attention analysis report for a given video file. It sends an AJAX request to a specified endpoint, receives and processes the response data, and updates the DOM accordingly to display the report and associated charts. If there are any issues during the process, it handles them by displaying appropriate messages to the user.
5. **function group_attention_analysis_report(array, user_data) {}**
   The group_attention_analysis_report function fetches a group attention analysis report for a given array of video IDs. It sends an AJAX request to a specified endpoint, receives and processes the response data, and populates the report table with relevant information, including user data associated with each video ID. If there are any issues during the process, it handles them by displaying an appropriate error message to the user.

File Path: content/js/recording.js

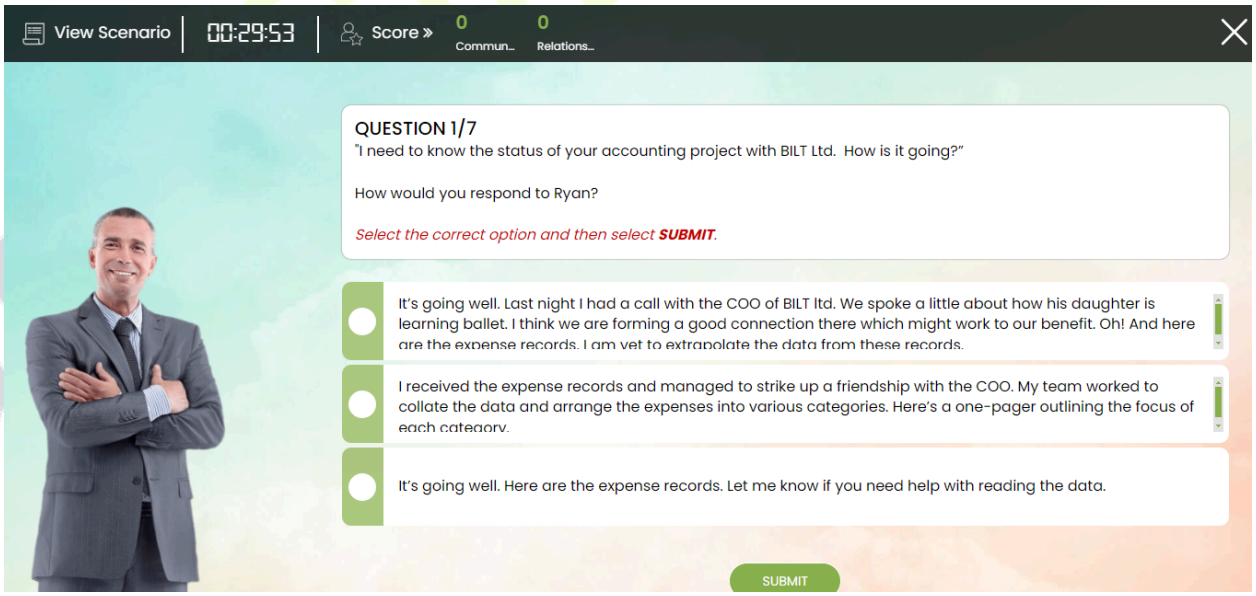
1. **var player = videojs('myVideo', options, function() {})**
   The code initializes a Video.js player on a specified HTML element with the given ID, applies additional options, and executes a callback function after initialization to log version information about Video.js, videojs-record plugin, and RecordRTC library.
2. **player.on('deviceError', function() {})**
   This code listens for device errors that may occur during the usage of the Video.js player. When a device error occurs, it displays the error code to the user using a SweetAlert modal, providing a visual indication of the error and its type.
3. **player.on('error', function(element, error) {})**
   This code listens for errors that may occur during the usage of the Video.js player. When an error occurs, it displays the error message to the user using a SweetAlert modal, providing a visual indication of the error.
4. **player.on('ready', function() {})**
   This code ensures that when the Video.js player is fully loaded and ready for use, it initializes recording functionality and retrieves the available media input devices. This setup is typically used to prepare the player for recording video and audio.
5. **player.on('deviceReady', function() {})**
   This code ensures that when the recording devices are ready for use, it starts the recording process on the Video.js player. It's typically used to automatically start recording once the necessary devices are initialized and ready.
6. **player.on('startRecord', function() {})**
   This code logs a message to the console when recording starts on the Video.js player. It's useful for debugging or providing feedback to users when recording begins.
7. **$('body').on('click', '.swal-button--confirm', function(e){})**
   This event handler is likely used in conjunction with a modal dialog (possibly a SweetAlert modal) to handle a user action (e.g., confirmation) and perform some action

(e.g., stopping a recording stream) while preventing the default behavior of the click event and stopping its propagation.

8. **player.on('timestamp', function() {})**
   This event listener is triggered when a timestamp event occurs, and if a specific element is visible, it stops the recording stream by calling the stopRecStream() function.
9. **player.on('finishRecord', function() {})**
   This event listener handles the finalization of video recording, uploads the recorded data to a server, processes the server response, and initiates further analysis or processing on the uploaded video data.

Simulation > Branching Classic

Template Name: launch-sim-branching-classic.php



PHP Functions Used

1. **getScenario($sim_id)**
   Global function from "db.class.php". This PHP function, "getScenario", retrieves scenario data from the database based on the provided scenario ID ($sim_id).
2. **getSimBranding($simID)**
   Global function from "db.class.php". This function, "getSimBranding", retrieves branding information associated with a specific scenario from the sim_branding_tbl database table.
3. **getScenarioCompetency($simID)**
   Global function from "db.class.php". This function, "getScenarioCompetency", retrieves competency information associated with a specific scenario from the competency_tbl database table.

4. **webBranding($client_id)**
   Global function from “db.class.php”. The function "webBranding" generates CSS styles and JavaScript for branding a web page based on a client's preferences stored in a database.
5. **UID()**
   Global function from “db.class.php”. The "UID" function generates a unique identifier (UUID) using MySQL's UUID() function. If the query does not return a result, it falls back to generating a unique identifier using md5(time()).
6. **getTotalAttempt($simID, $userid)**
   Global function from “db.class.php”. This function will return total attempts for the user and simulation passed as arguments.
7. **getAssignSimOtherData($simID, $userid)**
   Global function from “db.class.php”. This PHP function, "getAssignSimOtherData", is designed to retrieve assignment data related to a simulation ("$sim_id") for a particular user ("$user_id").
8. **Getting Questions & it’s Options for the Simulation**
   *$qSql = $db->prepare("SELECT * FROM question_tbl WHERE scenario_id = ?");*
   This retrieves all the retrieves all matching records from the "question_tbl" table into an array called "$questions" as per the Simulation ID passed in it.

   $dnd_choice_sql = "SELECT question_id, sub_option, image FROM sub_options_tbl WHERE question_id = ?";
   The SQL query is designed to fetch specific data (question_id, sub_option, image) from the sub_options_tbl table, filtered by a question_id.

   $dnd_match_sql = "SELECT answer_id, choice_option, image FROM answer_tbl WHERE question_id = ? ORDER BY RAND()";
   The SQL query is designed to fetch specific data (question_id, sub_option, image) from the answer_tbl table, filtered by a question_id.

Third Party JS Used

1. Articulate
2. OwlCarousel

JS Functions Used

1. **$('.dragdrop').draggable({})**
   The provided JavaScript/jQuery code snippet sets up draggable functionality for elements with the class dragdrop and defines their behavior when dropped onto elements with the class drop.
2. **$('body').on('click', '.submitbtn', function() {})**
   a. The script listens for clicks on .submitbtn elements and determines the type of question (MCQ, Swiping, Drag and Drop).

b. It validates if an option is selected for each type.
c. Shows a loading overlay during the process.
d. Loads the feedback modal with appropriate data or displays an error message if no option is selected.

JS Files Used

File Path: content/js/launch-sim.js

1. **$('body').on('click', '#view_sim', function () {})**
   This JavaScript code uses jQuery to handle a click event on an element with the ID view_sim. When this element is clicked, it performs an AJAX request to *'view-sim-modal.php'* to load content into a modal window.
2. **$('.close-banner').click(function(){})**
   This JavaScript function using "SweetAlert" and jQuery to confirm the user's intention to close a simulation. If the user confirms, a loading overlay is shown, and after a delay, the user is redirected to a URL specified in the "href" attribute of an element with the ID "close-ban".
3. **$('body').on('click', 'a.qassets_icon', function(e){})**
   This JavaScript code is a jQuery event handler that uses Fancybox to display content when an anchor element with the class "qassets_icon" is clicked. It customizes Fancybox to prevent closing when clicking outside the modal or pressing the ESC key and applies specific transition effects.
4. **$('body').on('click', '.document', function(e){})**
   This JavaScript code is a jQuery event handler that uses Fancybox to display content when an element with the class document is clicked. It customizes Fancybox to display the content in an iframe with specific open and close effects, and ensures the content auto-sizes within the modal.
5. **speak(obj) {}**
   This JavaScript function "speak", controls text-to-speech functionality using the Articulate plugin. It toggles between speaking, pausing, and resuming speech, and updates the display of play and pause icons accordingly.
6. **speak_feedback(obj) {}**
   The "speak_feedback" function manages text-to-speech functionality specifically for feedback, using the Articulate plugin. It toggles the speaking state (play/pause/resume) and updates the visibility of feedback-specific play and pause icons.

File Path: content/js/sim-branding.js

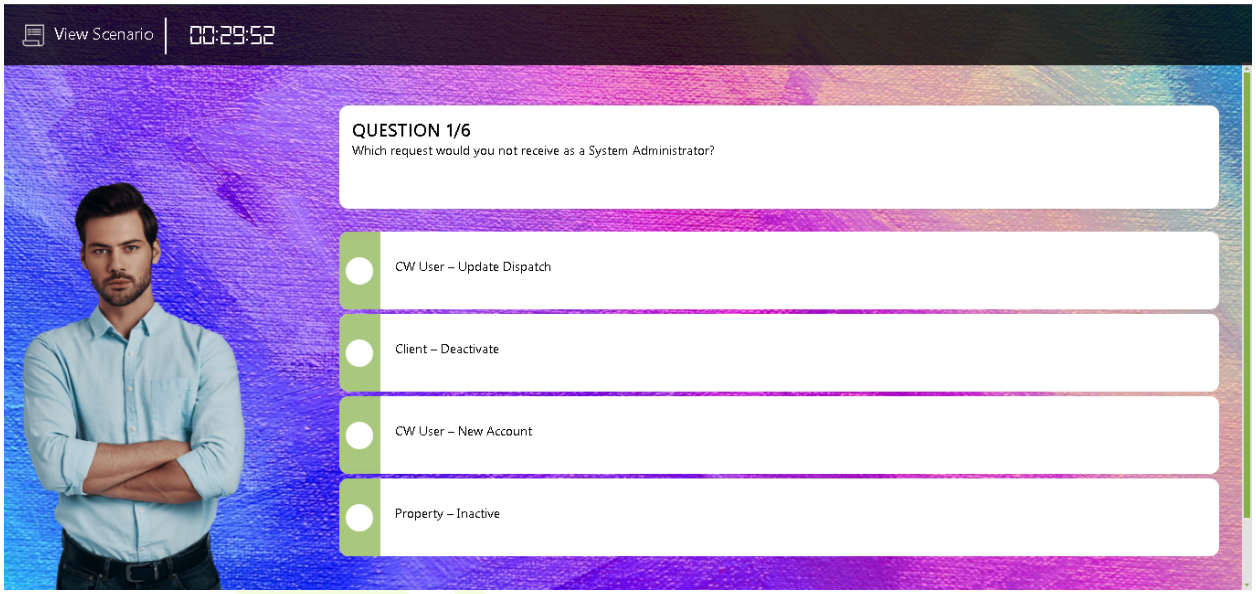
This JS file simply updates Background color on various elements on the page as per the JSON data fetched from the Server.

File Path: content/js/sim-temp.js

1. **const QuesTemp = (type, qid, aid = null, next = null, tabindex = null) => {}**
   The "QuesTemp" function is an asynchronous JavaScript function that fetches question template data from a server by sending AJAX request to *'includes/process.php'* and displays it in a modal. It uses jQuery for making AJAX requests and manipulating the DOM.
2. **$('body').on('click', '.launchAddData .naxtPreNav .preBtn', function(e){})**
   This jQuery script listens for a click event on the .preBtn button inside the .naxtPreNav navigation container within the .launchAddData section. When the button is clicked, it decreases the actSlNo variable by 1 to navigate to the previous slide. It also ensures actSlNo does not become negative, and then it calls the MakeSlideActive function to activate the corresponding slide.
3. **$('body').on('click', '.launchAddData .naxtPreNav .nextBtn', function(e){})**
   This jQuery script listens for a click event on an element and handles the navigation to the next slide in a series. It increments the actSlNo variable to move to the next slide, ensures it does not exceed the maximum number of slides, and calls the MakeSlideActive function to activate the corresponding slide.
4. **function MakeSlideActive(index, next = null) {}**
   The "MakeSlideActive" function manages the state of the slides within a modal by adding and removing classes to highlight the active slide. It also enables or disables navigation buttons based on the current slide index and handles special behavior when the last slide is reached.
5. **$('body').on('click', '.feedclose', function(e){})**
   The code defines an event handler for the ".feedclose" element. When this element is clicked, the function performs several actions, including preventing the default action, retrieving data attributes, hiding a modal, resetting a slide counter, and calling a function "QuesTemp" to load new content.
6. **$('body').on('click', '.qftemp_close', function(e){})**
   The code defines an event handler for the ".qftemp_close" element. When this element is clicked, the function performs several actions, including preventing the default action, retrieving data attributes, hiding a modal, resetting a slide counter, and calling a function "QuesTemp" to load new content.

Simulation > Branching Multiple

Template Name: launch-sim-branching-multiple.php



PHP Functions Used

1. **getScenario($sim_id)**
   Global function from “db.class.php”. This PHP function, "getScenario", retrieves scenario data from the database based on the provided scenario ID ($sim_id).
2. **getSimBranding($simID)**
   Global function from “db.class.php”. This function, “getSimBranding”, retrieves branding information associated with a specific scenario from the sim_branding_tbl database table.
3. **getScenarioCompetency($simID)**
   Global function from “db.class.php”. This function, "getScenarioCompetency", retrieves competency information associated with a specific scenario from the competency_tbl database table.
4. **webBranding($client_id)**
   Global function from “db.class.php”. The function "webBranding" generates CSS styles and JavaScript for branding a web page based on a client's preferences stored in a database.
5. **UID()**
   Global function from “db.class.php”. The "UID" function generates a unique identifier (UUID) using MySQL's UUID() function. If the query does not return a result, it falls back to generating a unique identifier using md5(time()).
6. **getTotalAttempt($simID, $userid)**
   Global function from “db.class.php”. This function will return total attempts for the user and simulation passed as arguments.
7. **getAssignSimOtherData($simID, $userid)**
   Global function from “db.class.php”. This PHP function, "getAssignSimOtherData", is designed to retrieve assignment data related to a simulation ("$sim_id") for a particular user ("$user_id").

8. **Getting Questions & it's Options for the Simulation**

*$qSql = $db->prepare("SELECT * FROM question_tbl WHERE scenario_id = ?");*
This retrieves all the retrieves all matching records from the "question_tbl" table into an array called "$questions" as per the Simulation ID passed in it.

$mtf_choice_sql = "SELECT answer_id, question_id, choice_option, match_option FROM answer_tbl WHERE question_id = ? ORDER BY (CASE WHEN choice_order_no != 0 THEN choice_order_no WHEN choice_order_no = 0 THEN RAND() END)";
This query is designed to fetch answer options for a specific question ID (question_id). The order of the options is determined by the choice_order_no column, with non-zero values indicating a specific order, and zero values resulting in random ordering.

$mtf_match_sql = "SELECT answer_id, match_option FROM answer_tbl WHERE question_id = ? ORDER BY (CASE WHEN match_order_no != 0 THEN match_order_no WHEN match_order_no = 0 THEN RAND() END)";
This query is designed to fetch match options for a specific question ID (question_id). The order of the match options is determined by the match_order_no column, with non-zero values indicating a specific order, and zero values resulting in random ordering.

$sq_sql = "SELECT answer_id, question_id, choice_option FROM answer_tbl WHERE question_id = ? ORDER BY RAND()";
The query retrieves the answer_id, question_id, and choice_option columns from the answer_tbl table for rows where the question_id matches a provided value. The results are then randomly ordered each time the query is executed. This approach is useful when you want to retrieve random rows from a table, such as in situations where you want to display random quiz questions or random choices to users.

$sort_choice_sql = "SELECT answer_id, question_id, choice_option, image FROM answer_tbl WHERE question_id = ?";
The query selects the answer_id, question_id, choice_option, and image columns from the answer_tbl table for rows where the question_id matches a provided value. This allows you to retrieve information about the choices for a specific question.

$sort_option_sql = "SELECT sub.sub_options_id, sub.sub_option FROM sub_options_tbl AS sub LEFT JOIN answer_tbl AS ans ON sub.answer_id = ans.answer_id WHERE ans.question_id = ? ".((! empty($shuffle)) ? "ORDER BY RAND() ": "");
The query retrieves the sub_options_id and sub_option columns from the sub_options_tbl table, joined with the answer_tbl table to retrieve options related to a specific question. If the $shuffle variable is not empty, the results are randomly ordered; otherwise, they are returned without any specific order.

$dnd_choice_sql = "SELECT question_id, sub_option, image FROM sub_options_tbl WHERE question_id = ?";

The SQL query is designed to fetch specific data (question_id, sub_option, image) from the sub_options_tbl table, filtered by a question_id.

$dnd2_choice_sql = "SELECT answer_id, question_id, image, choice_option FROM answer_tbl WHERE question_id = ? ".(($shuffle == TRUE) ? "ORDER BY RAND()" : "");
The query selects the answer_id, question_id, image, and choice_option columns from the answer_tbl table for rows where the question_id matches a provided value. If the $shuffle variable is TRUE, the results are randomly ordered; otherwise, they are returned without any specific order. This approach allows for flexibility in sorting the results based on a condition.

$sql2dnd = "SELECT answer_id, image2, match_option FROM answer_tbl WHERE question_id = ? ".(($shuffle == TRUE) ? "ORDER BY RAND()" : "");
The query selects the answer_id, image2, and match_option columns from the answer_tbl table for rows where the question_id matches a provided value. If the $shuffle variable is TRUE, the results are randomly ordered; otherwise, they are returned without any specific order. This approach provides flexibility in sorting the results based on a condition.

Third Party JS Used

1. Articulate
2. OwlCarousel

JS Functions Used

1. **$('.dragdrop').draggable({})**
   The provided JavaScript/jQuery code snippet sets up draggable functionality for elements with the class dragdrop and defines their behavior when dropped onto elements with the class drop.
2. **$('body').on('click', '.submitbtn', function() {})**
   a. The script listens for clicks on .submitbtn elements and determines the type of question (MCQ, Swiping, Drag and Drop).
   b. It validates if an option is selected for each type.
   c. Shows a loading overlay during the process.
   d. Loads the feedback modal with appropriate data or displays an error message if no option is selected.

JS Files Used

File Path: content/js/launch-sim.js

1. **$('body').on('click', '#view_sim', function () {})**
   This JavaScript code uses jQuery to handle a click event on an element with the ID

view_sim. When this element is clicked, it performs an AJAX request to *'view-sim-modal.php'* to load content into a modal window.

2. **$('.close-banner').click(function(){})**
   This JavaScript function using "SweetAlert" and jQuery to confirm the user's intention to close a simulation. If the user confirms, a loading overlay is shown, and after a delay, the user is redirected to a URL specified in the "href" attribute of an element with the ID "close-ban".
3. **$('body').on('click', 'a.qassets_icon', function(e){})**
   This JavaScript code is a jQuery event handler that uses Fancybox to display content when an anchor element with the class "qassets_icon" is clicked. It customizes Fancybox to prevent closing when clicking outside the modal or pressing the ESC key and applies specific transition effects.
4. **$('body').on('click', '.document', function(e){})**
   This JavaScript code is a jQuery event handler that uses Fancybox to display content when an element with the class document is clicked. It customizes Fancybox to display the content in an iframe with specific open and close effects, and ensures the content auto-sizes within the modal.
5. **speak(obj) {}**
   This JavaScript function "speak", controls text-to-speech functionality using the Articulate plugin. It toggles between speaking, pausing, and resuming speech, and updates the display of play and pause icons accordingly.
6. **speak_feedback(obj) {}**
   The "speak_feedback" function manages text-to-speech functionality specifically for feedback, using the Articulate plugin. It toggles the speaking state (play/pause/resume) and updates the visibility of feedback-specific play and pause icons.

File Path: content/js/sim-branding.js

This JS file simply updates Background color on various elements on the page as per the JSON data fetched from the Server.

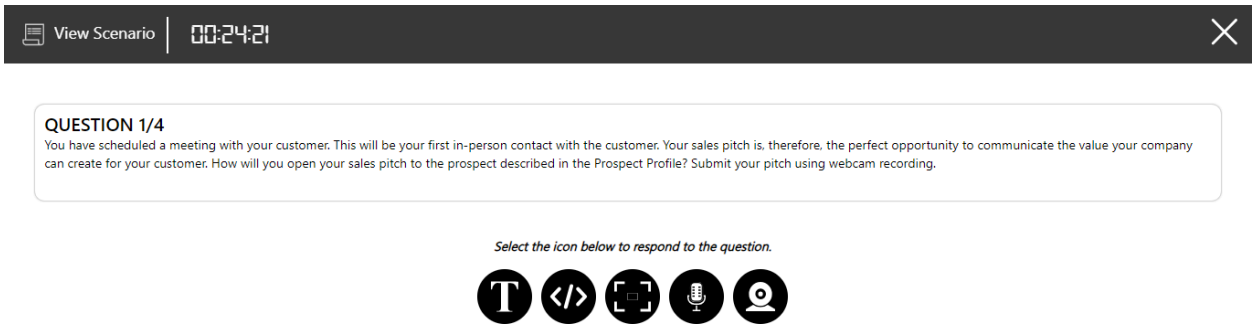
File Path: content/js/sim-temp.js

1. **const QuesTemp = (type, qid, aid = null, next = null, tabindex = null) => {}**
   The "QuesTemp" function is an asynchronous JavaScript function that fetches question template data from a server by sending AJAX request to *'includes/process.php'* and displays it in a modal. It uses jQuery for making AJAX requests and manipulating the DOM.
2. **$('body').on('click', '.launchAddData .naxtPreNav .preBtn', function(e){})**
   This jQuery script listens for a click event on the .preBtn button inside the .naxtPreNav navigation container within the .launchAddData section. When the button is clicked, it decreases the actSlNo variable by 1 to navigate to the previous slide. It also ensures actSlNo does not become negative, and then it calls the MakeSlideActive function to activate the corresponding slide.

3. **$('body').on('click', '.launchAddData .naxtPreNav .nextBtn', function(e){})**
   This jQuery script listens for a click event on an element and handles the navigation to the next slide in a series. It increments the actSlNo variable to move to the next slide, ensures it does not exceed the maximum number of slides, and calls the MakeSlideActive function to activate the corresponding slide.
4. **function MakeSlideActive(index, next = null) {}**
   The "MakeSlideActive" function manages the state of the slides within a modal by adding and removing classes to highlight the active slide. It also enables or disables navigation buttons based on the current slide index and handles special behavior when the last slide is reached.
5. **$('body').on('click', '.feedclose', function(e){})**
   The code defines an event handler for the ".feedclose" element. When this element is clicked, the function performs several actions, including preventing the default action, retrieving data attributes, hiding a modal, resetting a slide counter, and calling a function "QuesTemp" to load new content.
6. **$('body').on('click', '.qftemp_close', function(e){})**
   The code defines an event handler for the ".qftemp_close" element. When this element is clicked, the function performs several actions, including preventing the default action, retrieving data attributes, hiding a modal, resetting a slide counter, and calling a function "QuesTemp" to load new content.

Simulation > Open Responses

Template Name: launch-sim-open-response.php



PHP Functions Used

1. **getScenario($sim_id)**
   Global function from “db.class.php”. This PHP function, "getScenario", retrieves scenario data from the database based on the provided scenario ID ($sim_id).
2. **getSimBranding($simID)**
   Global function from “db.class.php”. This function, “getSimBranding”, retrieves branding

information associated with a specific scenario from the sim_branding_tbl database table.

3. **getScenarioCompetency($simID)**
   Global function from “db.class.php”. This function, "getScenarioCompetency", retrieves competency information associated with a specific scenario from the competency_tbl database table.
4. **webBranding($client_id)**
   Global function from “db.class.php”. The function "webBranding" generates CSS styles and JavaScript for branding a web page based on a client's preferences stored in a database.
5. **UID()**
   Global function from “db.class.php”. The "UID" function generates a unique identifier (UUID) using MySQL's UUID() function. If the query does not return a result, it falls back to generating a unique identifier using md5(time()).
6. **getTotalAttempt($simID, $userid)**
   Global function from “db.class.php”. This function will return total attempts for the user and simulation passed as arguments.
7. **getAssignSimOtherData($simID, $userid)**
   Global function from “db.class.php”. This PHP function, "getAssignSimOtherData", is designed to retrieve assignment data related to a simulation ("$sim_id") for a particular user ("$user_id").
8. **Getting Questions and Options**
   *$qSql = $db->prepare("SELECT * FROM question_tbl WHERE scenario_id = ? ORDER BY qorder = 0, qorder");*
   The line of code prepares an SQL statement to select all records from the "question_tbl" table where the "scenario_id" matches a specified value. The results are ordered such that rows with qorder equal to 0 appear first, followed by sorting based on the "qorder" column. This prepared statement can be executed with a specific "scenario_id" value later in the code.

Third Party JS Library

1. ArticulateJS
2. OwlCarousel

JS File Used

File Path: content/js/launch-sim.js

1. **$('body').on('click', '#view_sim', function () {})**
   This JavaScript code uses jQuery to handle a click event on an element with the ID view_sim. When this element is clicked, it performs an AJAX request to *'view-sim-modal.php'* to load content into a modal window.

2. **$('.close-banner').click(function(){})**
   This JavaScript function using "SweetAlert" and jQuery to confirm the user's intention to close a simulation. If the user confirms, a loading overlay is shown, and after a delay, the user is redirected to a URL specified in the "href" attribute of an element with the ID "close-ban".
3. **$('body').on('click', 'a.qassets_icon', function(e){})**
   This JavaScript code is a jQuery event handler that uses Fancybox to display content when an anchor element with the class "qassets_icon" is clicked. It customizes Fancybox to prevent closing when clicking outside the modal or pressing the ESC key and applies specific transition effects.
4. **$('body').on('click', '.document', function(e){})**
   This JavaScript code is a jQuery event handler that uses Fancybox to display content when an element with the class document is clicked. It customizes Fancybox to display the content in an iframe with specific open and close effects, and ensures the content auto-sizes within the modal.
5. **speak(obj) {}**
   This JavaScript function "speak", controls text-to-speech functionality using the Articulate plugin. It toggles between speaking, pausing, and resuming speech, and updates the display of play and pause icons accordingly.
6. **speak_feedback(obj) {}**
   The "speak_feedback" function manages text-to-speech functionality specifically for feedback, using the Articulate plugin. It toggles the speaking state (play/pause/resume) and updates the visibility of feedback-specific play and pause icons.

File Path: content/js/sim-temp.js

1. **const QuesTemp = (type, qid, aid = null, next = null, tabindex = null) => {}**
   The "QuesTemp" function is an asynchronous JavaScript function that fetches question template data from a server by sending AJAX request to *'includes/process.php'* and displays it in a modal. It uses jQuery for making AJAX requests and manipulating the DOM.
2. **$('body').on('click', '.launchAddData .naxtPreNav .preBtn', function(e){})**
   This jQuery script listens for a click event on the .preBtn button inside the .naxtPreNav navigation container within the .launchAddData section. When the button is clicked, it decreases the actSlNo variable by 1 to navigate to the previous slide. It also ensures actSlNo does not become negative, and then it calls the MakeSlideActive function to activate the corresponding slide.
3. **$('body').on('click', '.launchAddData .naxtPreNav .nextBtn', function(e){})**
   This jQuery script listens for a click event on an element and handles the navigation to the next slide in a series. It increments the actSlNo variable to move to the next slide, ensures it does not exceed the maximum number of slides, and calls the MakeSlideActive function to activate the corresponding slide.
4. **function MakeSlideActive(index, next = null) {}**
   The "MakeSlideActive" function manages the state of the slides within a modal by adding

and removing classes to highlight the active slide. It also enables or disables navigation buttons based on the current slide index and handles special behavior when the last slide is reached.

5. **$('body').on('click', '.feedclose', function(e){})**
   The code defines an event handler for the ".feedclose" element. When this element is clicked, the function performs several actions, including preventing the default action, retrieving data attributes, hiding a modal, resetting a slide counter, and calling a function "QuesTemp" to load new content.
6. **$('body').on('click', '.qftemp_close', function(e){})**
   The code defines an event handler for the ".qftemp_close" element. When this element is clicked, the function performs several actions, including preventing the default action, retrieving data attributes, hiding a modal, resetting a slide counter, and calling a function "QuesTemp" to load new content.

File Path: content/js/launch-sim-open-response.js

1. **$('body').on('click', '.add-tts', function() {})**
   This JavaScript/jQuery code handles an event triggered by an element with certain data-* attributes (data-box-id, data-qid, and data-qtype). It performs the following steps:
   a. Retrieves the values of these attributes.
   b. Checks if box_id is not empty.
   c. Shows a loading overlay.
   d. Makes an AJAX request to *tts-open-response-modal.php*, passing the retrieved data.
   e. Loads the response into a modal.
   f. Hides the loading overlay.
   g. If the response is not empty, it displays the modal with specific settings.
   h. If the response is empty, it shows an error alert using SweetAlert.
2. **$('body').on('click', '.add-code', function() {})**
   This JavaScript/jQuery code handles an event triggered by an element with certain data-* attributes (data-box-id, data-qid, and data-qtype). It performs the following steps:
   a. Retrieves the values of these attributes.
   b. Checks if box_id is not empty.
   c. Shows a loading overlay to indicate a process is ongoing.
   d. Makes an AJAX request to *code-open-response-modal.php*, passing the retrieved data.
   e. Loads the response into a modal element.
   f. Hides the loading overlay after the content has been loaded.
   g. If the response is not empty, it displays the modal with the loaded content, preventing it from being closed by clicking outside or pressing the ESC key.
   h. If the response is empty, it shows an error alert using SweetAlert.
3. **$('body').on('click', '.rec-audio', function() {})**
   This JavaScript/jQuery code handles a click event on elements with the class .rec-audio. It performs the following steps:

a. Sets up an event listener for click events on .rec-audio elements, using event delegation from the body element.
b. Retrieves the data-box-id, data-qid, and data-qtype attributes from the clicked .rec-audio element.
c. Checks if box_id is not empty.
d. Shows a loading overlay to indicate an ongoing process.
e. Makes an AJAX request to *"rec-audio-open-response-modal.php"*, passing the retrieved data.
f. Loads the response into a modal element.
g. Hides the loading overlay after the content has been loaded.
h. If the response is not empty, it displays the modal with the loaded content, preventing it from being closed by clicking outside or pressing the ESC key.
i. If the response is empty, it shows an error alert using SweetAlert.

4. **$('body').on('click', '.rec-video', function() {})**
   This JavaScript/jQuery code sets up an event handler for clicks on elements with the class .rec-video within the body element.
5. **$('body').on('click', '.rec-screen', function() {})**
   This JavaScript/jQuery code sets up an event handler for clicks on elements with the class .rec-screen within the body element. It loads content into a modal dialog, and providing error handling if the content cannot be loaded.
6. **$('body').on('click', '.delete_assets', function() {})**
   This JavaScript/jQuery code sets up an event handler for clicks on elements with the class .delete_assets within the body element.
   a. It checks if the file_name variable is not empty.
   b. If file_name is not empty, it displays a confirmation dialog using the SweetAlert library.
   c. If the user confirms deletion, it sends an AJAX request to delete the file using the URL *"includes/delete-file.php"* with the constructed data string.
   d. Upon successful deletion, it updates the UI accordingly, clears the associated input field, hides the answer, shows action buttons, and displays a success message.
   e. If deletion fails, it shows an error message.
7. **$('body').on('click', '.delete_tts', function() {})**
   This JavaScript/jQuery code sets up an event handler for clicks on elements with the class .delete_tts within the body element. It displays a confirmation dialog using the SweetAlert library, asking the user if they are sure they want to delete the text.

   If the user confirms deletion, it performs the following actions:
   a. Shows a loading overlay.
   b. Clears the associated input field (#tts_data_) using the val('') method.
   c. Hides related elements (#open_res_answer_ and #open_answer_btn_).
   d. Clears and hides any text boxes associated with the response answer.
   e. Shows action buttons associated with the text data.
   f. Hides the loading overlay.

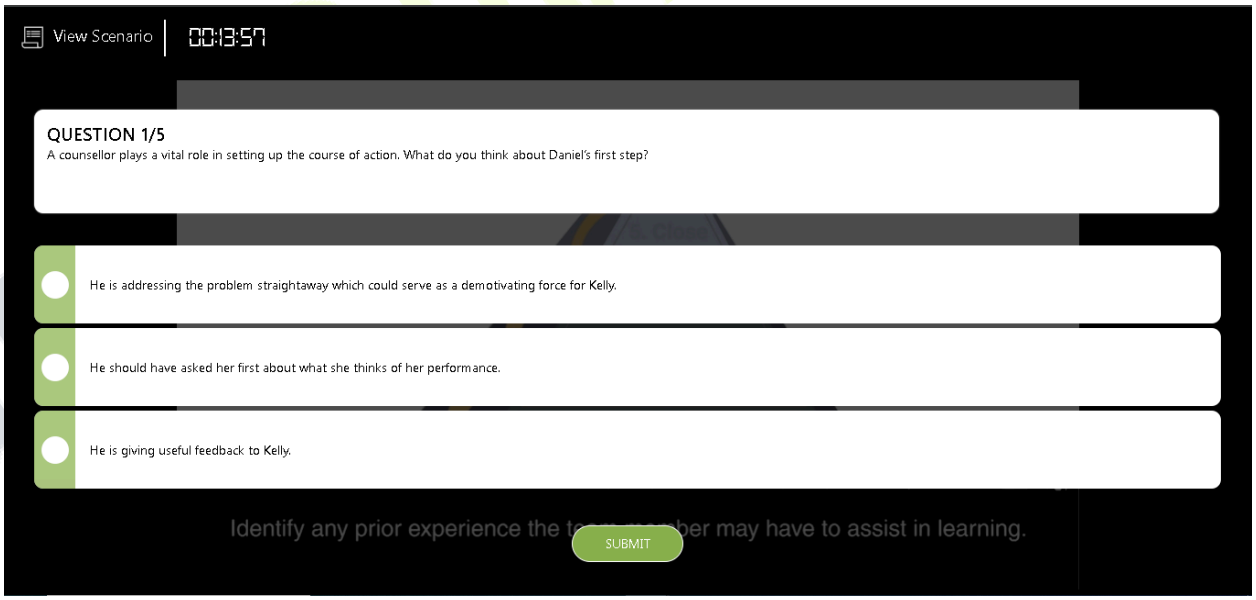
8. **$('body').on('click', '.submitbtn', function() {})**
   This JavaScript/jQuery code handles various scenarios related to submitting answers to questions.

File Path: content/js/sim-branding.js

This JS file simply updates Background color on various elements on the page as per the JSON data fetched from the Server.

Simulation > Linear Multiple Video

Template Name: launch-sim-linear-multiple-video.php



PHP Functions Used

1. **getScenario($sim_id)**
   Global function from "db.class.php". This PHP function, "getScenario", retrieves scenario data from the database based on the provided scenario ID ($sim_id).
2. **getSimBranding($simID)**
   Global function from "db.class.php". This function, "getSimBranding", retrieves branding information associated with a specific scenario from the sim_branding_tbl database table.
3. **getScenarioCompetency($simID)**
   Global function from "db.class.php". This function, "getScenarioCompetency", retrieves competency information associated with a specific scenario from the competency_tbl database table.
4. **webBranding($client_id)**
   Global function from "db.class.php". The function "webBranding" generates CSS styles

and JavaScript for branding a web page based on a client's preferences stored in a database.

5. **UID()**
   Global function from “db.class.php”. The "UID" function generates a unique identifier (UUID) using MySQL's UUID() function. If the query does not return a result, it falls back to generating a unique identifier using md5(time()).
6. **getTotalAttempt($simID, $userid)**
   Global function from “db.class.php”. This function will return total attempts for the user and simulation passed as arguments.
7. **getAssignSimOtherData($simID, $userid)**
   Global function from “db.class.php”. This PHP function, "getAssignSimOtherData", is designed to retrieve assignment data related to a simulation ("$sim_id") for a particular user ("$user_id").
8. **Getting Questions and Options**
   *$qSql = $db->prepare("SELECT * FROM question_tbl WHERE scenario_id = ? ORDER BY qorder = 0, qorder");*
   The line of code prepares an SQL statement to select all records from the "question_tbl" table where the "scenario_id" matches a specified value. The results are ordered such that rows with qorder equal to 0 appear first, followed by sorting based on the "qorder" column. This prepared statement can be executed with a specific "scenario_id" value later in the code.

   $mtf_choice_sql = "SELECT answer_id, question_id, choice_option, match_option FROM answer_tbl WHERE question_id = ? ORDER BY (CASE WHEN choice_order_no != 0 THEN choice_order_no WHEN choice_order_no = 0 THEN RAND() END)";
   This SQL query retrieves information from the answer_tbl table related to multiple-choice matching type questions (often abbreviated as MTF or Matching Type Format).

   $mtf_match_sql = "SELECT answer_id, match_option FROM answer_tbl WHERE question_id = ? ORDER BY (CASE WHEN match_order_no != 0 THEN match_order_no WHEN match_order_no = 0 THEN RAND() END)";
   The query retrieves the answer_id and match_option columns from the answer_tbl table for rows where the question_id matches a provided value. The results are ordered such that rows with a non-zero match_order_no are sorted by that value, while rows with a match_order_no of 0 are placed in a random order. This approach ensures that certain matches appear in a specific order, while others appear randomly, which can be useful in scenarios where some items need a fixed order while others do not.

   $sq_sql = "SELECT answer_id, question_id, choice_option FROM answer_tbl WHERE question_id = ? ORDER BY RAND()";
   The query retrieves the answer_id, question_id, and choice_option columns from the answer_tbl table for rows where the question_id matches a provided value. The results are then randomly ordered each time the query is executed. This approach is useful when you want to retrieve random rows from a table, such as in situations where you

want to display random quiz questions or random choices to users.

$sort_choice_sql = "SELECT answer_id, question_id, choice_option, image FROM answer_tbl WHERE question_id = ?";
The query selects the answer_id, question_id, choice_option, and image columns from the answer_tbl table for rows where the question_id matches a provided value. This allows you to retrieve information about the choices for a specific question.

$sort_option_sql = "SELECT sub.sub_options_id, sub.sub_option FROM sub_options_tbl AS sub LEFT JOIN answer_tbl AS ans ON sub.answer_id = ans.answer_id WHERE ans.question_id = ? ".((! empty($shuffle)) ? "ORDER BY RAND() ": "");
The query retrieves the sub_options_id and sub_option columns from the sub_options_tbl table, joined with the answer_tbl table to retrieve options related to a specific question. If the $shuffle variable is not empty, the results are randomly ordered; otherwise, they are returned without any specific order.

$dnd_choice_sql = "SELECT question_id, sub_option, image FROM sub_options_tbl WHERE question_id = ?";
The SQL query is designed to fetch specific data (question_id, sub_option, image) from the sub_options_tbl table, filtered by a question_id.

$dnd2_choice_sql = "SELECT answer_id, question_id, image, choice_option FROM answer_tbl WHERE question_id = ? ".(($shuffle == TRUE) ? "ORDER BY RAND()" : "");
The query selects the answer_id, question_id, image, and choice_option columns from the answer_tbl table for rows where the question_id matches a provided value. If the $shuffle variable is TRUE, the results are randomly ordered; otherwise, they are returned without any specific order. This approach allows for flexibility in sorting the results based on a condition.

$sql2dnd = "SELECT answer_id, image2, match_option FROM answer_tbl WHERE question_id = ? ".(($shuffle == TRUE) ? "ORDER BY RAND()" : "");
The query selects the answer_id, image2, and match_option columns from the answer_tbl table for rows where the question_id matches a provided value. If the $shuffle variable is TRUE, the results are randomly ordered; otherwise, they are returned without any specific order. This approach provides flexibility in sorting the results based on a condition.

9. **getAnswersByQuestionId(md5($qid), $shuffle)**
   Global function from “db.class.php”. The "getAnswersByQuestionId" function retrieves answers from the answer_tbl table based on a given question ID. It uses the MD5 hash of the question ID to perform the lookup. If the $shuffle parameter is TRUE, the results are returned in a random order. If there are results, they are fetched and returned as an array; otherwise, an empty array is returned.

Third Party JS Library

1. ArticulateJS
2. OwlCarousel

JS Function Used

1. **$('.dragdrop').draggable({})**
   This code enables drag-and-drop functionality for .dragdrop elements, allowing them to be moved between drop targets with swapping functionality implemented when necessary. Additionally, it includes logic to refresh the Owl Carousel ($('.COURSALbtnSty.swipecoursal').owlCarousel('refresh')) after the drop operation, ensuring proper display if the drop operation affects the carousel's content.
2. **$('.dragdrop1').draggable({})**
   This code enables drag-and-drop functionality for .dragdrop1 elements, allowing them to be moved and dropped into .drop1 targets. Additionally, it includes logic to clean up the carousel's content and refresh the carousel after the drop operation.

JS File Used

File Path: content/js/launch-sim.js

1. **$('body').on('click', '#view_sim', function () {})**
   This JavaScript code uses jQuery to handle a click event on an element with the ID view_sim. When this element is clicked, it performs an AJAX request to *'view-sim-modal.php'* to load content into a modal window.
2. **$('.close-banner').click(function(){})**
   This JavaScript function using "SweetAlert" and jQuery to confirm the user's intention to close a simulation. If the user confirms, a loading overlay is shown, and after a delay, the user is redirected to a URL specified in the "href" attribute of an element with the ID "close-ban".
3. **$('body').on('click', 'a.qassets_icon', function(e){})**
   This JavaScript code is a jQuery event handler that uses Fancybox to display content when an anchor element with the class "qassets_icon" is clicked. It customizes Fancybox to prevent closing when clicking outside the modal or pressing the ESC key and applies specific transition effects.
4. **$('body').on('click', '.document', function(e){})**
   This JavaScript code is a jQuery event handler that uses Fancybox to display content when an element with the class document is clicked. It customizes Fancybox to display the content in an iframe with specific open and close effects, and ensures the content auto-sizes within the modal.
5. **speak(obj) {}**
   This JavaScript function "speak", controls text-to-speech functionality using the Articulate plugin. It toggles between speaking, pausing, and resuming speech, and updates the display of play and pause icons accordingly.

6. **speak_feedback(obj) {}**
   The "speak_feedback" function manages text-to-speech functionality specifically for feedback, using the Articulate plugin. It toggles the speaking state (play/pause/resume) and updates the visibility of feedback-specific play and pause icons.

File Path: content/js/sim-branding.js

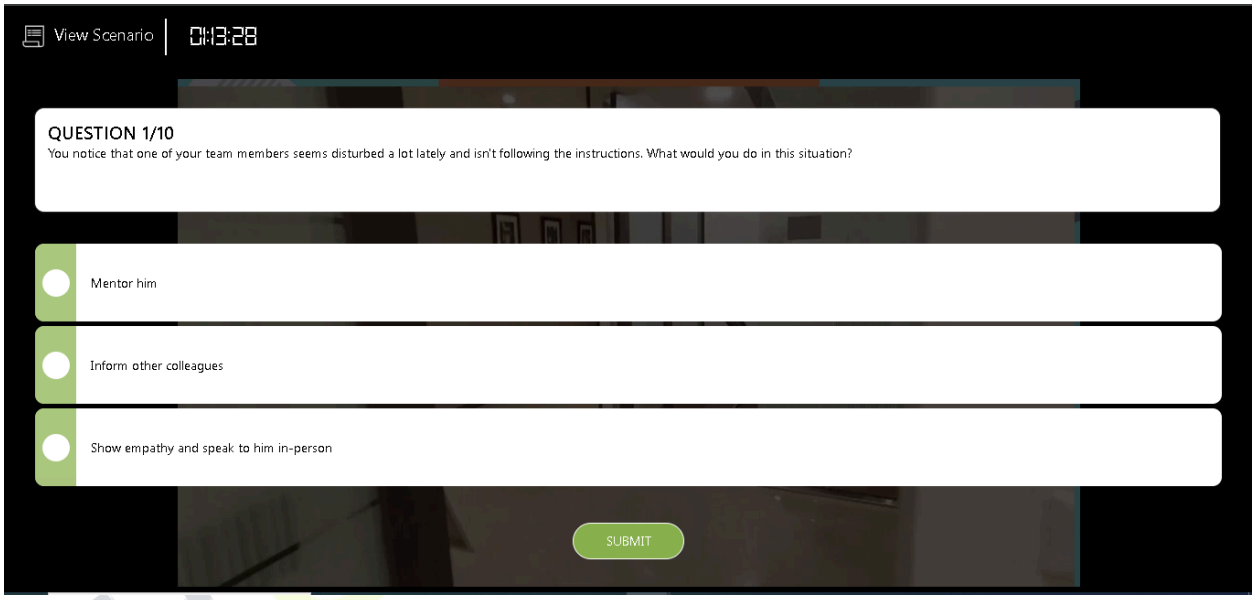
This JS file simply updates Background color on various elements on the page as per the JSON data fetched from the Server.

File Path: content/js/sim-temp.js

1. **const QuesTemp = (type, qid, aid = null, next = null, tabindex = null) => {}**
   The "QuesTemp" function is an asynchronous JavaScript function that fetches question template data from a server by sending AJAX request to *'includes/process.php'* and displays it in a modal. It uses jQuery for making AJAX requests and manipulating the DOM.
2. **$('body').on('click', '.launchAddData .naxtPreNav .preBtn', function(e){})**
   This jQuery script listens for a click event on the .preBtn button inside the .naxtPreNav navigation container within the .launchAddData section. When the button is clicked, it decreases the actSlNo variable by 1 to navigate to the previous slide. It also ensures actSlNo does not become negative, and then it calls the MakeSlideActive function to activate the corresponding slide.
3. **$('body').on('click', '.launchAddData .naxtPreNav .nextBtn', function(e){})**
   This jQuery script listens for a click event on an element and handles the navigation to the next slide in a series. It increments the actSlNo variable to move to the next slide, ensures it does not exceed the maximum number of slides, and calls the MakeSlideActive function to activate the corresponding slide.
4. **function MakeSlideActive(index, next = null) {}**
   The "MakeSlideActive" function manages the state of the slides within a modal by adding and removing classes to highlight the active slide. It also enables or disables navigation buttons based on the current slide index and handles special behavior when the last slide is reached.
5. **$('body').on('click', '.feedclose', function(e){})**
   The code defines an event handler for the ".feedclose" element. When this element is clicked, the function performs several actions, including preventing the default action, retrieving data attributes, hiding a modal, resetting a slide counter, and calling a function "QuesTemp" to load new content.
6. **$('body').on('click', '.qftemp_close', function(e){})**
   The code defines an event handler for the ".qftemp_close" element. When this element is clicked, the function performs several actions, including preventing the default action, retrieving data attributes, hiding a modal, resetting a slide counter, and calling a function "QuesTemp" to load new content.

Simulation > Linear Classic Video

Template Name: launch-sim-linear-classic-video.php



PHP Functions Used

1. **getScenario($sim_id)**
   Global function from "db.class.php". This PHP function, "getScenario", retrieves scenario data from the database based on the provided scenario ID ($sim_id).
2. **getSimBranding($simID)**
   Global function from "db.class.php". This function, "getSimBranding", retrieves branding information associated with a specific scenario from the sim_branding_tbl database table.
3. **getScenarioCompetency($simID)**
   Global function from "db.class.php". This function, "getScenarioCompetency", retrieves competency information associated with a specific scenario from the competency_tbl database table.
4. **webBranding($client_id)**
   Global function from "db.class.php". The function "webBranding" generates CSS styles and JavaScript for branding a web page based on a client's preferences stored in a database.
5. **UID()**
   Global function from "db.class.php". The "UID" function generates a unique identifier (UUID) using MySQL's UUID() function. If the query does not return a result, it falls back to generating a unique identifier using md5(time()).
6. **getTotalAttempt($simID, $userid)**
   Global function from "db.class.php". This function will return total attempts for the user and simulation passed as arguments.
7. **getAssignSimOtherData($simID, $userid)**
   Global function from "db.class.php". This PHP function, "getAssignSimOtherData", is

designed to retrieve assignment data related to a simulation ("$sim_id") for a particular user ("$user_id").

8. **Getting Questions and Options**
   *$qSql = $db->prepare("SELECT * FROM question_tbl WHERE scenario_id = ? ORDER BY qorder = 0, qorder");*
   The line of code prepares an SQL statement to select all records from the "question_tbl" table where the "scenario_id" matches a specified value. The results are ordered such that rows with qorder equal to 0 appear first, followed by sorting based on the "qorder" column. This prepared statement can be executed with a specific "scenario_id" value later in the code.

   $dnd_choice_sql = "SELECT question_id, sub_option, image FROM sub_options_tbl WHERE question_id = ?";
   It retrieves data from a database table named sub_options_tbl for options for the question based on a specified question_id.
9. **getAnswersByQuestionId(md5($qid), $shuffle)**
   Global function from "db.class.php". The "getAnswersByQuestionId" function retrieves answers from the answer_tbl table based on a given question ID. It uses the MD5 hash of the question ID to perform the lookup. If the $shuffle parameter is TRUE, the results are returned in a random order. If there are results, they are fetched and returned as an array; otherwise, an empty array is returned.

Third Party JS Library

1. Articulate
2. OwlCarousel

JS Functions Used

1. **$('.dragdrop').draggable({})**
   This code enables drag-and-drop functionality for .dragdrop elements, allowing them to be moved and dropped into .drop targets. It includes logic to handle both dropping into empty targets and swapping drops when the target is already occupied. Additionally, it performs specific actions like class addition and carousel refresh based on the drop operation.
2. **$('body').on('click', '.submitbtn', function(){})**
   This code manages the click event on elements with the .submitbtn class, retrieves relevant data attributes, loads a modal dialog based on the type of question, and handles errors appropriately.

JS File Used

File Path: content/js/launch-sim.js

1. **$('body').on('click', '#view_sim', function () {})**
   This JavaScript code uses jQuery to handle a click event on an element with the ID view_sim. When this element is clicked, it performs an AJAX request to *'view-sim-modal.php'* to load content into a modal window.
2. **$('.close-banner').click(function(){})**
   This JavaScript function using "SweetAlert" and jQuery to confirm the user's intention to close a simulation. If the user confirms, a loading overlay is shown, and after a delay, the user is redirected to a URL specified in the "href" attribute of an element with the ID "close-ban".
3. **$('body').on('click', 'a.qassets_icon', function(e){})**
   This JavaScript code is a jQuery event handler that uses Fancybox to display content when an anchor element with the class "qassets_icon" is clicked. It customizes Fancybox to prevent closing when clicking outside the modal or pressing the ESC key and applies specific transition effects.
4. **$('body').on('click', '.document', function(e){})**
   This JavaScript code is a jQuery event handler that uses Fancybox to display content when an element with the class document is clicked. It customizes Fancybox to display the content in an iframe with specific open and close effects, and ensures the content auto-sizes within the modal.
5. **speak(obj) {}**
   This JavaScript function "speak", controls text-to-speech functionality using the Articulate plugin. It toggles between speaking, pausing, and resuming speech, and updates the display of play and pause icons accordingly.
6. **speak_feedback(obj) {}**
   The "speak_feedback" function manages text-to-speech functionality specifically for feedback, using the Articulate plugin. It toggles the speaking state (play/pause/resume) and updates the visibility of feedback-specific play and pause icons.

File Path: content/js/sim-branding.js

This JS file simply updates Background color on various elements on the page as per the JSON data fetched from the Server.

File Path: content/js/sim-temp.js

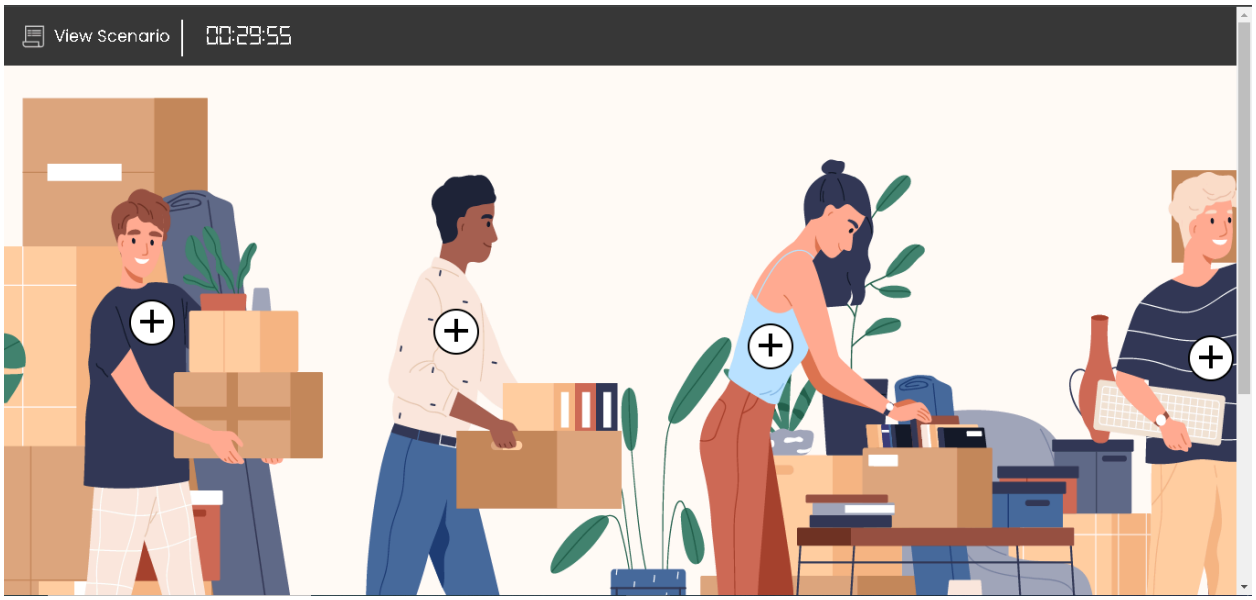
1. **const QuesTemp = (type, qid, aid = null, next = null, tabindex = null) => {}**
   The "QuesTemp" function is an asynchronous JavaScript function that fetches question template data from a server by sending AJAX request to *'includes/process.php'* and displays it in a modal. It uses jQuery for making AJAX requests and manipulating the DOM.
2. **$('body').on('click', '.launchAddData .naxtPreNav .preBtn', function(e){})**
   This jQuery script listens for a click event on the .preBtn button inside the .naxtPreNav navigation container within the .launchAddData section. When the button is clicked, it decreases the actSlNo variable by 1 to navigate to the previous slide. It also ensures

actSlNo does not become negative, and then it calls the MakeSlideActive function to activate the corresponding slide.

3. **$('body').on('click', '.launchAddData .naxtPreNav .nextBtn', function(e){})**
   This jQuery script listens for a click event on an element and handles the navigation to the next slide in a series. It increments the actSlNo variable to move to the next slide, ensures it does not exceed the maximum number of slides, and calls the MakeSlideActive function to activate the corresponding slide.
4. **function MakeSlideActive(index, next = null) {}**
   The "MakeSlideActive" function manages the state of the slides within a modal by adding and removing classes to highlight the active slide. It also enables or disables navigation buttons based on the current slide index and handles special behavior when the last slide is reached.
5. **$('body').on('click', '.feedclose', function(e){})**
   The code defines an event handler for the ".feedclose" element. When this element is clicked, the function performs several actions, including preventing the default action, retrieving data attributes, hiding a modal, resetting a slide counter, and calling a function "QuesTemp" to load new content.
6. **$('body').on('click', '.qftemp_close', function(e){})**
   The code defines an event handler for the ".qftemp_close" element. When this element is clicked, the function performs several actions, including preventing the default action, retrieving data attributes, hiding a modal, resetting a slide counter, and calling a function "QuesTemp" to load new content.

Simulation > 360 Simulation

Template Name: launch-360-sim.php



PHP Functions Used

1. **getScenario($sim_id)**
   Global function from “db.class.php”. This PHP function, "getScenario", retrieves scenario data from the database based on the provided scenario ID ($sim_id).
2. **getSimBranding($simID)**
   Global function from “db.class.php”. This function, “getSimBranding”, retrieves branding information associated with a specific scenario from the sim_branding_tbl database table.
3. **getScenarioCompetency($simID)**
   Global function from “db.class.php”. This function, "getScenarioCompetency", retrieves competency information associated with a specific scenario from the competency_tbl database table.
4. **webBranding($client_id)**
   Global function from “db.class.php”. The function "webBranding" generates CSS styles and JavaScript for branding a web page based on a client's preferences stored in a database.
5. **UID()**
   Global function from “db.class.php”. The "UID" function generates a unique identifier (UUID) using MySQL's UUID() function. If the query does not return a result, it falls back to generating a unique identifier using md5(time()).
6. **getTotalAttempt($simID, $userid)**
   Global function from “db.class.php”. This function will return total attempts for the user and simulation passed as arguments.
7. **getAssignSimOtherData($simID, $userid)**
   Global function from “db.class.php”. This PHP function, "getAssignSimOtherData", is designed to retrieve assignment data related to a simulation ("$sim_id") for a particular user ("$user_id").

JS Functions Used

1. **document.addEventListener("dragstart", function(event) {})**
   The purpose of this code is to hide the default drag image that appears when an element is dragged. It achieves this by first setting a drag image that is offset outside the window, then replacing it with a blank image positioned at (0, 0), making the drag image invisible.
2. **document.addEventListener("dragover", event => {})**
   The purpose of this code is to continuously call the SetBgPos function with the current horizontal mouse position (event.clientX) whenever an item is dragged over the document. This is used for dynamically updating the background position of an element in response to the drag position.
3. **document.getElementById('view360').addEventListener('touchstart', touchStart, false);**
   document.getElementById('Drag').addEventListener('touchmove', touchDragStart, false);
   document.getElementById('Drag').addEventListener('touchend', touchDragEnd,

false);
The code sets up touch event listeners on two elements identified by their IDs (view360 and Drag). The #view360 element listens for the touchstart event and calls the touchStart function when the event occurs. The #Drag element listens for the touchmove event and calls the touchDragStart function when the event occurs. The #Drag element also listens for the touchend event and calls the touchDragEnd function when the event occurs.

4. **$('.view360').mousemove(function(e2) {})**
   The purpose of this code is to track the mouse movement within elements that have the view360 class. As the mouse moves, the relative position of the mouse pointer (relative to the top-left corner of the view360 element) is calculated and the left attribute of the element with the ID Drag is updated accordingly.
5. **var setHSPos = setInterval(function() {})**
   The purpose of this code is to ensure that the width of the .mainPic element matches the width of its child image and to adjust the positions of any hotspot elements within .mainPic so they wrap around correctly if they are outside the visible bounds of the container. The interval repeatedly executes these adjustments until all hotspots are positioned correctly, after which the interval is cleared to stop further executions.

Third Party JS Library

1. ArticulateJS
2. OwlCarousel

JS File Used

File Path: content/js/launch-sim.js

1. **$('body').on('click', '#view_sim', function () {})**
   This JavaScript code uses jQuery to handle a click event on an element with the ID view_sim. When this element is clicked, it performs an AJAX request to *'view-sim-modal.php'* to load content into a modal window.
2. **$('.close-banner').click(function(){})**
   This JavaScript function using "SweetAlert" and jQuery to confirm the user's intention to close a simulation. If the user confirms, a loading overlay is shown, and after a delay, the user is redirected to a URL specified in the "href" attribute of an element with the ID "close-ban".
3. **$('body').on('click', 'a.qassets_icon', function(e){})**
   This JavaScript code is a jQuery event handler that uses Fancybox to display content when an anchor element with the class "qassets_icon" is clicked. It customizes Fancybox to prevent closing when clicking outside the modal or pressing the ESC key and applies specific transition effects.
4. **$('body').on('click', '.document', function(e){})**
   This JavaScript code is a jQuery event handler that uses Fancybox to display content

when an element with the class document is clicked. It customizes Fancybox to display the content in an iframe with specific open and close effects, and ensures the content auto-sizes within the modal.

5. **speak(obj) {}**
   This JavaScript function "speak", controls text-to-speech functionality using the Articulate plugin. It toggles between speaking, pausing, and resuming speech, and updates the display of play and pause icons accordingly.
6. **speak_feedback(obj) {}**
   The "speak_feedback" function manages text-to-speech functionality specifically for feedback, using the Articulate plugin. It toggles the speaking state (play/pause/resume) and updates the visibility of feedback-specific play and pause icons.

File Path: content/js/sim-branding.js
This JS file simply updates Background color on various elements on the page as per the JSON data fetched from the Server.

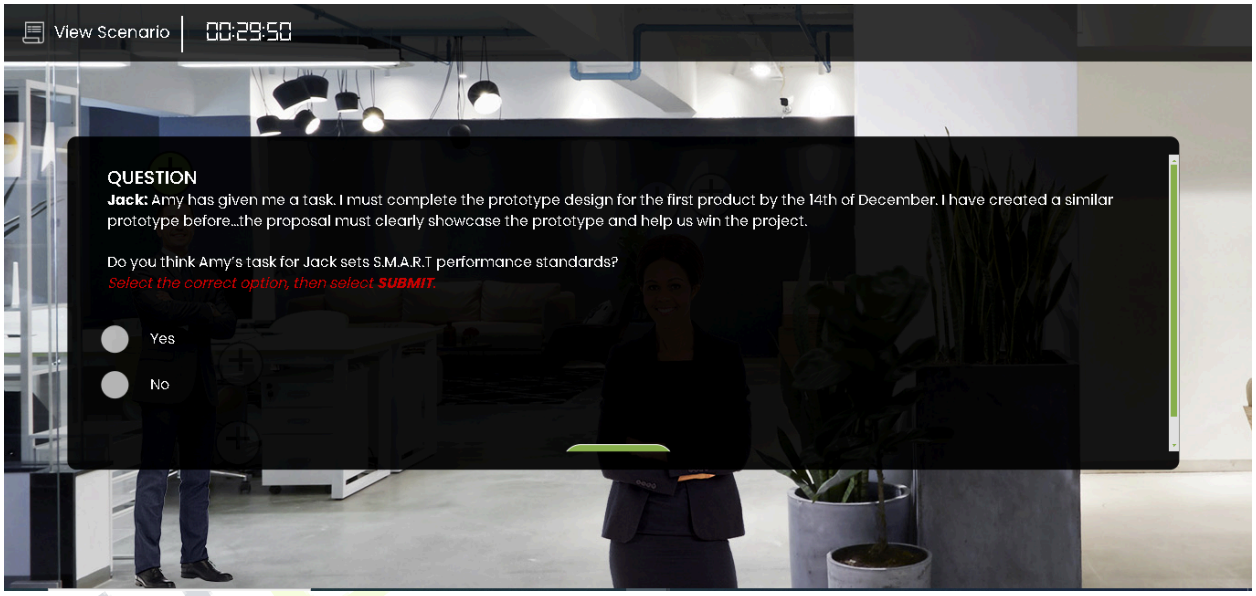
File Path: content/js/launch-360.js

1. **getExtension = (filename) => {}**
   The getExtension function safely extracts and returns the file extension from a given filename string.
2. **const shuffle = (array) => {}**
   The shuffle function takes an array and returns a new array with its elements shuffled in random order.
3. **const ddSubOptions = (qid) => {}**
   The provided code defines an asynchronous function named ddSubOptions using an arrow function syntax in JavaScript. This function sends an AJAX POST request to *'includes/process.php'* with parameters as *{'sim_id': sim_id, 'hid': hid, 'get_hotspot_question': true}* and fetch sub-options for a given question ID (qid) and builds an HTML string based on the response.
4. **const getHotspot = (sim_id) => {}**
   The provided code defines a function named getHotspot using an arrow function syntax in JavaScript. This function sends a POST request to fetch hotspot data based on a given simulation ID (sim_id) and dynamically appends the hotspot elements to the DOM.
5. **const syncHS = (ele, offset) => {}**
   The provided code defines a function named syncHS using an arrow function syntax in JavaScript. This function synchronizes the position of elements with the class hotspot based on a given offset and hides or shows these elements depending on their position relative to a specified element.
6. **$('body').on('click', '.option-hotspot', function(){})**
   This code handles a click event on elements. It retrieves certain data attributes from the clicked element, updates attributes of other elements based on this data, adds/removes classes to/from elements for visual effects (such as highlighting), and enables elements for interaction by removing the disabled attribute.

7. **$('body').on('click', '.preview_hybrid', function(){})**
   This code snippet defines an event handler for click events on elements with the class .preview_hybrid. Upon a click event, it retrieves a data attribute value, hides a modal dialog, updates the classes of certain elements, and displays a warning message using a sweet alert dialog.
8. **$('body').on('click', '.add-tts', function() {})**
   This code sets up a click event listener on elements with the class .add-tts. When clicked, it retrieves data attributes from the clicked element and uses them to load content via AJAX request to *'tts-hybrid-modal.php'*. After loading the content, it shows a modal dialog. If the content loading fails or the content is empty, it displays an error message.
9. **$('body').on('click', '.add-code', function() {})**
   This code sets up a click event listener on elements with the class .add-code. When clicked, it retrieves data attributes from the clicked element and uses them to load content via AJAX request to *'code-hybrid-modal.php'*. After loading the content, it shows a modal dialog. If the content loading fails or the content is empty, it displays an error message.
10. **$('body').on('click', '.rec-audio', function() {})**
    This code essentially handles a click event on elements with the class .rec-audio. When clicked, it loads content via AJAX to *'rec-audio-hybrid-modal.php'*, displays a modal dialog with the loaded content, and handles error conditions.
11. **$('body').on('click', '.rec-video', function() {})**
    This code essentially handles a click event on elements with the class .rec-video. When clicked, it loads content via AJAX request to *'rec-video-hybrid-modal.php'*, displays a modal dialog with the loaded content, and handles error conditions.
12. **$('body').on('click', '.rec-screen', function() {})**
    This code essentially handles a click event on elements with the class .rec-screen. When clicked, it loads content via AJAX request to *'rec-screen-hybrid-modal.php'*, displays a modal dialog with the loaded content, and handles error conditions.
13. **$('body').on('click', '.delete_assets', function() {})**
    This JavaScript code is an event handler attached to the click event of elements with the class "delete_assets" that are children of the "body" element. When such an element is clicked, it performs a series of actions to delete an asset.
14. **$('body').on('click', '.delete_tts', function() {})**
    This JavaScript code is another event handler attached to the click event of elements with the class "delete_tts" that are children of the "body" element. When such an element is clicked, it displays a confirmation dialog using the swal() function from the SweetAlert library.
15. **$('body').on('click', '.submitbtn_hybrid', function() {})**
    This code handles the submission of hybrid question types, interacts with an API to insert data into a database, performs additional actions based on the type of data submitted, and manages UI elements based on certain conditions.

Simulation > 360 Branching Simulation

Template Name: launch-360-branching-sim.php



PHP Functions Used

1. **getScenario($sim_id)**
   Global function from "db.class.php". This PHP function, "getScenario", retrieves scenario data from the database based on the provided scenario ID ($sim_id).
2. **getSimBranding($simID)**
   Global function from "db.class.php". This function, "getSimBranding", retrieves branding information associated with a specific scenario from the sim_branding_tbl database table.
3. **getScenarioCompetency($simID)**
   Global function from "db.class.php". This function, "getScenarioCompetency", retrieves competency information associated with a specific scenario from the competency_tbl database table.
4. **webBranding($client_id)**
   Global function from "db.class.php". The function "webBranding" generates CSS styles and JavaScript for branding a web page based on a client's preferences stored in a database.
5. **UID()**
   Global function from "db.class.php". The "UID" function generates a unique identifier (UUID) using MySQL's UUID() function. If the query does not return a result, it falls back to generating a unique identifier using md5(time()).
6. **getTotalAttempt($simID, $userid)**
   Global function from "db.class.php". This function will return total attempts for the user and simulation passed as arguments.
7. **getAssignSimOtherData($simID, $userid)**
   Global function from "db.class.php". This PHP function, "getAssignSimOtherData", is

designed to retrieve assignment data related to a simulation ("$sim_id") for a particular user ("$user_id").

JS Functions Used

1. **document.addEventListener("dragstart", function(event) {})**
   The purpose of this code is to hide the default drag image that appears when an element is dragged. It achieves this by first setting a drag image that is offset outside the window, then replacing it with a blank image positioned at (0, 0), making the drag image invisible.
2. **document.addEventListener("dragover", event => {})**
   The purpose of this code is to continuously call the SetBgPos function with the current horizontal mouse position (event.clientX) whenever an item is dragged over the document. This is used for dynamically updating the background position of an element in response to the drag position.
3. **document.getElementById('view360').addEventListener('touchstart', touchStart, false);**
   document.getElementById('Drag').addEventListener('touchmove', touchDragStart, false);
   document.getElementById('Drag').addEventListener('touchend', touchDragEnd, false);
   The code sets up touch event listeners on two elements identified by their IDs (view360 and Drag). The #view360 element listens for the touchstart event and calls the touchStart function when the event occurs. The #Drag element listens for the touchmove event and calls the touchDragStart function when the event occurs. The #Drag element also listens for the touchend event and calls the touchDragEnd function when the event occurs.
4. **$('.view360').mousemove(function(e2) {})**
   The purpose of this code is to track the mouse movement within elements that have the view360 class. As the mouse moves, the relative position of the mouse pointer (relative to the top-left corner of the view360 element) is calculated and the left attribute of the element with the ID Drag is updated accordingly.
5. **var setHSPos = setInterval(function() {})**
   The purpose of this code is to ensure that the width of the .mainPic element matches the width of its child image and to adjust the positions of any hotspot elements within .mainPic so they wrap around correctly if they are outside the visible bounds of the container. The interval repeatedly executes these adjustments until all hotspots are positioned correctly, after which the interval is cleared to stop further executions.
6. **$('body').on('click', '.submitbtn', function() {})**
   This code handles the submission of various types of questions, ensures that an option is selected before submission, and displays corresponding modal content for further actions or feedback by loading *'view-360-branching-feedback-modal.php'* through AJAX.

Third Party JS Library

1. ArticulateJS
2. OwlCarousel

JS File Used

File Path: content/js/launch-sim.js

1. **$('body').on('click', '#view_sim', function () {})**
   This JavaScript code uses jQuery to handle a click event on an element with the ID view_sim. When this element is clicked, it performs an AJAX request to *'view-sim-modal.php'* to load content into a modal window.
2. **$('.close-banner').click(function(){})**
   This JavaScript function using "SweetAlert" and jQuery to confirm the user's intention to close a simulation. If the user confirms, a loading overlay is shown, and after a delay, the user is redirected to a URL specified in the "href" attribute of an element with the ID "close-ban".
3. **$('body').on('click', 'a.qassets_icon', function(e){})**
   This JavaScript code is a jQuery event handler that uses Fancybox to display content when an anchor element with the class "qassets_icon" is clicked. It customizes Fancybox to prevent closing when clicking outside the modal or pressing the ESC key and applies specific transition effects.
4. **$('body').on('click', '.document', function(e){})**
   This JavaScript code is a jQuery event handler that uses Fancybox to display content when an element with the class document is clicked. It customizes Fancybox to display the content in an iframe with specific open and close effects, and ensures the content auto-sizes within the modal.
5. **speak(obj) {}**
   This JavaScript function "speak", controls text-to-speech functionality using the Articulate plugin. It toggles between speaking, pausing, and resuming speech, and updates the display of play and pause icons accordingly.
6. **speak_feedback(obj) {}**
   The "speak_feedback" function manages text-to-speech functionality specifically for feedback, using the Articulate plugin. It toggles the speaking state (play/pause/resume) and updates the visibility of feedback-specific play and pause icons.

File Path: content/js/sim-branding.js

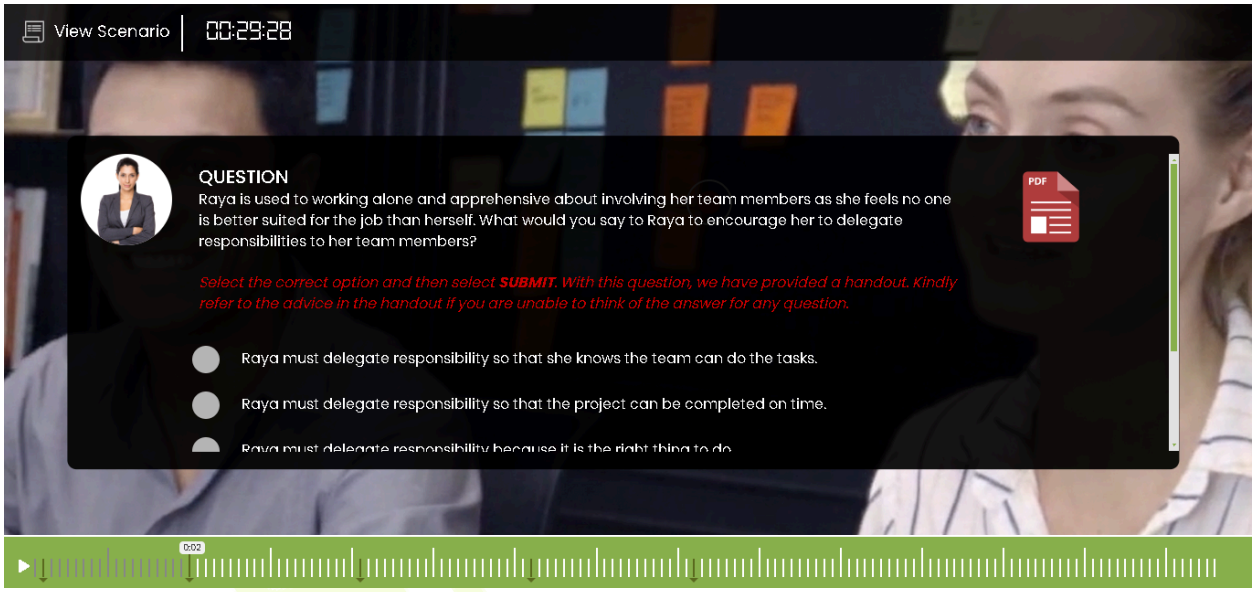
This JS file simply updates Background color on various elements on the page as per the JSON data fetched from the Server.

File Path: content/js/launch-360-branching.js

1. **getExtension = (filename) => {}**
   The getExtension function safely extracts and returns the file extension from a given filename string.
2. **const shuffle = (array) => {}**
   The shuffle function takes an array and returns a new array with its elements shuffled in random order.
3. **const ddSubOptions = (qid) => {}**
   The provided code defines an asynchronous function named ddSubOptions using an arrow function syntax in JavaScript. This function sends an AJAX POST request to *'includes/process.php'* with parameters as *{'sim_id': sim_id, 'hid': hid, 'get_hotspot_question': true}* and fetch sub-options for a given question ID (qid) and builds an HTML string based on the response.
4. **const hsOp = (hs) => {}**
   This function handles the retrieval and display of hotspot questions within a modal popup, allowing users to interact with and submit their responses. It dynamically generates the necessary HTML elements based on the question data retrieved from the server.
5. **const getHotspot = (sim_id) => {}**
   The provided code defines a function named getHotspot using an arrow function syntax in JavaScript. This function sends a POST request to fetch hotspot data based on a given simulation ID (sim_id) and dynamically appends the hotspot elements to the DOM.
6. **const syncHS = (ele, offset) => {}**
   The provided code defines a function named syncHS using an arrow function syntax in JavaScript. This function synchronizes the position of elements with the class hotspot based on a given offset and hides or shows these elements depending on their position relative to a specified element.
7. **$('body').on('click', '.option-hotspot', function(){})**
   This code handles a click event on elements. It retrieves certain data attributes from the clicked element, updates attributes of other elements based on this data, adds/removes classes to/from elements for visual effects (such as highlighting), and enables elements for interaction by removing the disabled attribute.

Simulation > 360 Video Branching Simulation

Template Name: launch-360v-branching-sim.php



PHP Functions Used

1. **getScenario($sim_id)**
   Global function from “db.class.php”. This PHP function, "getScenario", retrieves scenario data from the database based on the provided scenario ID ($sim_id).
2. **getSimBranding($simID)**
   Global function from “db.class.php”. This function, “getSimBranding”, retrieves branding information associated with a specific scenario from the sim_branding_tbl database table.
3. **getScenarioCompetency($simID)**
   Global function from “db.class.php”. This function, "getScenarioCompetency", retrieves competency information associated with a specific scenario from the competency_tbl database table.
4. **webBranding($client_id)**
   Global function from “db.class.php”. The function "webBranding" generates CSS styles and JavaScript for branding a web page based on a client's preferences stored in a database.
5. **UID()**
   Global function from “db.class.php”. The "UID" function generates a unique identifier (UUID) using MySQL's UUID() function. If the query does not return a result, it falls back to generating a unique identifier using md5(time()).
6. **getTotalAttempt($simID, $userid)**
   Global function from “db.class.php”. This function will return total attempts for the user and simulation passed as arguments.
7. **getAssignSimOtherData($simID, $userid)**
   Global function from “db.class.php”. This PHP function, "getAssignSimOtherData", is designed to retrieve assignment data related to a simulation ("$sim_id") for a particular user ("$user_id").

JS Functions Used

1. **document.getElementById('vidOuter').addEventListener('touchstart', touchStart, false);**
   document.getElementById('Vid3D').addEventListener('touchmove', touchDragStart, false);
   document.getElementById('Vid3D').addEventListener('touchend', touchDragEnd, false);
   These event listeners enable touch-based interactions on specific elements in the HTML document. They are commonly used in web applications to provide touch support for mobile devices, allowing users to perform actions such as dragging, scrolling, or tapping.
2. **function touchDragStart(event) {}**
   This function allows the user to drag an element (presumably a video or a 3D object) within a specified area by touching and moving their finger on a touch-enabled device. It ensures that the element stays within the defined boundaries and updates its position dynamically based on the user's touch input.
3. **function touchStart(event) {}**
   This function captures the initial touch position when a user starts interacting with a touch-enabled device. These coordinates are then used as reference points for calculating movement or other interactions during the touch event.
4. **function touchDragEnd() {}**
   This function restores the position of an element after a touch-dragging interaction has ended, effectively resetting it to its original location within the web page layout.

Third Party JS Library

1. ArticulateJS
2. OwlCarousel

JS File Used

File Path: content/js/launch-sim.js

1. **$('body').on('click', '#view_sim', function () {})**
   This JavaScript code uses jQuery to handle a click event on an element with the ID view_sim. When this element is clicked, it performs an AJAX request to *'view-sim-modal.php'* to load content into a modal window.
2. **$('.close-banner').click(function(){})**
   This JavaScript function using "SweetAlert" and jQuery to confirm the user's intention to close a simulation. If the user confirms, a loading overlay is shown, and after a delay, the user is redirected to a URL specified in the "href" attribute of an element with the ID "close-ban".
3. **$('body').on('click', 'a.qassets_icon', function(e){})**
   This JavaScript code is a jQuery event handler that uses Fancybox to display content when an anchor element with the class "qassets_icon" is clicked. It customizes

Fancybox to prevent closing when clicking outside the modal or pressing the ESC key and applies specific transition effects.

4. **$('body').on('click', '.document', function(e){})**
   This JavaScript code is a jQuery event handler that uses Fancybox to display content when an element with the class document is clicked. It customizes Fancybox to display the content in an iframe with specific open and close effects, and ensures the content auto-sizes within the modal.
5. **speak(obj) {}**
   This JavaScript function "speak", controls text-to-speech functionality using the Articulate plugin. It toggles between speaking, pausing, and resuming speech, and updates the display of play and pause icons accordingly.
6. **speak_feedback(obj) {}**
   The "speak_feedback" function manages text-to-speech functionality specifically for feedback, using the Articulate plugin. It toggles the speaking state (play/pause/resume) and updates the visibility of feedback-specific play and pause icons.

File Path: content/js/sim-branding.js

This JS file simply updates Background color on various elements on the page as per the JSON data fetched from the Server.

File Path: content/js/360v.js

1. **markerAndVideoFound = setInterval(function() {})**
   This code continuously synchronizes marker events with the video, updating the timeline display accordingly. Once synchronization is completed, the interval is cleared to prevent redundant executions.
2. **video3D.addEventListener('loadeddata', function() {})**
   This code ensures that some processing or action dependent on the event represented by eventLoaded being loaded is delayed until the event has indeed loaded. Once it's loaded, any necessary subsequent actions can be triggered.
3. **$('.vid3DWrap .playPause').click(function() {})**
   It sets up a click event listener on elements with the class ".vid3DWrap .playPause" and Toggle Play/Pause video element with the ID 'Vid3D'.
4. **$('.Vid3D #Vid3D').click(function(){})**
   This code snippet sets up a click event listener for a specific video element and toggles between playing and pausing the video when the element is clicked.
5. **$('body').on('click', '.timeFrame .framWrap .fv span', function(){})**
   This code allows users to click on specific time frames within the .timeFrame .framWrap .fv element, which in turn updates the playback time of the video3D element to that specific time and pauses the video playback.
6. **var vidPlayInterval = setInterval(function(){})**
   This code ensures that the time frame corresponding to the current playback time of the

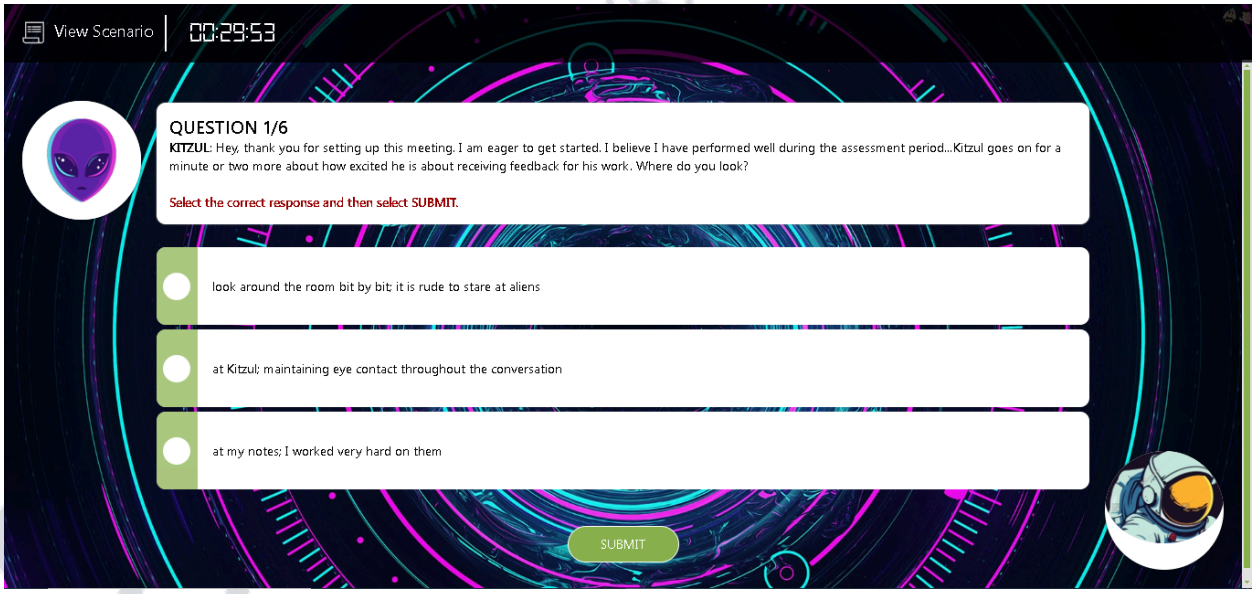
video3D element is visually highlighted by adding the active class, and it updates this highlight continuously as the video plays.

7. **video3D.addEventListener('play', (event) => {})**
   This event listener capture the current time of the video when it starts playing.
8. **video3D.addEventListener('pause', (event) => {})**
   This event listener capture the current time of the video when it starts playing.
9. **function updateTime() {}**
   This function is used to update the display of time frames based on the current playback time of the video3D element. It marks the frames as played or unplayed depending on whether their associated times have been reached during the video playback.
10. **const shuffle = (array) => {}**
    The shuffle function takes an array and returns a new array with its elements shuffled in random order.
11. **const ddSubOptions = (qid) => {}**
    The provided code defines an asynchronous function named ddSubOptions using an arrow function syntax in JavaScript. This function sends an AJAX POST request to *'includes/process.php'* with parameters as *{'sim_id': sim_id, 'hid': hid, 'get_hotspot_question': true}* and fetch sub-options for a given question ID (qid) and builds an HTML string based on the response.
12. **$('body').on('click', '.vid3DWrap .vidOuter .vm', function() {})**
    The code sets up a click event handler for certain elements, pauses the video when one of these elements is clicked, fetches data from the server *('*includes/process.php'*)*, dynamically generates HTML content based on the response, and displays this content in a modal. It ensures that the user interface updates seamlessly to reflect the fetched data, providing an interactive experience.
13. **function SetMarker(vTime, MarkerEvents) {}**
    The SetMarker function dynamically updates the markers on a video at a specified time (vTime) based on a set of marker events (MarkerEvents).
14. **function secToMinSec(sec) {}**
    The secToMinSec function converts a time value given in seconds into a string formatted as minutes and seconds.
15. **function resizeVid3DHeight() {}**
    The resizeVid3DHeight function adjusts the height and position of a video element within a wrapper to maintain a specific aspect ratio (16:9) and center the video content appropriately.
16. **function dragVideo(event) {}**
    The dragVideo function allows a user to drag a video element around within its container. This functionality typically enhances user interaction, enabling them to reposition the video within specified constraints.
17. **$('body').on('click', '.option-hotspot', function(){})**
    This jQuery code snippet handles the click event for elements with the class option-hotspot and performs several actions based on the clicked element.
18. **$('body').on('click', '.submitbtn', function() {})**
    This jQuery code snippet handles the click event for elements with the class submitbtn. It

processes the form submission based on the question type (MCQ, swiping, or drag-and-drop) and displays feedback in a modal.

Simulation > Hybrid Linear Classic

Template Name: launch-sim-hybrid-linear-classic.php



PHP Functions Used

1. **getScenario($sim_id)**
   Global function from “db.class.php”. This PHP function, "getScenario", retrieves scenario data from the database based on the provided scenario ID ($sim_id).
2. **getSimBranding($simID)**
   Global function from “db.class.php”. This function, “getSimBranding”, retrieves branding information associated with a specific scenario from the sim_branding_tbl database table.
3. **getScenarioCompetency($simID)**
   Global function from “db.class.php”. This function, "getScenarioCompetency", retrieves competency information associated with a specific scenario from the competency_tbl database table.
4. **webBranding($client_id)**
   Global function from “db.class.php”. The function "webBranding" generates CSS styles and JavaScript for branding a web page based on a client's preferences stored in a database.
5. **UID()**
   Global function from “db.class.php”. The "UID" function generates a unique identifier (UUID) using MySQL's UUID() function. If the query does not return a result, it falls back to generating a unique identifier using md5(time()).

6. **getTotalAttempt($simID, $userid)**
   Global function from “db.class.php”. This function will return total attempts for the user and simulation passed as arguments.
7. **getAssignSimOtherData($simID, $userid)**
   Global function from “db.class.php”. This PHP function, "getAssignSimOtherData", is designed to retrieve assignment data related to a simulation ("$sim_id") for a particular user ("$user_id").
8. **Questions Data**
   $dnd_choice_sql = "SELECT question_id, sub_option, image FROM sub_options_tbl WHERE question_id = ?";
   It retrieves data from a database table named sub_options_tbl for options for the question based on a specified question_id.

Third Party JS Library

1. ArticulateJS
2. OwlCarousel

JS Functions Used

1. **$('.options').click(function() {})**
   This jQuery code snippet sets up a click event handler for elements with the class options.
2. **$('.dragdrop').draggable({})**
   This Javascript allows elements with the class dragdrop to be dragged and dropped into elements with the class drop.
3. **$('body').on('click', '.submitbtn', function() {})**
   This code snippet manages the click event for submit buttons, loads appropriate content into a modal based on the type of question, and provides error handling if necessary. It utilizes data attributes and AJAX for dynamic content loading from *'view-feedback-modal.php'* and provides visual feedback to the user during the loading process.

Simulation > Hybrid Linear Multiple

Template Name: launch-sim-hybrid-linear-multiple.php

ImageToBePlacedHere

PHP Functions Used

1. **getScenario($sim_id)**
   Global function from “db.class.php”. This PHP function, "getScenario", retrieves scenario data from the database based on the provided scenario ID ($sim_id).
2. **getSimBranding($simID)**
   Global function from “db.class.php”. This function, “getSimBranding”, retrieves branding

information associated with a specific scenario from the sim_branding_tbl database table.

3. **getScenarioCompetency($simID)**
   Global function from “db.class.php”. This function, "getScenarioCompetency", retrieves competency information associated with a specific scenario from the competency_tbl database table.
4. **webBranding($client_id)**
   Global function from “db.class.php”. The function "webBranding" generates CSS styles and JavaScript for branding a web page based on a client's preferences stored in a database.
5. **UID()**
   Global function from “db.class.php”. The "UID" function generates a unique identifier (UUID) using MySQL's UUID() function. If the query does not return a result, it falls back to generating a unique identifier using md5(time()).
6. **getTotalAttempt($simID, $userid)**
   Global function from “db.class.php”. This function will return total attempts for the user and simulation passed as arguments.
7. **getAssignSimOtherData($simID, $userid)**
   Global function from “db.class.php”. This PHP function, "getAssignSimOtherData", is designed to retrieve assignment data related to a simulation ("$sim_id") for a particular user ("$user_id").
8. **Questions Data**
   $mtf_choice_sql = "SELECT answer_id, question_id, choice_option, match_option FROM answer_tbl WHERE question_id = ? ORDER BY (CASE WHEN choice_order_no != 0 THEN choice_order_no WHEN choice_order_no = 0 THEN RAND() END)";
   This SQL query retrieves information from the answer_tbl table related to multiple-choice matching type questions (often abbreviated as MTF or Matching Type Format).

   $mtf_match_sql = "SELECT answer_id, match_option FROM answer_tbl WHERE question_id = ? ORDER BY (CASE WHEN match_order_no != 0 THEN match_order_no WHEN match_order_no = 0 THEN RAND() END)";
   The query retrieves the answer_id and match_option columns from the answer_tbl table for rows where the question_id matches a provided value. The results are ordered such that rows with a non-zero match_order_no are sorted by that value, while rows with a match_order_no of 0 are placed in a random order. This approach ensures that certain matches appear in a specific order, while others appear randomly, which can be useful in scenarios where some items need a fixed order while others do not.

   $sq_sql = "SELECT answer_id, question_id, choice_option FROM answer_tbl WHERE question_id = ? ORDER BY RAND()";
   The query retrieves the answer_id, question_id, and choice_option columns from the answer_tbl table for rows where the question_id matches a provided value. The results are then randomly ordered each time the query is executed. This approach is useful when you want to retrieve random rows from a table, such as in situations where you

want to display random quiz questions or random choices to users.

$sort_choice_sql = "SELECT answer_id, question_id, choice_option, image FROM answer_tbl WHERE question_id = ?";
The query selects the answer_id, question_id, choice_option, and image columns from the answer_tbl table for rows where the question_id matches a provided value. This allows you to retrieve information about the choices for a specific question.

$sort_option_sql = "SELECT sub.sub_options_id, sub.sub_option FROM sub_options_tbl AS sub LEFT JOIN answer_tbl AS ans ON sub.answer_id = ans.answer_id WHERE ans.question_id = ? ".((! empty($shuffle)) ? "ORDER BY RAND() ": "");
The query retrieves the sub_options_id and sub_option columns from the sub_options_tbl table, joined with the answer_tbl table to retrieve options related to a specific question. If the $shuffle variable is not empty, the results are randomly ordered; otherwise, they are returned without any specific order.

$dnd_choice_sql = "SELECT question_id, sub_option, image FROM sub_options_tbl WHERE question_id = ?";
It retrieves data from a database table named sub_options_tbl for options for the question based on a specified question_id.

$dnd2_choice_sql = "SELECT answer_id, question_id, image, choice_option FROM answer_tbl WHERE question_id = ? ".(($shuffle == TRUE) ? "ORDER BY RAND()" : "");
The query selects the answer_id, question_id, image, and choice_option columns from the answer_tbl table for rows where the question_id matches a provided value. If the $shuffle variable is TRUE, the results are randomly ordered; otherwise, they are returned without any specific order. This approach allows for flexibility in sorting the results based on a condition.

$sql2dnd = "SELECT answer_id, image2, match_option FROM answer_tbl WHERE question_id = ? ".(($shuffle == TRUE) ? "ORDER BY RAND()" : "");
The query selects the answer_id, image2, and match_option columns from the answer_tbl table for rows where the question_id matches a provided value. If the $shuffle variable is TRUE, the results are randomly ordered; otherwise, they are returned without any specific order. This approach provides flexibility in sorting the results based on a condition.

Third Party JS Library

1. ArticulateJS
2. OwlCarousel

JS Functions Used

1. **$('.dragdrop').draggable({})**
   This code sets up draggable functionality for elements with the class dragdrop. It ensures that elements can be dragged, dropped, and swapped among valid drop targets, with additional features such as reverting to the original position when dropped outside a valid target, displaying a placeholder during dragging, and enabling scrolling while dragging.
2. **$('.dragdrop1').draggable({})**
   This code snippet enables elements with the class dragdrop1 to be draggable and dropped onto elements with the class drop1. It ensures that dropped elements are re-initialized as draggable and performs additional actions such as removing empty elements and refreshing the Owl Carousel after dropping.
3. **$('body').on('click', '.submitbtn', function() {})**
   This code snippet handles the click event for elements with the class submitbtn. It collects relevant data based on the clicked element and other elements on the page, prepares the data, loads content into a modal, and provides visual feedback to the user during the loading process.

JS File Used

File Path: content/js/launch-sim.js

1. **$('body').on('click', '#view_sim', function () {})**
   This JavaScript code uses jQuery to handle a click event on an element with the ID view_sim. When this element is clicked, it performs an AJAX request to *'view-sim-modal.php'* to load content into a modal window.
2. **$('.close-banner').click(function(){})**
   This JavaScript function using "SweetAlert" and jQuery to confirm the user's intention to close a simulation. If the user confirms, a loading overlay is shown, and after a delay, the user is redirected to a URL specified in the "href" attribute of an element with the ID "close-ban".
3. **$('body').on('click', 'a.qassets_icon', function(e){})**
   This JavaScript code is a jQuery event handler that uses Fancybox to display content when an anchor element with the class "qassets_icon" is clicked. It customizes Fancybox to prevent closing when clicking outside the modal or pressing the ESC key and applies specific transition effects.
4. **$('body').on('click', '.document', function(e){})**
   This JavaScript code is a jQuery event handler that uses Fancybox to display content when an element with the class document is clicked. It customizes Fancybox to display the content in an iframe with specific open and close effects, and ensures the content auto-sizes within the modal.
5. **speak(obj) {}**
   This JavaScript function "speak", controls text-to-speech functionality using the Articulate

plugin. It toggles between speaking, pausing, and resuming speech, and updates the display of play and pause icons accordingly.

6. **speak_feedback(obj) {}**
   The "speak_feedback" function manages text-to-speech functionality specifically for feedback, using the Articulate plugin. It toggles the speaking state (play/pause/resume) and updates the visibility of feedback-specific play and pause icons.

File Path: content/js/sim-branding.js

This JS file simply updates Background color on various elements on the page as per the JSON data fetched from the Server.

File Path: content/js/sim-temp.js

1. **const QuesTemp = (type, qid, aid = null, next = null, tabindex = null) => {}**
   The "QuesTemp" function is an asynchronous JavaScript function that fetches question template data from a server by sending AJAX request to *'includes/process.php'* and displays it in a modal. It uses jQuery for making AJAX requests and manipulating the DOM.
2. **$('body').on('click', '.launchAddData .naxtPreNav .preBtn', function(e){})**
   This jQuery script listens for a click event on the .preBtn button inside the .naxtPreNav navigation container within the .launchAddData section. When the button is clicked, it decreases the actSlNo variable by 1 to navigate to the previous slide. It also ensures actSlNo does not become negative, and then it calls the MakeSlideActive function to activate the corresponding slide.
3. **$('body').on('click', '.launchAddData .naxtPreNav .nextBtn', function(e){})**
   This jQuery script listens for a click event on an element and handles the navigation to the next slide in a series. It increments the actSlNo variable to move to the next slide, ensures it does not exceed the maximum number of slides, and calls the MakeSlideActive function to activate the corresponding slide.
4. **function MakeSlideActive(index, next = null) {}**
   The "MakeSlideActive" function manages the state of the slides within a modal by adding and removing classes to highlight the active slide. It also enables or disables navigation buttons based on the current slide index and handles special behavior when the last slide is reached.
5. **$('body').on('click', '.feedclose', function(e){})**
   The code defines an event handler for the ".feedclose" element. When this element is clicked, the function performs several actions, including preventing the default action, retrieving data attributes, hiding a modal, resetting a slide counter, and calling a function "QuesTemp" to load new content.
6. **$('body').on('click', '.qftemp_close', function(e){})**
   The code defines an event handler for the ".qftemp_close" element. When this element is clicked, the function performs several actions, including preventing the default action,

retrieving data attributes, hiding a modal, resetting a slide counter, and calling a function "QuesTemp" to load new content.

File Path: content/js/launch-sim-hybrid-linear-classic.js

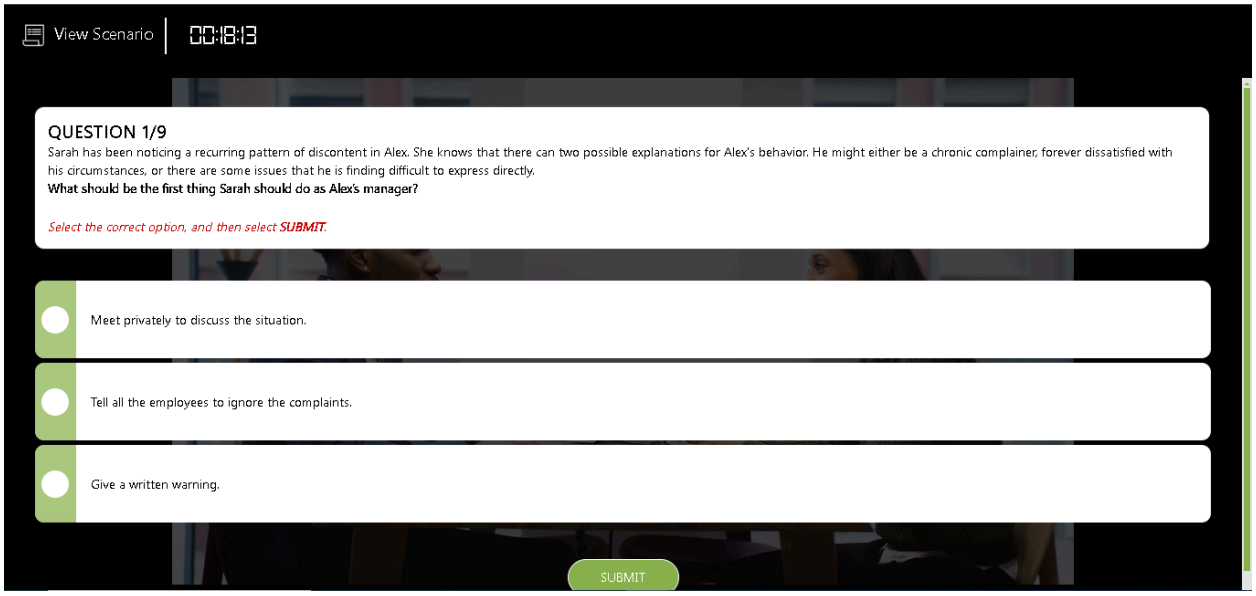
1. **$('.preview_hybrid').on('click', function(){})**
   This code snippet handles the click event for elements with the class preview_hybrid. When clicked, it shows a specific button and displays a warning notification using SweetAlert.
2. **$('body').on('click', '.submitbtn_hybrid_preview', function() {})**
   This code snippet manages the click event for elements with the class submitbtn_hybrid_preview. Depending on the data attributes of the clicked element, it triggers different actions, such as showing/hiding elements, displaying success messages, and handling video playback and interactions.
3. **$('body').on('click', '.add-tts', function() {})**
   This code snippet handles the click event for elements with the class add-tts. It loads content into a modal using AJAX *('tts-hybrid-modal.php' with parameters {'box_id': box_id, 'qid': qid, 'type': type})* based on the data attributes of the clicked element (box_id, qid, type), shows the modal if the content is successfully loaded, and displays an error message if the content is empty. Additionally, it provides visual feedback to the user during the loading process.
4. **$('body').on('click', '.add-code', function() {})**
   This code snippet handles the click event for elements with the class add-code. It loads content into a modal using AJAX *('code-hybrid-modal.php' with parameters {'box_id': box_id, 'qid': qid, 'type': type})* based on the data attributes of the clicked element (box_id, qid, type), shows the modal if the content is successfully loaded, and displays an error message if the content is empty.
5. **$('body').on('click', '.rec-audio', function() {})**
   This code snippet handles the click event for elements with the class rec-audio. It loads content into a modal using AJAX *('rec-audio-hybrid-modal.php' with parameters {'box_id': box_id, 'qid': qid, 'type': type})* based on the data attributes of the clicked element (box_id, qid, type). If the content is successfully loaded, it displays the modal; otherwise, it shows an error message.
6. **$('body').on('click', '.rec-video', function() {})**
   This code snippet handles the click event for elements with the class rec-video. It loads content into a modal using AJAX *('rec-video-hybrid-modal.php' with parameters {'box_id': box_id, 'qid': qid, 'type': type})* based on the data attributes of the clicked element (box_id, qid, type). If the content is successfully loaded, it displays the modal; otherwise, it shows an error message.
7. **$('body').on('click', '.rec-screen', function() {})**
   This code snippet handles the click event for elements with the class rec-screen. It loads content into a modal using AJAX *('rec-screen-hybrid-modal.php' with parameters {'box_id': box_id, 'qid': qid, 'type': type})* based on the data attributes of the clicked

element (box_id, qid, type). If the content is successfully loaded, it displays the modal; otherwise, it shows an error message.

8. **$('body').on('click', '.delete_assets', function() {})**
   This code snippet handles the click event for elements with the class delete_assets. It presents a confirmation dialog using SweetAlert to confirm the deletion of an asset. Upon confirmation, it sends an AJAX *('includes/delete-file.php')* request to delete the file. Depending on the response, it may update the UI elements and display appropriate messages. If the user cancels the deletion, it shows a message indicating that the file is safe.
9. **$('body').on('click', '.delete_tts', function() {})**
   This JavaScript code snippet handles the click event for elements with the class delete_tts. It presents a confirmation dialog using SweetAlert to confirm the deletion of text data. Upon confirmation, it performs the deletion process by clearing the value of an input field and hiding/showing certain elements. It also provides visual feedback to the user during the deletion process with a loading overlay. If the user cancels the deletion, it shows a message indicating that the text data is safe.
10. **$('body').on('click', '.submitbtn_hybrid', function() {})**
    This JavaScript code snippet handles the submission of hybrid questions in a learning application. It processes different types of responses (file uploads, text, code) based on the question type and handles report generation and navigation to the next question. The script interacts with server-side processing for data analysis and database storage, providing real-time feedback to users and managing the progression through the question set.

Simulation > Hybrid Linear Classic Video

Template Name: launch-sim-hybrid-linear-classic-video.php



PHP Functions Used

1. **getScenario($sim_id)**
   Global function from “db.class.php”. This PHP function, "getScenario", retrieves scenario data from the database based on the provided scenario ID ($sim_id).
2. **getSimBranding($simID)**
   Global function from “db.class.php”. This function, “getSimBranding”, retrieves branding information associated with a specific scenario from the sim_branding_tbl database table.
3. **getScenarioCompetency($simID)**
   Global function from “db.class.php”. This function, "getScenarioCompetency", retrieves competency information associated with a specific scenario from the competency_tbl database table.
4. **webBranding($client_id)**
   Global function from “db.class.php”. The function "webBranding" generates CSS styles and JavaScript for branding a web page based on a client's preferences stored in a database.
5. **UID()**
   Global function from “db.class.php”. The "UID" function generates a unique identifier (UUID) using MySQL's UUID() function. If the query does not return a result, it falls back to generating a unique identifier using md5(time()).
6. **getTotalAttempt($simID, $userid)**
   Global function from “db.class.php”. This function will return total attempts for the user and simulation passed as arguments.
7. **getAssignSimOtherData($simID, $userid)**
   Global function from “db.class.php”. This PHP function, "getAssignSimOtherData", is designed to retrieve assignment data related to a simulation ("$sim_id") for a particular user ("$user_id").
8. **Questions Data**
   *$qSql = $db->prepare("SELECT * FROM question_tbl WHERE scenario_id = ? ORDER BY CASE WHEN qorder <> 0 THEN qorder ELSE question_id END ASC");*
   This SQL statements select data from a table based on a condition (scenario_id) and sorts the results by a combination of a custom order (qorder) and the default order (question_id).

   $dnd_choice_sql = "SELECT question_id, sub_option, image FROM sub_options_tbl WHERE question_id = ?";
   It retrieves data from a database table named sub_options_tbl for options for the question based on a specified question_id.

Third Party JS Library

1. ArticulateJS
2. OwlCarousel

JS Functions Used

1. **$('.dragdrop').draggable({})**
   This code enables draggable behavior for elements with the class dragdrop, allowing them to be dragged within a specific context and handling the drop logic accordingly.
2. **$('body').on('click', '.submitbtn', function(){})**
   This code facilitates the submission process for different types of questions by loading appropriate feedback content based on the question type.

JS File Used

File Path: content/js/launch-sim.js

1. **$('body').on('click', '#view_sim', function () {})**
   This JavaScript code uses jQuery to handle a click event on an element with the ID view_sim. When this element is clicked, it performs an AJAX request to *'view-sim-modal.php'* to load content into a modal window.
2. **$('.close-banner').click(function(){})**
   This JavaScript function using "SweetAlert" and jQuery to confirm the user's intention to close a simulation. If the user confirms, a loading overlay is shown, and after a delay, the user is redirected to a URL specified in the "href" attribute of an element with the ID "close-ban".
3. **$('body').on('click', 'a.qassets_icon', function(e){})**
   This JavaScript code is a jQuery event handler that uses Fancybox to display content when an anchor element with the class "qassets_icon" is clicked. It customizes Fancybox to prevent closing when clicking outside the modal or pressing the ESC key and applies specific transition effects.
4. **$('body').on('click', '.document', function(e){})**
   This JavaScript code is a jQuery event handler that uses Fancybox to display content when an element with the class document is clicked. It customizes Fancybox to display the content in an iframe with specific open and close effects, and ensures the content auto-sizes within the modal.
5. **speak(obj) {}**
   This JavaScript function "speak", controls text-to-speech functionality using the Articulate plugin. It toggles between speaking, pausing, and resuming speech, and updates the display of play and pause icons accordingly.
6. **speak_feedback(obj) {}**
   The "speak_feedback" function manages text-to-speech functionality specifically for feedback, using the Articulate plugin. It toggles the speaking state (play/pause/resume) and updates the visibility of feedback-specific play and pause icons.

File Path: content/js/sim-branding.js

This JS file simply updates Background color on various elements on the page as per the JSON data fetched from the Server.

File Path: content/js/sim-temp.js

1. **const QuesTemp = (type, qid, aid = null, next = null, tabindex = null) => {}**
   The "QuesTemp" function is an asynchronous JavaScript function that fetches question template data from a server by sending AJAX request to *'includes/process.php'* and displays it in a modal. It uses jQuery for making AJAX requests and manipulating the DOM.
2. **$('body').on('click', '.launchAddData .naxtPreNav .preBtn', function(e){})**
   This jQuery script listens for a click event on the .preBtn button inside the .naxtPreNav navigation container within the .launchAddData section. When the button is clicked, it decreases the actSlNo variable by 1 to navigate to the previous slide. It also ensures actSlNo does not become negative, and then it calls the MakeSlideActive function to activate the corresponding slide.
3. **$('body').on('click', '.launchAddData .naxtPreNav .nextBtn', function(e){})**
   This jQuery script listens for a click event on an element and handles the navigation to the next slide in a series. It increments the actSlNo variable to move to the next slide, ensures it does not exceed the maximum number of slides, and calls the MakeSlideActive function to activate the corresponding slide.
4. **function MakeSlideActive(index, next = null) {}**
   The "MakeSlideActive" function manages the state of the slides within a modal by adding and removing classes to highlight the active slide. It also enables or disables navigation buttons based on the current slide index and handles special behavior when the last slide is reached.
5. **$('body').on('click', '.feedclose', function(e){})**
   The code defines an event handler for the ".feedclose" element. When this element is clicked, the function performs several actions, including preventing the default action, retrieving data attributes, hiding a modal, resetting a slide counter, and calling a function "QuesTemp" to load new content.
6. **$('body').on('click', '.qftemp_close', function(e){})**
   The code defines an event handler for the ".qftemp_close" element. When this element is clicked, the function performs several actions, including preventing the default action, retrieving data attributes, hiding a modal, resetting a slide counter, and calling a function "QuesTemp" to load new content.

File Path: content/js/launch-sim-hybrid-linear-classic.js

1. **$('.preview_hybrid').on('click', function(){})**
   This code snippet handles the click event for elements with the class preview_hybrid. When clicked, it shows a specific button and displays a warning notification using SweetAlert.
2. **$('body').on('click', '.submitbtn_hybrid_preview', function() {})**
   This code snippet manages the click event for elements with the class submitbtn_hybrid_preview. Depending on the data attributes of the clicked element, it

triggers different actions, such as showing/hiding elements, displaying success messages, and handling video playback and interactions.

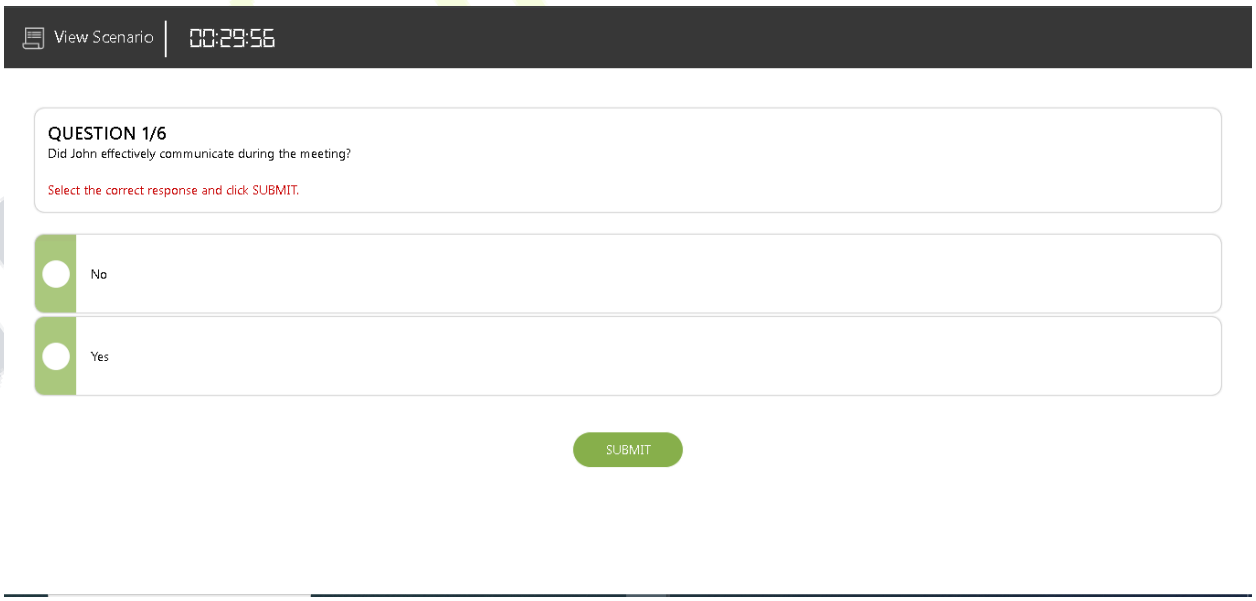
3. **$('body').on('click', '.add-tts', function() {})**
   This code snippet handles the click event for elements with the class add-tts. It loads content into a modal using AJAX *('tts-hybrid-modal.php' with parameters {'box_id': box_id, 'qid': qid, 'type': type})* based on the data attributes of the clicked element (box_id, qid, type), shows the modal if the content is successfully loaded, and displays an error message if the content is empty. Additionally, it provides visual feedback to the user during the loading process.
4. **$('body').on('click', '.add-code', function() {})**
   This code snippet handles the click event for elements with the class add-code. It loads content into a modal using AJAX *('code-hybrid-modal.php' with parameters {'box_id': box_id, 'qid': qid, 'type': type})* based on the data attributes of the clicked element (box_id, qid, type), shows the modal if the content is successfully loaded, and displays an error message if the content is empty.
5. **$('body').on('click', '.rec-audio', function() {})**
   This code snippet handles the click event for elements with the class rec-audio. It loads content into a modal using AJAX *('rec-audio-hybrid-modal.php' with parameters {'box_id': box_id, 'qid': qid, 'type': type})* based on the data attributes of the clicked element (box_id, qid, type). If the content is successfully loaded, it displays the modal; otherwise, it shows an error message.
6. **$('body').on('click', '.rec-video', function() {})**
   This code snippet handles the click event for elements with the class rec-video. It loads content into a modal using AJAX *('rec-video-hybrid-modal.php' with parameters {'box_id': box_id, 'qid': qid, 'type': type})* based on the data attributes of the clicked element (box_id, qid, type). If the content is successfully loaded, it displays the modal; otherwise, it shows an error message.
7. **$('body').on('click', '.rec-screen', function() {})**
   This code snippet handles the click event for elements with the class rec-screen. It loads content into a modal using AJAX *('rec-screen-hybrid-modal.php' with parameters {'box_id': box_id, 'qid': qid, 'type': type})* based on the data attributes of the clicked element (box_id, qid, type). If the content is successfully loaded, it displays the modal; otherwise, it shows an error message.
8. **$('body').on('click', '.delete_assets', function() {})**
   This code snippet handles the click event for elements with the class delete_assets. It presents a confirmation dialog using SweetAlert to confirm the deletion of an asset. Upon confirmation, it sends an AJAX *('includes/delete-file.php')* request to delete the file. Depending on the response, it may update the UI elements and display appropriate messages. If the user cancels the deletion, it shows a message indicating that the file is safe.
9. **$('body').on('click', '.delete_tts', function() {})**
   This JavaScript code snippet handles the click event for elements with the class delete_tts. It presents a confirmation dialog using SweetAlert to confirm the deletion of text data. Upon confirmation, it performs the deletion process by clearing the value of an

input field and hiding/showing certain elements. It also provides visual feedback to the user during the deletion process with a loading overlay. If the user cancels the deletion, it shows a message indicating that the text data is safe.

10. **$('body').on('click', '.submitbtn_hybrid', function() {})**
    This JavaScript code snippet handles the submission of hybrid questions in a learning application. It processes different types of responses (file uploads, text, code) based on the question type and handles report generation and navigation to the next question. The script interacts with server-side processing for data analysis and database storage, providing real-time feedback to users and managing the progression through the question set.

Simulation > Hybrid Linear Multiple Video

Template Name: launch-sim-hybrid-linear-multiple-video.php



PHP Functions Used

1. **getScenario($sim_id)**
   Global function from “db.class.php”. This PHP function, "getScenario", retrieves scenario data from the database based on the provided scenario ID ($sim_id).
2. **getSimBranding($simID)**
   Global function from “db.class.php”. This function, “getSimBranding”, retrieves branding information associated with a specific scenario from the sim_branding_tbl database table.
3. **getScenarioCompetency($simID)**
   Global function from “db.class.php”. This function, "getScenarioCompetency", retrieves competency information associated with a specific scenario from the competency_tbl database table.

4. **webBranding($client_id)**
   Global function from “db.class.php”. The function "webBranding" generates CSS styles and JavaScript for branding a web page based on a client's preferences stored in a database.
5. **UID()**
   Global function from “db.class.php”. The "UID" function generates a unique identifier (UUID) using MySQL's UUID() function. If the query does not return a result, it falls back to generating a unique identifier using md5(time()).
6. **getTotalAttempt($simID, $userid)**
   Global function from “db.class.php”. This function will return total attempts for the user and simulation passed as arguments.
7. **getAssignSimOtherData($simID, $userid)**
   Global function from “db.class.php”. This PHP function, "getAssignSimOtherData", is designed to retrieve assignment data related to a simulation ("$sim_id") for a particular user ("$user_id").
8. **Questions Data**
   *$qSql = $db->prepare("SELECT * FROM question_tbl WHERE scenario_id = ? ORDER BY qorder = 0, qorder");*
   The line of code prepares an SQL statement to select all records from the "question_tbl" table where the "scenario_id" matches a specified value. The results are ordered such that rows with qorder equal to 0 appear first, followed by sorting based on the "qorder" column. This prepared statement can be executed with a specific "scenario_id" value later in the code.

   $mtf_choice_sql = "SELECT answer_id, question_id, choice_option, match_option FROM answer_tbl WHERE question_id = ? ORDER BY (CASE WHEN choice_order_no != 0 THEN choice_order_no WHEN choice_order_no = 0 THEN RAND() END)";
   This SQL query retrieves information from the answer_tbl table related to multiple-choice matching type questions (often abbreviated as MTF or Matching Type Format).

   $mtf_match_sql = "SELECT answer_id, match_option FROM answer_tbl WHERE question_id = ? ORDER BY (CASE WHEN match_order_no != 0 THEN match_order_no WHEN match_order_no = 0 THEN RAND() END)";
   The query retrieves the answer_id and match_option columns from the answer_tbl table for rows where the question_id matches a provided value. The results are ordered such that rows with a non-zero match_order_no are sorted by that value, while rows with a match_order_no of 0 are placed in a random order. This approach ensures that certain matches appear in a specific order, while others appear randomly, which can be useful in scenarios where some items need a fixed order while others do not.

   $sq_sql = "SELECT answer_id, question_id, choice_option FROM answer_tbl WHERE question_id = ? ORDER BY RAND()";
   The query retrieves the answer_id, question_id, and choice_option columns from the answer_tbl table for rows where the question_id matches a provided value. The results

are then randomly ordered each time the query is executed. This approach is useful when you want to retrieve random rows from a table, such as in situations where you want to display random quiz questions or random choices to users.

$sort_choice_sql = "SELECT answer_id, question_id, choice_option, image FROM answer_tbl WHERE question_id = ?";
The query selects the answer_id, question_id, choice_option, and image columns from the answer_tbl table for rows where the question_id matches a provided value. This allows you to retrieve information about the choices for a specific question.

$sort_option_sql = "SELECT sub.sub_options_id, sub.sub_option FROM sub_options_tbl AS sub LEFT JOIN answer_tbl AS ans ON sub.answer_id = ans.answer_id WHERE ans.question_id = ? ".((! empty($shuffle)) ? "ORDER BY RAND() ": "");
The query retrieves the sub_options_id and sub_option columns from the sub_options_tbl table, joined with the answer_tbl table to retrieve options related to a specific question. If the $shuffle variable is not empty, the results are randomly ordered; otherwise, they are returned without any specific order.

$dnd_choice_sql = "SELECT question_id, sub_option, image FROM sub_options_tbl WHERE question_id = ?";
It retrieves data from a database table named sub_options_tbl for options for the question based on a specified question_id.

$dnd2_choice_sql = "SELECT answer_id, question_id, image, choice_option FROM answer_tbl WHERE question_id = ? ".(($shuffle == TRUE) ? "ORDER BY RAND()" : "");
The query selects the answer_id, question_id, image, and choice_option columns from the answer_tbl table for rows where the question_id matches a provided value. If the $shuffle variable is TRUE, the results are randomly ordered; otherwise, they are returned without any specific order. This approach allows for flexibility in sorting the results based on a condition.

$sql2dnd = "SELECT answer_id, image2, match_option FROM answer_tbl WHERE question_id = ? ".(($shuffle == TRUE) ? "ORDER BY RAND()" : "");
The query selects the answer_id, image2, and match_option columns from the answer_tbl table for rows where the question_id matches a provided value. If the $shuffle variable is TRUE, the results are randomly ordered; otherwise, they are returned without any specific order. This approach provides flexibility in sorting the results based on a condition.

Third Party JS Library

1. ArticulateJS
2. OwlCarousel

JS Functions Used

1. **$('.dragdrop').draggable({})**
   This code enables drag-and-drop functionality for elements with the class dragdrop, allowing them to be moved and rearranged within droppable targets. If a target is empty, the element is dropped there. If the target already contains an element, swapping occurs between the dragged element and the existing one.
2. **$('body').on('click', '.submitbtn', function() {})**
   This code handles the submission of form data related to different question types and displays appropriate feedback modals based on the response received from the server.

JS File Used

File Path: content/js/launch-sim.js

1. **$('body').on('click', '#view_sim', function () {})**
   This JavaScript code uses jQuery to handle a click event on an element with the ID view_sim. When this element is clicked, it performs an AJAX request to *'view-sim-modal.php'* to load content into a modal window.
2. **$('.close-banner').click(function(){})**
   This JavaScript function using "SweetAlert" and jQuery to confirm the user's intention to close a simulation. If the user confirms, a loading overlay is shown, and after a delay, the user is redirected to a URL specified in the "href" attribute of an element with the ID "close-ban".
3. **$('body').on('click', 'a.qassets_icon', function(e){})**
   This JavaScript code is a jQuery event handler that uses Fancybox to display content when an anchor element with the class "qassets_icon" is clicked. It customizes Fancybox to prevent closing when clicking outside the modal or pressing the ESC key and applies specific transition effects.
4. **$('body').on('click', '.document', function(e){})**
   This JavaScript code is a jQuery event handler that uses Fancybox to display content when an element with the class document is clicked. It customizes Fancybox to display the content in an iframe with specific open and close effects, and ensures the content auto-sizes within the modal.
5. **speak(obj) {}**
   This JavaScript function "speak", controls text-to-speech functionality using the Articulate plugin. It toggles between speaking, pausing, and resuming speech, and updates the display of play and pause icons accordingly.
6. **speak_feedback(obj) {}**
   The "speak_feedback" function manages text-to-speech functionality specifically for feedback, using the Articulate plugin. It toggles the speaking state (play/pause/resume) and updates the visibility of feedback-specific play and pause icons.

File Path: content/js/sim-branding.js

This JS file simply updates Background color on various elements on the page as per the JSON data fetched from the Server.

File Path: content/js/sim-temp.js

1. **const QuesTemp = (type, qid, aid = null, next = null, tabindex = null) => {}**
   The "QuesTemp" function is an asynchronous JavaScript function that fetches question template data from a server by sending AJAX request to *'includes/process.php'* and displays it in a modal. It uses jQuery for making AJAX requests and manipulating the DOM.
2. **$('body').on('click', '.launchAddData .naxtPreNav .preBtn', function(e){})**
   This jQuery script listens for a click event on the .preBtn button inside the .naxtPreNav navigation container within the .launchAddData section. When the button is clicked, it decreases the actSlNo variable by 1 to navigate to the previous slide. It also ensures actSlNo does not become negative, and then it calls the MakeSlideActive function to activate the corresponding slide.
3. **$('body').on('click', '.launchAddData .naxtPreNav .nextBtn', function(e){})**
   This jQuery script listens for a click event on an element and handles the navigation to the next slide in a series. It increments the actSlNo variable to move to the next slide, ensures it does not exceed the maximum number of slides, and calls the MakeSlideActive function to activate the corresponding slide.
4. **function MakeSlideActive(index, next = null) {}**
   The "MakeSlideActive" function manages the state of the slides within a modal by adding and removing classes to highlight the active slide. It also enables or disables navigation buttons based on the current slide index and handles special behavior when the last slide is reached.
5. **$('body').on('click', '.feedclose', function(e){})**
   The code defines an event handler for the ".feedclose" element. When this element is clicked, the function performs several actions, including preventing the default action, retrieving data attributes, hiding a modal, resetting a slide counter, and calling a function "QuesTemp" to load new content.
6. **$('body').on('click', '.qftemp_close', function(e){})**
   The code defines an event handler for the ".qftemp_close" element. When this element is clicked, the function performs several actions, including preventing the default action, retrieving data attributes, hiding a modal, resetting a slide counter, and calling a function "QuesTemp" to load new content.

File Path: content/js/launch-sim-hybrid-linear-classic.js

1. **$('.preview_hybrid').on('click', function(){})**
   This code snippet handles the click event for elements with the class preview_hybrid.

When clicked, it shows a specific button and displays a warning notification using SweetAlert.

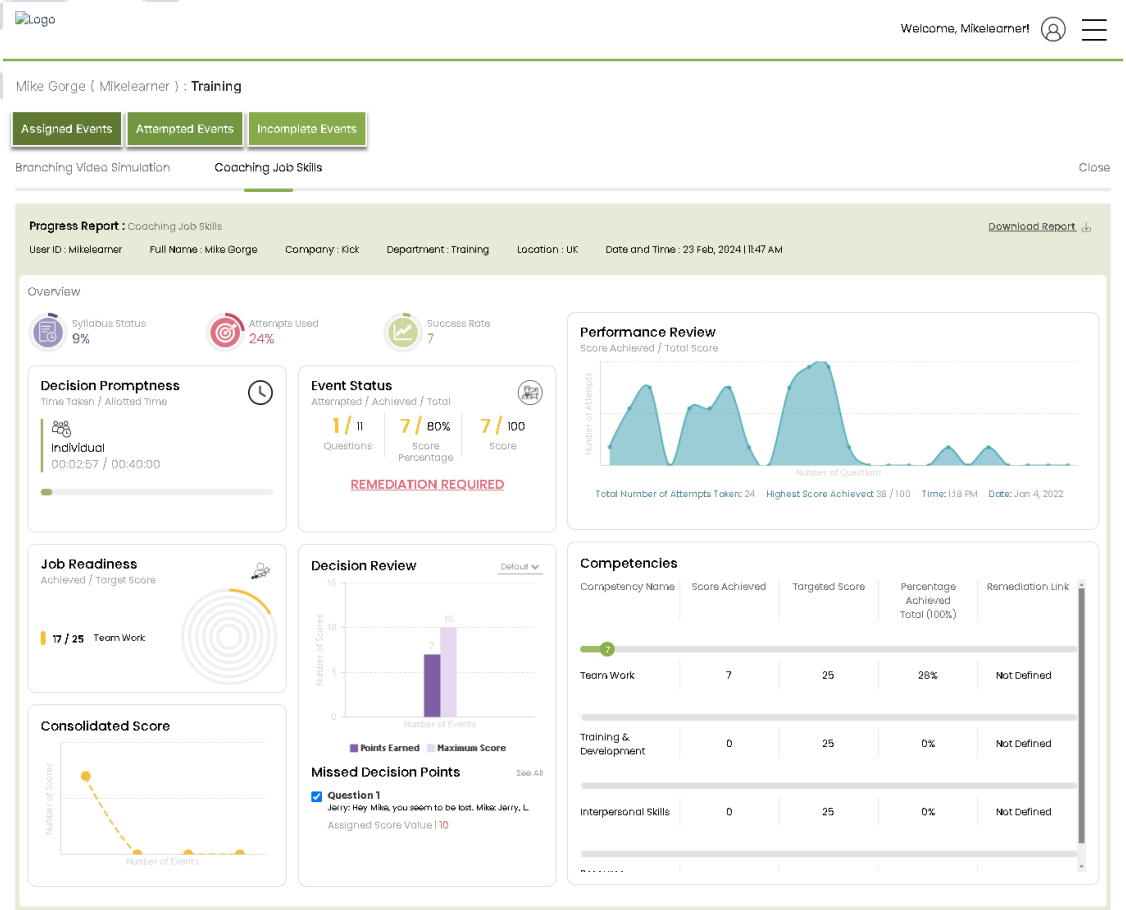
2. **$('body').on('click', '.submitbtn_hybrid_preview', function() {})**
   This code snippet manages the click event for elements with the class submitbtn_hybrid_preview. Depending on the data attributes of the clicked element, it triggers different actions, such as showing/hiding elements, displaying success messages, and handling video playback and interactions.
3. **$('body').on('click', '.add-tts', function() {})**
   This code snippet handles the click event for elements with the class add-tts. It loads content into a modal using AJAX *('tts-hybrid-modal.php' with parameters {'box_id': box_id, 'qid': qid, 'type': type})* based on the data attributes of the clicked element (box_id, qid, type), shows the modal if the content is successfully loaded, and displays an error message if the content is empty. Additionally, it provides visual feedback to the user during the loading process.
4. **$('body').on('click', '.add-code', function() {})**
   This code snippet handles the click event for elements with the class add-code. It loads content into a modal using AJAX *('code-hybrid-modal.php' with parameters {'box_id': box_id, 'qid': qid, 'type': type})* based on the data attributes of the clicked element (box_id, qid, type), shows the modal if the content is successfully loaded, and displays an error message if the content is empty.
5. **$('body').on('click', '.rec-audio', function() {})**
   This code snippet handles the click event for elements with the class rec-audio. It loads content into a modal using AJAX *('rec-audio-hybrid-modal.php' with parameters {'box_id': box_id, 'qid': qid, 'type': type})* based on the data attributes of the clicked element (box_id, qid, type). If the content is successfully loaded, it displays the modal; otherwise, it shows an error message.
6. **$('body').on('click', '.rec-video', function() {})**
   This code snippet handles the click event for elements with the class rec-video. It loads content into a modal using AJAX *('rec-video-hybrid-modal.php' with parameters {'box_id': box_id, 'qid': qid, 'type': type})* based on the data attributes of the clicked element (box_id, qid, type). If the content is successfully loaded, it displays the modal; otherwise, it shows an error message.
7. **$('body').on('click', '.rec-screen', function() {})**
   This code snippet handles the click event for elements with the class rec-screen. It loads content into a modal using AJAX *('rec-screen-hybrid-modal.php' with parameters {'box_id': box_id, 'qid': qid, 'type': type})* based on the data attributes of the clicked element (box_id, qid, type). If the content is successfully loaded, it displays the modal; otherwise, it shows an error message.
8. **$('body').on('click', '.delete_assets', function() {})**
   This code snippet handles the click event for elements with the class delete_assets. It presents a confirmation dialog using SweetAlert to confirm the deletion of an asset. Upon confirmation, it sends an AJAX *('includes/delete-file.php')* request to delete the file. Depending on the response, it may update the UI elements and display appropriate

messages. If the user cancels the deletion, it shows a message indicating that the file is safe.

9. **$('body').on('click', '.delete_tts', function() {})**
   This JavaScript code snippet handles the click event for elements with the class delete_tts. It presents a confirmation dialog using SweetAlert to confirm the deletion of text data. Upon confirmation, it performs the deletion process by clearing the value of an input field and hiding/showing certain elements. It also provides visual feedback to the user during the deletion process with a loading overlay. If the user cancels the deletion, it shows a message indicating that the text data is safe.
10. **$('body').on('click', '.submitbtn_hybrid', function() {})**
    This JavaScript code snippet handles the submission of hybrid questions in a learning application. It processes different types of responses (file uploads, text, code) based on the question type and handles report generation and navigation to the next question. The script interacts with server-side processing for data analysis and database storage, providing real-time feedback to users and managing the progression through the question set.

Simulation > Learner Progress

Template Name: learner-progress-new.php



Third Party JS Library

1. JSPDF
2. HTML3Canvas
3. GreenAudioPlayer

JS File Used

File Path: content/js/learner-progress-report.js

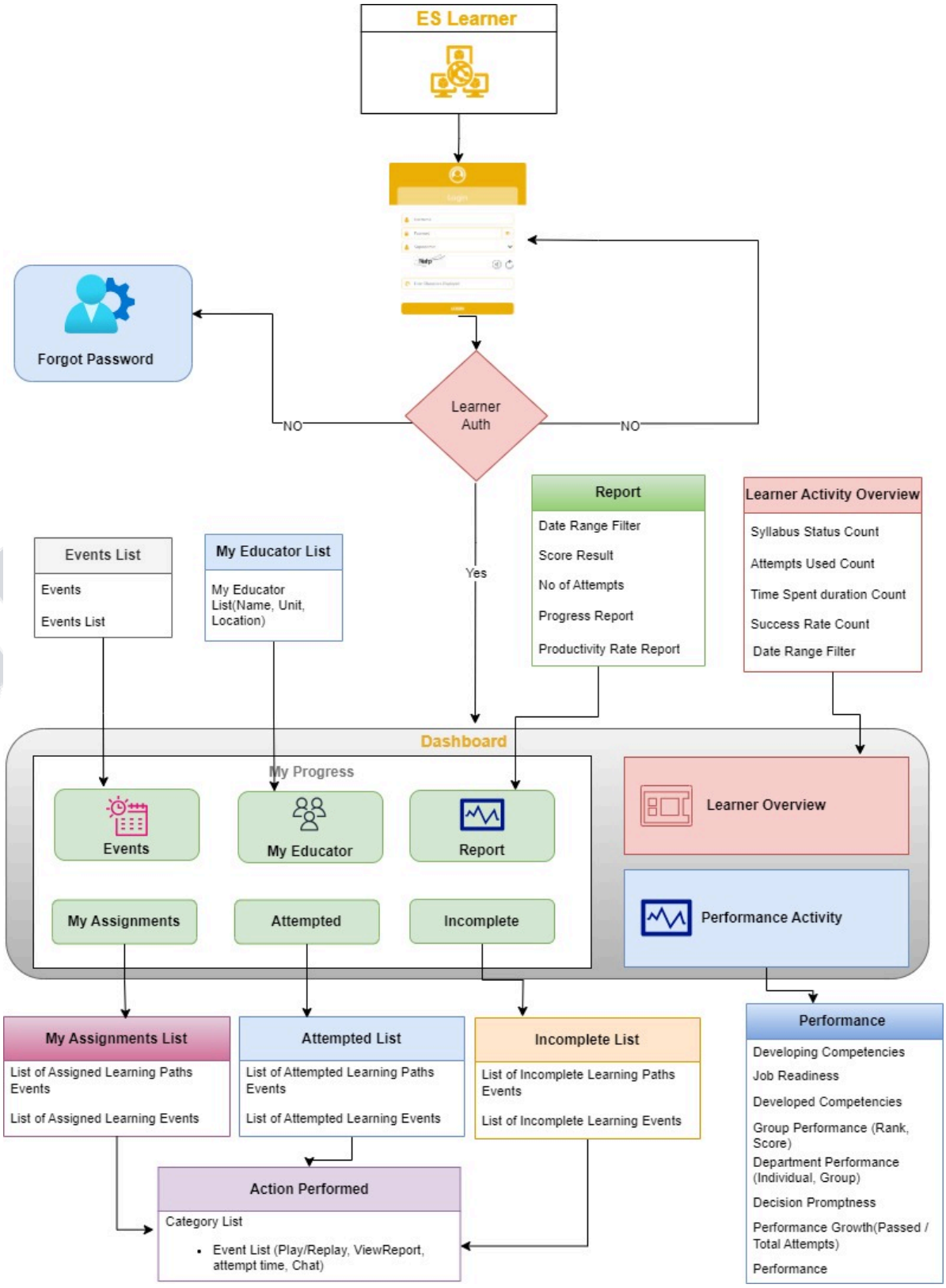
1. **const getPercentage = (x, y, html = false) => {}**
   This JavaScript code defines a function named "getPercentage" that calculates the percentage of x relative to y, optionally returning the result as a formatted string with a percentage sign.
2. **const calculateMinutes = (timeArray) => {}**
   This JavaScript code defines a function named "calculateMinutes". It calculates the total number of minutes from a time string in the format "HH".
3. **const progressBar = () => {}**
   This JavaScript code defines a function named "progressBar" that updates the width of progress bar elements on a webpage based on a data attribute.
4. **const CircleBar = () => {}**
   The "CircleBar" function initializes circular progress bars on elements with specific classes by setting the "stroke-dashoffset" CSS property based on a data attribute.
5. **const parseTime = (time) => {}**
   This JavaScript code defines a function named "parseTime" that takes a time string in the format "HH:MM" and returns a formatted string displaying the hours and minutes with HTML bold tags.
6. **const getUsername = (text) => {}**
   This JavaScript code defines a function named "getUsername". It extracts a username from a text string that matches a specific pattern, or generates a random username if no match is found.
7. **const getPDF = (id, title) => {}**
   The "getPDF" function converts a specified HTML element into a multi-page PDF document, adds a user-specific filename, and allows the user to download it. This function leverages "html2canvas" to capture the HTML element as an image and jsPDF to create and save the PDF.
8. **const getFileExtension = (file) => {}**
   The function takes a file name string as input, checks if it's a valid string, and then extracts the file extension from it. If no extension is found, it returns an empty string.
9. **function getUrlVars() {}**
   This function getUrlVars extracts query parameters from the current URL and returns them as an object, where keys represent parameter names, and values represent parameter values.
10. **const toolTipFormatter = (e) => {}**
    The toolTipFormatter function formats tooltips for a chart by iterating over data entries,

accumulating total values, and generating additional information. The function then returns the formatted tooltip content.

11. **$("#feedback_form").on('submit', (function (e) {})**
    This code essentially handles form submission asynchronously via AJAX request to *'includes/process.php'*, providing user feedback using SweetAlert for both success and error scenarios.
12. **$('.close_report').on('click', function (e){})**
    This code essentially presents a confirmation dialog to the user when they click on elements with the class close_report, allowing them to choose between submitting an experience or closing the tab/window.
13. **$('.back_report').on('click', function (e){})**
    This code essentially presents a confirmation dialog to the user when they click on elements with the class back_report, allowing them to choose between submitting an experience or going back to assigned events.

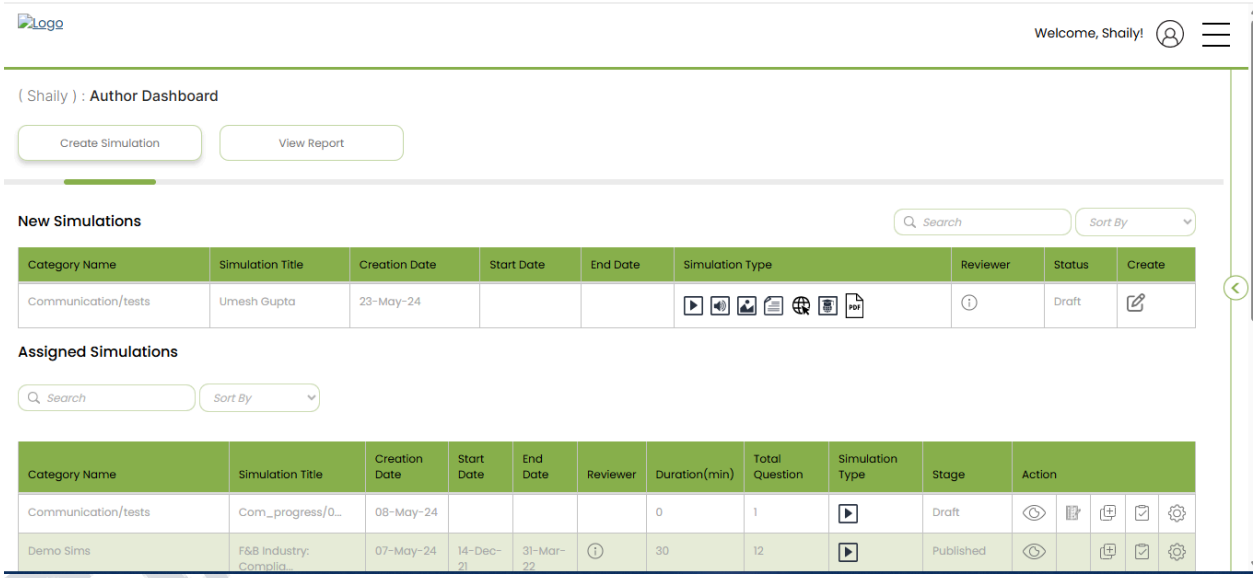
EasySIM®

Learner DFD


ES Learner
Forgot Password
NO
Learner Auth
NO
Yes
Report
Date Range Filter
Score Result
No of Attempts
Progress Report
Productivity Rate Report
Learner Activity Overview
Syllabus Status Count
Attempts Used Count
Time Spent duration Count
Success Rate Count
Date Range Filter
Events List
Events
Events List
My Educator List
My Educator List(Name, Unit, Location)
Dashboard
My Progress
Events
My Educator
Report
My Assignments
Attempted
Incomplete
Learner Overview
Performance Activity
My Assignments List
List of Assigned Learning Paths Events
List of Assigned Learning Events
Attempted List
List of Attempted Learning Paths Events
List of Attempted Learning Events
Incomplete List
List of Incomplete Learning Paths Events
List of Incomplete Learning Events
Performance
Developing Competencies
Job Readiness
Developed Competencies
Group Performance (Rank, Score)
Department Performance (Individual, Group)
Decision Promptness
Performance Growth(Passed / Total Attempts)
Performance
Action Performed
Category List
Event List (Play/Replay, ViewReport, attempt time, Chat)

Author

Overview



This domain will only be accessible to users with **Author** credentials. It will allow management of New Simulations and Assigned Simulations on the Dashboard's "Create Simulation" tab. Another tab, named "View Report," will enable users to view all reports related to "**Simulation Update**," "**Publish & Review**," and "**Assigned Learner**."

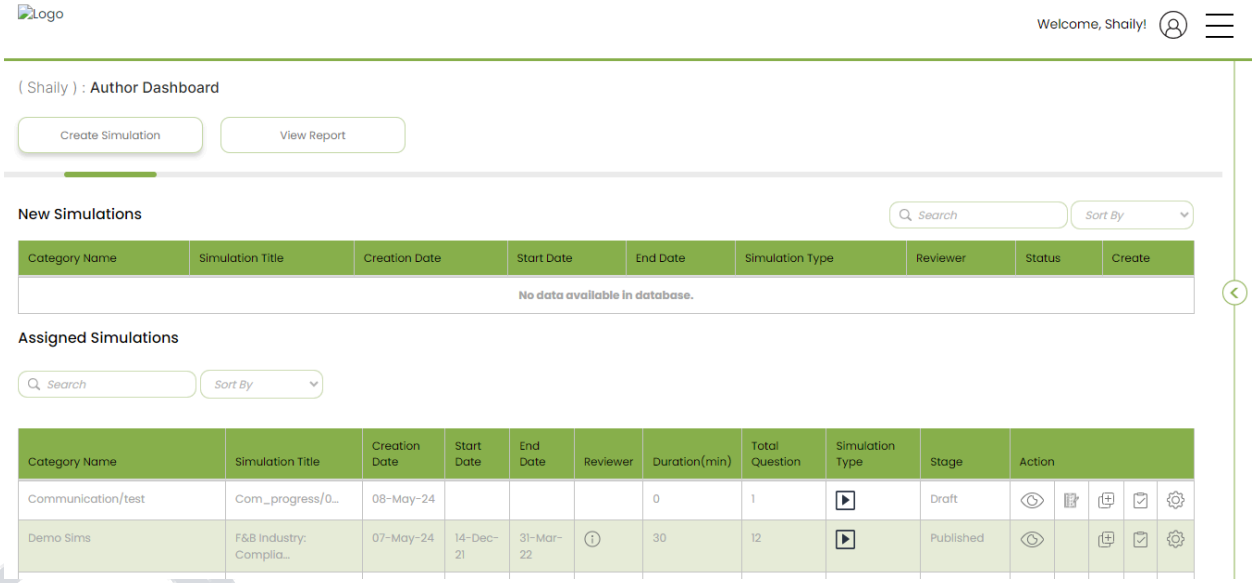
The Author dashboard have two tab
1. Create Simulation
2. View Report

1. Create Simulation
   Its have two simulation list
   a. New simulation list
      i. **New simulation list**: When the admin assigns any simulation to the author, the new simulation will appear in this list. The author can edit it.
      ii. After Editing, go to this page. The author can choose any one Simulation Type, add a Cover Image, edit the Simulation Name, and then create the simulation.
   b. Assigned simulation:-All created simulation is shown in this table. The author can Edit Simulation, Preview Simulation, Duplicate Simulation, Review Simulation, and Setting Simulation.
2. View Report:- It has all simulation reports, such as the number of simulation updates, the total number of simulations, publishing review simulations, assigned learners, etc.

Code Stuff

Author Dashboard

Template Name: author-overview-dashboard.php



PHP Functions Used

1. **getRole()**
   Global function from "db.class.php". This PHP function getRole() is designed to retrieve roles from a database table named "role" and return them as an associative array.
2. **getAssigneeName($userid)**
   Global function from "db.class.php". This function will return all the data for user as per the User ID passed in the argument.
3. **SQL Queries**
   $nsqlr = "SELECT DISTINCT MD5(s.scenario_id) AS en_sid, s.Scenario_title, LEFT(s.Scenario_title, 25) AS title, s.assign_start_date, s.assign_end_date, DATE_FORMAT(s.assign_start_date, '%d-%b-%y') AS assign_sdate, DATE_FORMAT(s.assign_end_date, '%d-%b-%y') AS assign_edate, s.assign_reviewer, s.scenario_type, s.scenario_category, DATE_FORMAT(s.created_on, '%d-%b-%y') AS created, c.category, st.status_name FROM scenario_master s INNER JOIN group_tbl g LEFT JOIN category_tbl c ON s.scenario_category = c.cat_id LEFT JOIN scenario_status st ON s.status = st.sid WHERE s.author_scenario_type = 0 AND s.sim_ques_type = 0 AND (CASE WHEN s.assign_author IS NOT NULL THEN FIND_IN_SET($userid, s.assign_author) WHEN s.assign_group IS NOT NULL THEN FIND_IN_SET($userid, g.author) END) $norderby";
   This query retrieves distinct scenario records from the scenario_master table, joined with group_tbl, category_tbl, and scenario_status tables, based on certain conditions. The results include various formatted dates, scenario details, and category and status

names. The final ordering of the results is determined by the $norderby variable.

$sqlr = "SELECT DISTINCT s.scenario_id, s.Scenario_title, LEFT(s.Scenario_title, 25) AS title, s.assign_start_date, s.assign_end_date, DATE_FORMAT(s.assign_start_date, '%d-%b-%y') AS assign_sdate, DATE_FORMAT(s.assign_end_date, '%d-%b-%y') AS assign_edate, s.assign_reviewer, s.scenario_type, s.scenario_category, s.duration, s.status, DATE_FORMAT(s.created_on, '%d-%b-%y') AS created, s.sim_ques_type, s.sim_temp_type, s.update_sim, s.vtype, s.web_object, s.web_object_file, c.category, st.status_name, t.type_name, t.type_icon FROM scenario_master s INNER JOIN group_tbl g LEFT JOIN category_tbl c ON s.scenario_category = c.cat_id LEFT JOIN scenario_status st ON s.status = st.sid LEFT JOIN scenario_type t ON s.author_scenario_type = t.st_id WHERE s.author_scenario_type != 0 AND (CASE WHEN s.assign_author IS NOT NULL THEN FIND_IN_SET($userid, s.assign_author) WHEN s.assign_group IS NOT NULL THEN FIND_IN_SET($userid, g.author) END) $orderby";

This query retrieves distinct scenario records from the scenario_master table, joined with group_tbl, category_tbl, scenario_status, and scenario_type tables, based on certain conditions. The results include various formatted dates, scenario details, category and status names, and type information. The final ordering of the results is determined by the $orderby variable.

4. **isAuthor()**
   Global function from “db.class.php”. The provided PHP function isAuthor checks if the current user's role is authorized to access a page. If not, it displays an alert and redirects the user to a different page.

Third Party JS Library

1. TooltipJS
2. CanvasJS

JS File Used

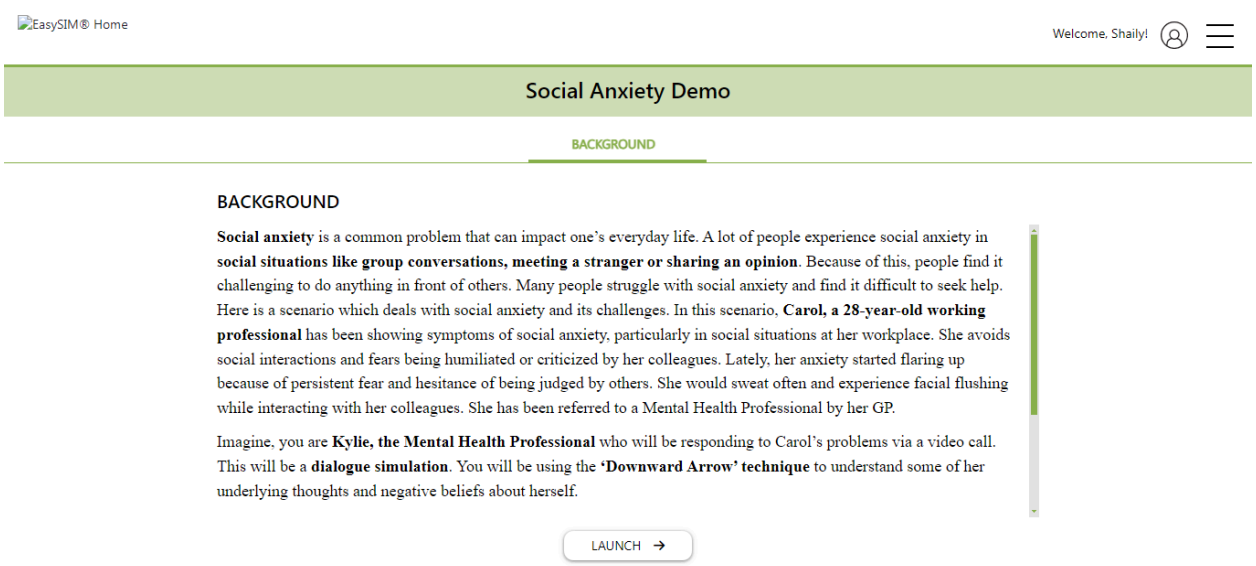
File Path: content/js/author-overview-dash.js

1. **getPercentage = (x, y, html = false) => {}**
   This JavaScript code defines a function named "getPercentage" that calculates the percentage of x relative to y, optionally returning the result as a formatted string with a percentage sign.
2. **CircleBar = () => {}**
   This JavaScript code defines a function named "CircleBar" that dynamically updates the style of elements representing circular progress bars on a webpage.
3. **getUrlVars = () => {}**
   The "getUrlVars" function extracts query parameters from the current URL and returns them as an array where each parameter is both an item and a property of the array.

4. **overview = () => {}**
   The overview function is triggered to fetch data asynchronously from a server-side endpoint at *'includes/author/overview-dashboard-data.php'*. Upon a successful response, it updates the UI with the received data. If the request fails, it displays an error message using SweetAlert.
5. **report = () => {}**
   The report function is triggered to fetch data asynchronously from a server-side endpoint at *'includes/author/overview-dashboard-data.php'*. Upon a successful response, it updates various elements of the UI with the received data, including HTML content and chart visualizations.
6. **$('.update_stage').click(function () {})**
   On click of element with the class update_stage, this code retrieves the associated scenario and status IDs, initiates a loading overlay, and then loads *'edit-scenario-stage-modal.php'* into a modal dialog with the details of the scenario's stage.
7. **$('.clone_sim').click(function () {})**
   On click of button with class clone_sim this code prompts the user to confirm if they want to clone the simulation. If confirmed, it sends an AJAX request to *"includes/process.php"* to the server to perform the cloning operation. Depending on the response from the server, it displays success or error messages using SweetAlert.

Author > Preview

Template Name: preview.php



PHP Functions Used

1. **checkLogin()**
   Global function from "db.class.php". The checkLogin function ensures that a user is properly logged in by verifying that all essential session variables are set and not empty.

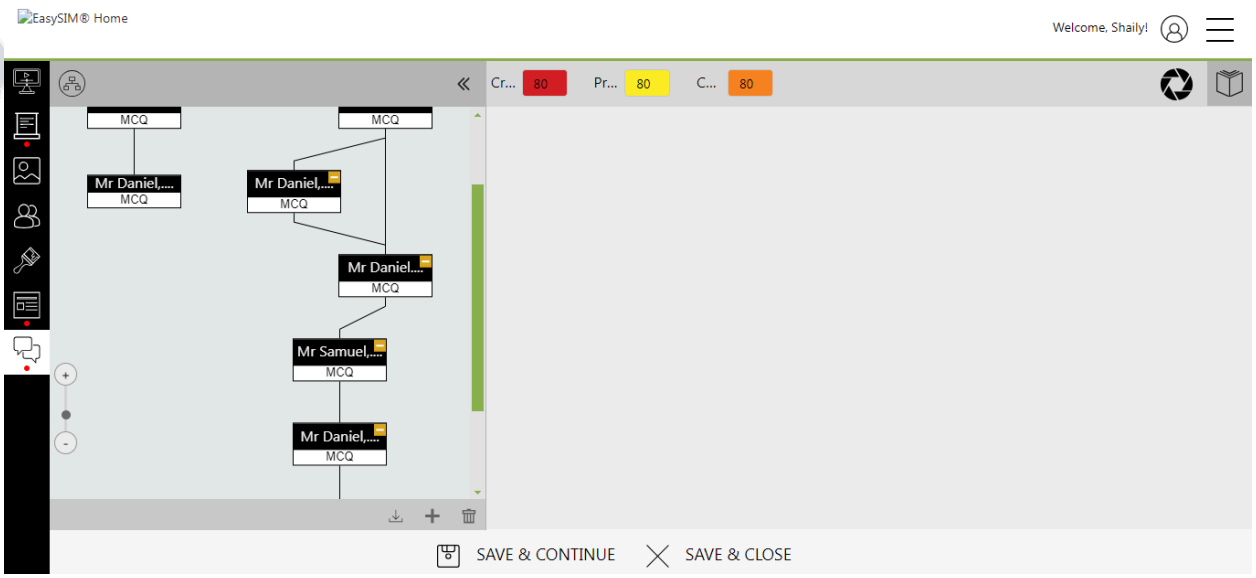
2. **getScenario($sid)**
   Global function from “db.class.php”. This PHP function, "getScenario", retrieves scenario data from the database based on the provided scenario ID ($sim_id).
3. **getSimPage($sim_id)**
   Global function from “db.class.php”. This PHP function, "getSimPage", retrieves simulation pages associated with a specific scenario ID from the sim_pages table in the database.
4. **checkProctorPermission($client_id)**
   Global function from “db.class.php”. This function queries the database to check if a client has proctor permissions based on their client_id. It returns TRUE if proctor permissions are granted and FALSE otherwise.

Third Party JS Library

1. OwlCarousel
2. GreenAudioPlayer

Edit Simulation > Branching Classic

Template Name: branching-classic-template.php



PHP Functions Used

1. **checkLogin()**
   Global function from “db.class.php”. The checkLogin function ensures that a user is properly logged in by verifying that all essential session variables are set and not empty.

2. **getScenario($sim_id)**
   Global function from “db.class.php”. This PHP function, "getScenario", retrieves scenario data from the database based on the provided scenario ID ($sim_id).
3. **getScenarioCompetency($sid)**
   Global function from “db.class.php”. This function, "getScenarioCompetency", retrieves competency information associated with a specific scenario from the competency_tbl database table.
4. **getSimPage($sid)**
   Global function from “db.class.php”. This PHP function, "getSimPage", retrieves simulation pages associated with a specific scenario ID from the sim_pages table in the database.
5. **getSimBranding($sid)**
   Global function from “db.class.php”. This function, “getSimBranding”, retrieves branding information associated with a specific scenario from the sim_branding_tbl database table.
6. **getMultiScenarioType($id)**
   Global function from “db.class.php”. The getMultiScenarioType function retrieves data from a database based on the provided scenario type IDs.
7. **SQL Queries**
   *$tqsql = "SELECT *, LEFT(questions, 8) AS qtitle FROM question_tbl WHERE scenario_id = ?";*
   The query retrieves all columns from the question_tbl table. It also extracts the first 8 characters from the questions column and assigns this substring to a new column named qtitle in the result set. The query includes a condition to filter the results based on a specific scenario_id value, which is supplied as a parameter when the query is executed.

   $feedSql = "SELECT question_id, nextQid FROM answer_tbl WHERE question_id = ?";
   This query selects specific data from the answer_tbl table in a database based on the question_id passed.

   $questionListSql = "SELECT question_id, questions AS ques FROM question_tbl WHERE scenario_id = ? AND question_id != ?";
   This SQL query selects specific data from the question_tbl table in a database based on the scenario_id and question_id passed into it.

   $mcq_sql = "SELECT a., f.* FROM answer_tbl a LEFT JOIN feedback_tbl f ON f.answer_id = a.answer_id WHERE a.question_id = ?";*
   This SQL query string designed to select data from two tables, answer_tbl and feedback_tbl, using a LEFT JOIN as per the question_id passed in WHERE clause.

   *$ddq2_sql = "SELECT * FROM sub_options_tbl WHERE question_id = ?";*
   This SQL query selects data from the sub_options_tbl table based on a specific question_id.

$ddqop_sql = "SELECT a., f.* FROM answer_tbl a LEFT JOIN feedback_tbl f ON a.answer_id = f.answer_id WHERE a.question_id = ?";*
This SQL query selects data from two tables, answer_tbl and feedback_tbl, using a LEFT JOIN. This query retrieves all answers for a specific question along with any associated feedback.

$mappedListSql = "SELECT End_Sim, nextQid FROM answer_tbl WHERE question_id = ? AND answer_id = ?";
This SQL query selects specific data from the answer_tbl table based on given question_id and answer_id values.

$swa_sql = "SELECT a., f.* FROM answer_tbl a LEFT JOIN feedback_tbl f ON f.answer_id = a.answer_id WHERE a.question_id = ?";*
This SQL query retrieves data from two tables, answer_tbl and feedback_tbl, utilizing a LEFT JOIN operation. This query aims to retrieve information about answers to a particular question along with any associated feedback.

8. **truncateText($text, $chars)**
   Global function from "db.class.php". The function shortens the text to the specified character limit (20 characters) without cutting off words in the middle and adds ellipses to indicate that the text has been truncated.

Third Party JS Library

1. InputMask
2. Spectrum
3. JqueryUI
4. GoJS
5. HyperLinkText
6. ZoomSlider
7. Polyfill
8. JSPdf
9. HTML2Canvas

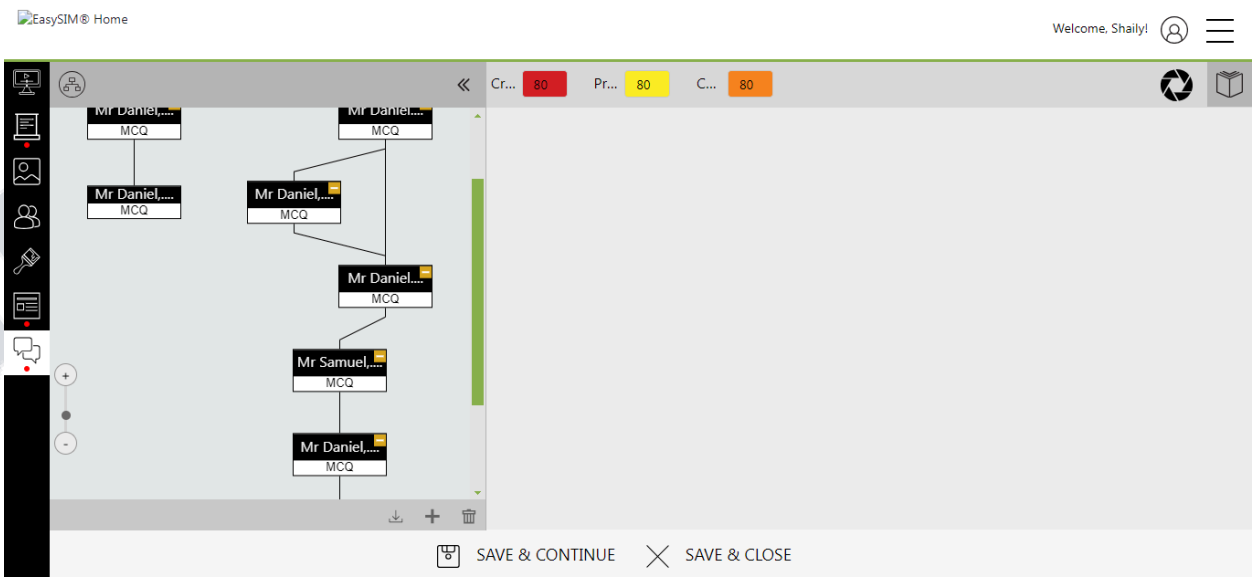
JS Functions Used

1. **getPDF(id) {}**
   This JavaScript function, getPDF(id), converts an HTML element specified by its ID into a PDF document.
2. **$('.qtype').on('click', function(){})**
   Clicking on any element with the class qtype would set the value of the hidden input field with the ID question_type to the value of its data-qtype attribute. The get_question_type input field can be used to retain the original question type value if needed.

3. **$(".MCQadd").click(function(){})**
   This function adds a new MCQ form dynamically when a specific element is clicked, and it also initializes jQuery Text Editors on the text areas within the newly added MCQ form.
4. **$(".Add_Drag").click(function(){})**
   This JavaScript code is part of a application where users can create questions with various features like multiple-choice questions (MCQs) and drag-and-drop (DND) questions.
5. **$(".swipAdd").click(function(){})**
   This code allows users to dynamically add form elements and upload files, with additional functionality for rich text editing and text-to-speech conversion.

Edit Simulation > Branching Multiple

Template Name: branching-multiple-template.php



PHP Functions Used

1. **SQL Queries**
   $questionListSql = "SELECT question_id, questions AS ques FROM question_tbl WHERE scenario_id = ? AND question_id != ?";
   This SQL query selects question_id and questions (aliased as ques) from a table named question_tbl, filtering the rows based on specific conditions related to scenario_id and question_id.

   $mappedCorrectListSql = "SELECT nextQid FROM feedback_tbl WHERE question_id = ? AND feed_id = ?";
   This SQL query selects the nextQid column from a table named feedback_tbl, filtering the rows based on specific conditions related to question_id and feed_id.

$mtf_choice_sql = "SELECT answer_id, choice_option, choice_order_no, match_order_no, match_option FROM answer_tbl WHERE question_id = ? AND choice_order_no != '0' AND match_order_no != '0'";
This SQL query selects specific columns from a table named answer_tbl, filtering the rows based on conditions related to question_id, choice_order_no, and match_order_no.

$mtf_choice_sql1 = "SELECT answer_id, choice_option, match_option FROM answer_tbl WHERE question_id = ? AND choice_order_no = '0' AND match_order_no = '0'";
This SQL query selects specific columns from a table named answer_tbl, filtering the rows based on conditions related to question_id, choice_order_no, and match_order_no. This query seems to target rows where both choice_order_no and match_order_no are equal to zero.

$sort_sql = "SELECT answer_id, choice_option, match_option, match_sorting_item, image FROM answer_tbl WHERE question_id = ?";
This SQL query selects specific columns from a table named answer_tbl, filtering the rows based on the question_id column. This query retrieves data related to answers for a specific question identified by its question_id.

$sortop_sql = "SELECT sub_options_id, answer_id, sub_option FROM sub_options_tbl WHERE answer_id = ?";
This SQL query retrieves sub-options related to a specific answer identified by its answer_id. The sub-options are stored in the sub_options_tbl table.

$ddq_sql = "SELECT answer_id, choice_option, match_option, image, image2 FROM answer_tbl WHERE question_id = ?";
This SQL query retrieves answers related to a specific question identified by its question_id. The answers' details include the option choices, match options, and potentially images associated with those answers.

*$ddq2_sql = "SELECT * FROM sub_options_tbl WHERE question_id = ?";*
This SQL query retrieves all data from the sub_options_tbl table where the question_id matches a specified value. It's used to fetch sub-options associated with a particular question identified by its question_id.

*$ddqop_sql = "SELECT * FROM answer_tbl WHERE question_id = ?";*
This SQL query retrieves all data from the answer_tbl table where the question_id matches a specified value. It's used to fetch answers associated with a particular question identified by its question_id.

$mcq_sql = "SELECT answer_id, choice_option FROM answer_tbl WHERE question_id = ?";

This SQL query retrieves the answer_id and choice_option columns from the answer_tbl table for rows where the question_id matches a specified value. It's used to fetch specific information about answers associated with a particular question identified by its question_id.

$swa_sql = "SELECT answer_id, choice_option, image FROM answer_tbl WHERE question_id = ?";
This SQL query retrieves the answer_id, choice_option, and image columns from the answer_tbl table for rows where the question_id matches a specified value. It's used to fetch specific information about answers associated with a particular question identified by its question_id, including any images linked to those answers.

Third Party JS Library

1. Polyfill
2. JSPdf
3. HTML2Canvas

JS Functions Used

1. **getPDF(id) {}**
   This JavaScript function, getPDF(id), converts an HTML element specified by its ID into a PDF document.
2. **$('.qtype').on('click', function(){})**
   This code dynamically sets the required attribute for certain elements based on the clicked question type and potentially updates the question type value in the UI based on certain conditions. It seems to be part of a broader functionality related to managing question types and form validation.
3. **$('.mtf_shuffle').on('click', function(){})**
   This code seems is a part of a functionality related to managing shuffling options for a Match The Following type of question in a form. Depending on the value of the clicked element, it toggles the required attribute for certain elements accordingly, presumably to enforce form validation rules.
4. **$(".AddAnsICON").click(function(){})**
   This code dynamically adds new MTF options to a form when a certain button (AddAnsICON) is clicked. It ensures that the newly added options are properly initialized, including form validation and rich text editing functionality.
5. **$(".AddAnsICON1").click(function(){})**
   This code dynamically adds new MTF options to a form when a certain button (AddAnsICON1) is clicked. It ensures that the newly added options are properly initialized, including form validation and rich text editing functionality, similar to the previous snippet. However, these options might have different styling or behavior, as indicated by the different class used for their containers.
6. **$(".DNDAddAnsICON1").click(function(){})**
   This code dynamically adds new options for a Drag and Drop question type to a form

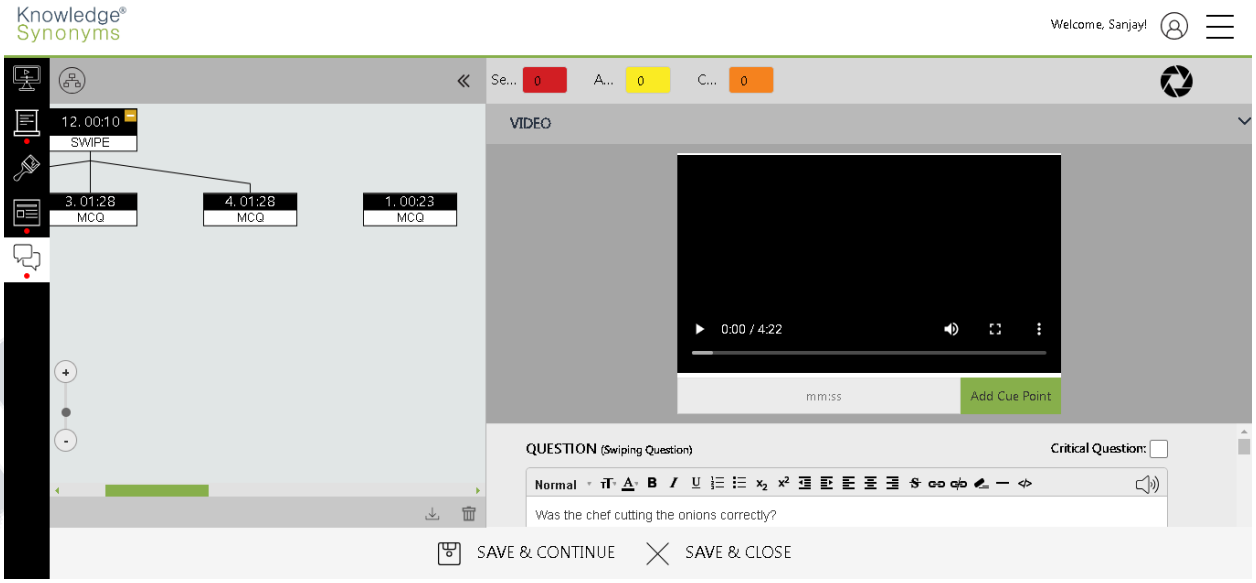
when a certain button (DNDAddAnsICON1) is clicked. It ensures that the newly added options are properly initialized, including rich text editing functionality.

7. **$(".DNDAddAnsICON").click(function(){})**
   This code dynamically adds new drag and drop options to a form when a certain button (DNDAddAnsICON) is clicked. It ensures that the newly added options are properly initialized, including rich text editing functionality.
8. **$(".SeqAdd").click(function(){})**
   This code dynamically adds new sequence answer options to a form when a certain button (SeqAdd) is clicked. It ensures that the newly added options are properly initialized, including rich text editing functionality.
9. **$(".ShortAddICON").click(function(){})**
   This code dynamically adds new options for a short answer question with drag-and-drop functionality when a certain button (ShortAddICON) is clicked. It ensures that the newly added options are properly initialized, including rich text editing functionality, and provides functionality to add and remove drag options dynamically.
10. **$('body').on('click', '.ShortDragAddICON', function() {})**
    This code dynamically adds new drag options within a specific sorting item for a short answer question when a certain button (ShortDragAddICON) is clicked. It ensures that the new drag option is properly initialized with rich text editing functionality and provides functionality to remove the drag option if needed.
11. **$(".MCQAdd").click(function(){})**
    This code dynamically adds new MCQ options when a certain button (MCQAdd) is clicked. It ensures that each new MCQ option is properly initialized with a text editor for text input and provides functionality to remove the MCQ option if needed.
12. **$(".MMCQAdd").click(function(){})**
    This function dynamically adds a new set of checkbox and textarea elements for multiple choice questions (MMCQ) each time the .MMCQAdd element is clicked, and initializes a rich text editor for the textarea element.
13. **$(".SwaAdd").click(function(){})**
    This function dynamically adds a new set of radio button, textarea, and image upload elements for single word answer questions (SWA) each time the .SwaAdd element is clicked, and initializes a rich text editor for the textarea element.
14. **$('body').on('click', '.removeOption', function(e) {})**
    This code provides a user-friendly way to delete options from a list and handles the deletion process asynchronously without refreshing the page. Ajax request is sent to the server using the $.ajax method. The request is sent to the URL specified (*includes/process.php*) with the data string containing the information about the answer to be deleted.
15. **$('body').on('click', '.removeSortingOption', function(e) {})**
    This code provides a user-friendly way to delete sorting options and handles the deletion process asynchronously without refreshing the page. Ajax request is sent to the server using the $.ajax method. The request is sent to the URL specified (*includes/process.php*) with the data string containing the information about the sorting answer to be deleted.

16. **$('body').on('click', '.removeSortingSubOption', function(e) {})**
    This code provides a user-friendly way to delete sorting sub-options and handles the deletion process asynchronously without refreshing the page. Ajax request is sent to the server using the $.ajax method. The request is sent to the URL specified (*includes/process.php*) with the data string containing the information about the sorting sub-answer to be deleted.

Edit Simulation > Branching Video Classic

Template Name: branching-video-classic-template.php



PHP Functions Used

1. **checkLogin()**
   Global function from "db.class.php". The checkLogin function ensures that a user is properly logged in by verifying that all essential session variables are set and not empty.
2. **getScenario($sim_id)**
   Global function from "db.class.php". This PHP function, "getScenario", retrieves scenario data from the database based on the provided scenario ID ($sim_id).
3. **getScenarioCompetency($sid)**
   Global function from "db.class.php". This function, "getScenarioCompetency", retrieves competency information associated with a specific scenario from the competency_tbl database table.
4. **getSimPage($sid)**
   Global function from "db.class.php". This PHP function, "getSimPage", retrieves simulation pages associated with a specific scenario ID from the sim_pages table in the database.
5. **getSimBranding($sid)**
   Global function from "db.class.php". This function, "getSimBranding", retrieves branding

information associated with a specific scenario from the sim_branding_tbl database table.

6. **getMultiScenarioType($id)**
   Global function from "db.class.php". The getMultiScenarioType function retrieves data from a database based on the provided scenario type IDs.
7. **SQL Queries**
   *$qsql = "SELECT * FROM question_tbl WHERE question_id = ?";*
   This query retrieves all columns from the question_tbl table for a specific question_id.

   *$tqsql = "SELECT *, LEFT(questions, 8) AS qtitle FROM question_tbl WHERE scenario_id = ?";*
   The query retrieves all columns from the question_tbl table. It also extracts the first 8 characters from the questions column and assigns this substring to a new column named qtitle in the result set. The query includes a condition to filter the results based on a specific scenario_id value, which is supplied as a parameter when the query is executed.

   $feedSql = "SELECT question_id, nextQid FROM answer_tbl WHERE question_id = ?";
   This query selects specific data from the answer_tbl table in a database based on the question_id passed.

   $questionListSql = "SELECT question_id, questions, videoq_cue_point FROM question_tbl WHERE scenario_id = ? AND question_id != ?";
   This query retrieves question_id, questions, and videoq_cue_point columns from the question_tbl table where the scenario_id matches a specified value and the question_id does not match another specified value.

   $mcq_sql = "SELECT a., f.* FROM answer_tbl a LEFT JOIN feedback_tbl f ON f.answer_id = a.answer_id WHERE a.question_id = ?";*
   This SQL query string designed to select data from two tables, answer_tbl and feedback_tbl, using a LEFT JOIN as per the question_id passed in WHERE clause.

   *$ddq2_sql = "SELECT * FROM sub_options_tbl WHERE question_id = ?";*
   This SQL query selects data from the sub_options_tbl table based on a specific question_id.

   $ddqop_sql = "SELECT a., f.* FROM answer_tbl a LEFT JOIN feedback_tbl f ON a.answer_id = f.answer_id WHERE a.question_id = ?";*
   This SQL query selects data from two tables, answer_tbl and feedback_tbl, using a LEFT JOIN. This query retrieves all answers for a specific question along with any associated feedback.

   $mappedListSql = "SELECT End_Sim, nextQid FROM answer_tbl WHERE question_id = ? AND answer_id = ?";

This SQL query selects specific data from the answer_tbl table based on given question_id and answer_id values.

$swa_sql = "SELECT a., f.* FROM answer_tbl a LEFT JOIN feedback_tbl f ON f.answer_id = a.answer_id WHERE a.question_id = ?";*
This SQL query retrieves data from two tables, answer_tbl and feedback_tbl, utilizing a LEFT JOIN operation. This query aims to retrieve information about answers to a particular question along with any associated feedback.

Third Party JS Library

1. Polyfill
2. JSPdf
3. HTML2Canvas

JS Functions Used

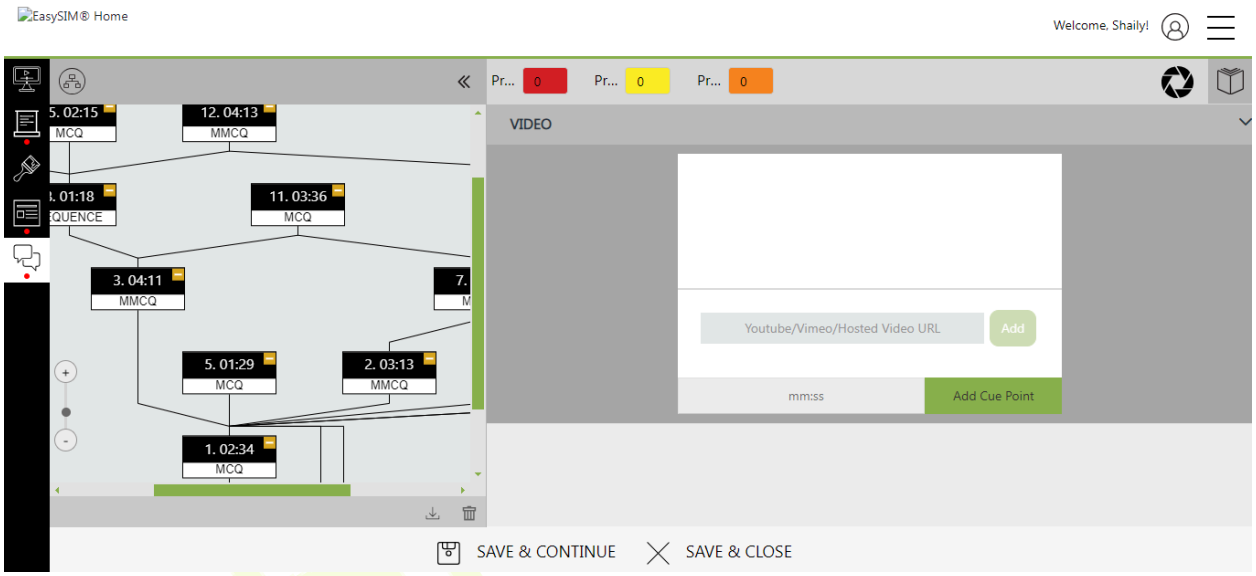
1. **getPDF(id) {}**
   This JavaScript function, getPDF(id), converts an HTML element specified by its ID into a PDF document.
2. **$('.qtype').on('click', function(){})**
   This code dynamically sets the required attribute for certain elements based on the clicked question type and potentially updates the question type value in the UI based on certain conditions. It seems to be part of a broader functionality related to managing question types and form validation.
3. **$(".Add_Drag").click(function(){})**
   This JavaScript code is part of a application where users can create questions with various features like multiple-choice questions (MCQs) and drag-and-drop (DND) questions.
4. **$('body').on('click', '.endSimChkDnD', function() {})**
   This code dynamically controls the display of elements based on the state of checkboxes. If a checkbox is checked, it hides certain elements, and if it's unchecked, it shows them. This is done for each checkbox with the class .endSimChkDnD.
5. **$(".MCQAdd").click(function(){})**
   This code dynamically adds new MCQ options when a certain button (MCQAdd) is clicked. It ensures that each new MCQ option is properly initialized with a text editor for text input and provides functionality to remove the MCQ option if needed.
6. **$(".swipAdd").click(function(){})**
   This code allows users to dynamically add form elements and upload files, with additional functionality for rich text editing and text-to-speech conversion.
7. **$('#upload_videoq_file').click(function() {})**
   This function handles the upload of a video file asynchronously without reloading the entire page, providing a smooth user experience. It also dynamically updates the UI

based on the upload result. It sends an AJAX request to a PHP script (*upload.php*) with the form data.

8. **$('body').on('click', '.delete_video_assets', function() {})**
   This function provides a smooth way for users to delete video assets and provides feedback on the success or failure of the deletion process. It sends an AJAX request to a PHP script (*delete-file.php*) with the data string.. It also utilizes a confirmation dialog to ensure users are aware of the action they are taking.
9. **$('#cloud_url').on('keyup', function(){})**
   This function seems to validate a video URL entered by the user, ensuring it matches a specified type (YouTube, Vimeo, or hosted video), enabling/disabling certain UI elements accordingly, and providing feedback to the user through warning messages.
10. **$('#cloud_multi_url').on('keyup', function(){})**
    This function appears to validate multiple video URLs entered by the user, ensuring they match a specified type (YouTube, Vimeo, or hosted video), enabling/disabling certain UI elements accordingly, and providing feedback to the user through warning messages.
11. **$('#add_cloud_video').on('click', function(){})**
    This function is responsible for embedding a cloud-hosted video (from YouTube, Vimeo, or a direct URL) into the webpage when a user clicks a specific button, providing a visual representation of the video content.
12. **$('#add_cloud_multi_video').on('click', function(){})**
    This function embeds a cloud-hosted video (from YouTube, Vimeo, or a direct URL) into the webpage and displays additional content related to the video when a user clicks a specific button. It also stores the URL of the video for future reference.
13. **$('body').on('click', '.delete_cloudv_assets', function() {})**
    This function provides a mechanism for users to delete cloud-hosted video assets, along with associated actions such as clearing the embedded video, resetting input fields, and adjusting the visibility and status of other elements accordingly.
14. **$('body').on('click', '.delete_cloud_multiv_assets', function() {})**
    This function provides a way for users to delete multiple cloud-hosted video assets, along with associated actions such as clearing the embedded video, resetting input fields, and adjusting the visibility and status of other elements accordingly.
15. **$('body').on('click', '.cloud_add_cue', function(){})**
    Ihis function allows users to add cue points to a cloud-hosted video by sending the necessary data to a server-side script for processing. It sends an AJAX POST request to a PHP script (*process.php*) with the constructed data. It provides feedback to the user based on the success or failure of the operation and updates the page accordingly.

Edit Simulation > Branching Video Multiple

Template Name: branching-video-multiple-template.php



PHP Functions Used

1. **checkLogin()**
   Global function from "db.class.php". The checkLogin function ensures that a user is properly logged in by verifying that all essential session variables are set and not empty.
2. **getScenario($sim_id)**
   Global function from "db.class.php". This PHP function, "getScenario", retrieves scenario data from the database based on the provided scenario ID ($sim_id).
3. **getScenarioCompetency($sid)**
   Global function from "db.class.php". This function, "getScenarioCompetency", retrieves competency information associated with a specific scenario from the competency_tbl database table.
4. **getSimPage($sid)**
   Global function from "db.class.php". This PHP function, "getSimPage", retrieves simulation pages associated with a specific scenario ID from the sim_pages table in the database.
5. **getSimBranding($sid)**
   Global function from "db.class.php". This function, "getSimBranding", retrieves branding information associated with a specific scenario from the sim_branding_tbl database table.
6. **getMultiScenarioType($id)**
   Global function from "db.class.php". The getMultiScenarioType function retrieves data from a database based on the provided scenario type IDs.
7. **SQL Queries**
   *$qsql = "SELECT * FROM question_tbl WHERE question_id = ?";*
   This query retrieves all columns from the question_tbl table for a specific question_id.

   *$feedsql = "SELECT * FROM feedback_tbl WHERE question_id = :question_id";*
   This SQL query retrieves all columns from the feedback_tbl table where the question_id

column matches a specific value.

*$tqsql = "SELECT *, LEFT(questions, 8) AS qtitle FROM question_tbl WHERE scenario_id = ?";*
The query retrieves all columns from the question_tbl table. It also extracts the first 8 characters from the questions column and assigns this substring to a new column named qtitle in the result set. The query includes a condition to filter the results based on a specific scenario_id value, which is supplied as a parameter when the query is executed.

$feedSql = "SELECT question_id, nextQid FROM feedback_tbl WHERE question_id = ?";
This SQL query retrieves the question_id and nextQid columns from the feedback_tbl table where the question_id matches a specific value.

$questionListSql = "SELECT question_id, questions, videoq_cue_point FROM question_tbl WHERE scenario_id = ? AND question_id != ?";
This query retrieves question_id, questions, and videoq_cue_point columns from the question_tbl table where the scenario_id matches a specified value and the question_id does not match another specified value.

$mappedCorrectListSql = "SELECT nextQid FROM feedback_tbl WHERE question_id = ? AND feed_id = ?";
This SQL query selects the nextQid column from a table named feedback_tbl, filtering the rows based on specific conditions related to question_id and feed_id.

$mtf_choice_sql = "SELECT answer_id, choice_option, choice_order_no, match_order_no, match_option FROM answer_tbl WHERE question_id = ? AND choice_order_no != '0' AND match_order_no != '0'";
This SQL query selects specific columns from a table named answer_tbl, filtering the rows based on conditions related to question_id, choice_order_no, and match_order_no.

$mtf_choice_sql1 = "SELECT answer_id, choice_option, match_option FROM answer_tbl WHERE question_id = ? AND choice_order_no = '0' AND match_order_no = '0'";
This SQL query selects specific columns from a table named answer_tbl, filtering the rows based on conditions related to question_id, choice_order_no, and match_order_no. This query seems to target rows where both choice_order_no and match_order_no are equal to zero.

$sq_sql = "SELECT answer_id, choice_option FROM answer_tbl WHERE question_id = ?";
This SQL query retrieves the answer_id and choice_option columns from the answer_tbl table where the question_id matches a specific value.

$sort_sql = "SELECT answer_id, choice_option, match_option, match_sorting_item, image FROM answer_tbl WHERE question_id = ?";
This SQL query selects specific columns from a table named answer_tbl, filtering the rows based on the question_id column. This query retrieves data related to answers for a specific question identified by its question_id.

$sortop_sql = "SELECT sub_options_id, answer_id, sub_option FROM sub_options_tbl WHERE answer_id = ?";
This SQL query retrieves sub-options related to a specific answer identified by its answer_id. The sub-options are stored in the sub_options_tbl table.

$ddq_sql = "SELECT answer_id, choice_option, match_option, image, image2 FROM answer_tbl WHERE question_id = ?";
This SQL query retrieves answers related to a specific question identified by its question_id. The answers' details include the option choices, match options, and potentially images associated with those answers.

*$ddq2_sql = "SELECT * FROM sub_options_tbl WHERE question_id = ?";*
This SQL query retrieves all data from the sub_options_tbl table where the question_id matches a specified value. It's used to fetch sub-options associated with a particular question identified by its question_id.

*$ddqop_sql = "SELECT * FROM answer_tbl WHERE question_id = ?";*
This SQL query retrieves all data from the answer_tbl table where the question_id matches a specified value. It's used to fetch answers associated with a particular question identified by its question_id.

$mcq_sql = "SELECT answer_id, choice_option FROM answer_tbl WHERE question_id = ?";
This SQL query retrieves the answer_id and choice_option columns from the answer_tbl table for rows where the question_id matches a specified value. It's used to fetch specific information about answers associated with a particular question identified by its question_id.

$mmcq_sql = "SELECT answer_id, choice_option FROM answer_tbl WHERE question_id = ?";
The query retrieves the answer_id and choice_option columns from the answer_tbl table for rows where the question_id matches a specific value.

$swa_sql = "SELECT answer_id, choice_option, image FROM answer_tbl WHERE question_id = ?";
This SQL query retrieves the answer_id, choice_option, and image columns from the answer_tbl table for rows where the question_id matches a specified value. It's used to

fetch specific information about answers associated with a particular question identified by its question_id, including any images linked to those answers.

Third Party JS Library

1. Polyfill
2. JSPdf
3. HTML2Canvas

JS Functions Used

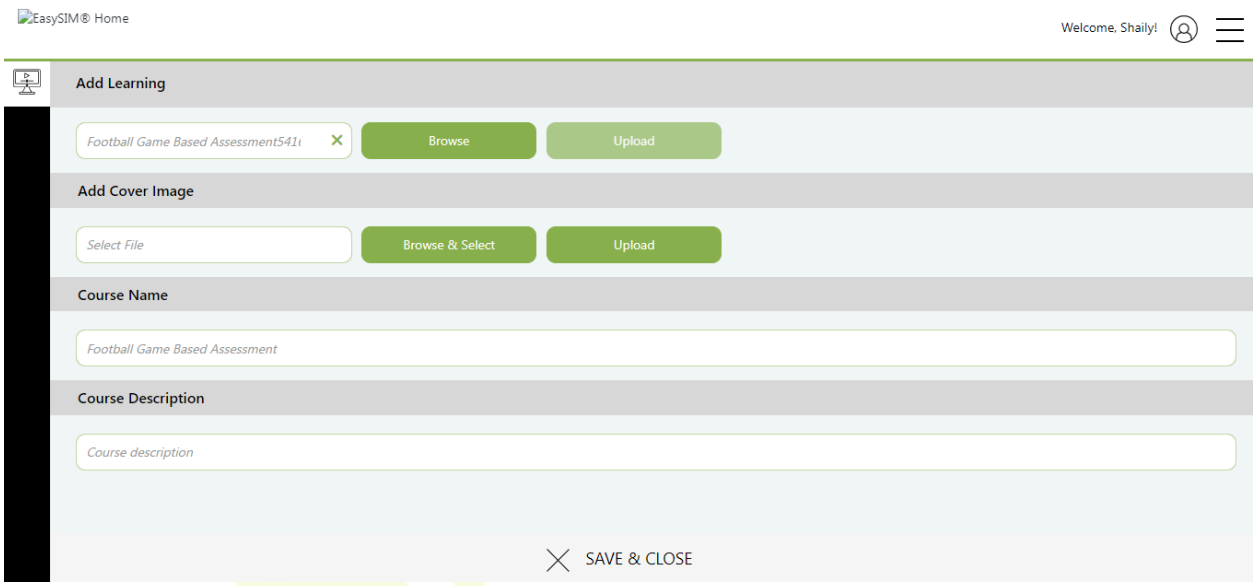
1. **getPDF(id) {}**
   This JavaScript function, getPDF(id), converts an HTML element specified by its ID into a PDF document.
2. **$('.qtype').on('click', function(){})**
   This code dynamically sets the required attribute for certain elements based on the clicked question type and potentially updates the question type value in the UI based on certain conditions. It seems to be part of a broader functionality related to managing question types and form validation.
3. **$(".AddAnsICON").click(function(){})**
   This code dynamically adds new MTF options to a form when a certain button (AddAnsICON) is clicked. It ensures that the newly added options are properly initialized, including form validation and rich text editing functionality.
4. **$(".AddAnsICON1").click(function(){})**
   This code dynamically adds new MTF options to a form when a certain button (AddAnsICON1) is clicked. It ensures that the newly added options are properly initialized, including form validation and rich text editing functionality, similar to the previous snippet. However, these options might have different styling or behavior, as indicated by the different class used for their containers.
5. **$(".DNDAddAnsICON1").click(function(){})**
   This code dynamically adds new options for a Drag and Drop question type to a form when a certain button (DNDAddAnsICON1) is clicked. It ensures that the newly added options are properly initialized, including rich text editing functionality.
6. **$(".DNDAddAnsICON").click(function(){})**
   This code dynamically adds new drag and drop options to a form when a certain button (DNDAddAnsICON) is clicked. It ensures that the newly added options are properly initialized, including rich text editing functionality.
7. **$(".SeqAdd").click(function(){})**
   This code dynamically adds new sequence answer options to a form when a certain button (SeqAdd) is clicked. It ensures that the newly added options are properly initialized, including rich text editing functionality.
8. **$(".ShortAddICON").click(function(){})**
   This code dynamically adds new options for a short answer question with drag-and-drop

functionality when a certain button (ShortAddICON) is clicked. It ensures that the newly added options are properly initialized, including rich text editing functionality, and provides functionality to add and remove drag options dynamically.

9. **$('body').on('click', '.ShortDragAddICON', function() {})**
   This code dynamically adds new drag options within a specific sorting item for a short answer question when a certain button (ShortDragAddICON) is clicked. It ensures that the new drag option is properly initialized with rich text editing functionality and provides functionality to remove the drag option if needed.
10. **$(".MCQAdd").click(function(){})**
    This code dynamically adds new MCQ options when a certain button (MCQAdd) is clicked. It ensures that each new MCQ option is properly initialized with a text editor for text input and provides functionality to remove the MCQ option if needed.
11. **$(".MMCQAdd").click(function(){})**
    This function dynamically adds a new set of checkbox and textarea elements for multiple choice questions (MMCQ) each time the .MMCQAdd element is clicked, and initializes a rich text editor for the textarea element.
12. **$(".SwaAdd").click(function(){})**
    This function dynamically adds a new set of radio button, textarea, and image upload elements for single word answer questions (SWA) each time the .SwaAdd element is clicked, and initializes a rich text editor for the textarea element.
13. **$('body').on('click', '.removeOption', function(e) {})**
    This code provides a user-friendly way to delete options from a list and handles the deletion process asynchronously without refreshing the page. Ajax request is sent to the server using the $.ajax method. The request is sent to the URL specified (*includes/process.php*) with the data string containing the information about the answer to be deleted.
14. **$('body').on('click', '.removeSortingOption', function(e) {})**
    This code provides a user-friendly way to delete sorting options and handles the deletion process asynchronously without refreshing the page. Ajax request is sent to the server using the $.ajax method. The request is sent to the URL specified (*includes/process.php*) with the data string containing the information about the sorting answer to be deleted.
15. **$('body').on('click', '.removeSortingSubOption', function(e) {})**
    This code provides a user-friendly way to delete sorting sub-options and handles the deletion process asynchronously without refreshing the page. Ajax request is sent to the server using the $.ajax method. The request is sent to the URL specified (*includes/process.php*) with the data string containing the information about the sorting sub-answer to be deleted.

Edit Simulation > e-Learning

Template Name: e-learning-template.php



PHP Functions Used

1. **checkLogin()**
   Global function from "db.class.php". The checkLogin function ensures that a user is properly logged in by verifying that all essential session variables are set and not empty.
2. **getScenario($sim_id)**
   Global function from "db.class.php". This PHP function, "getScenario", retrieves scenario data from the database based on the provided scenario ID ($sim_id).

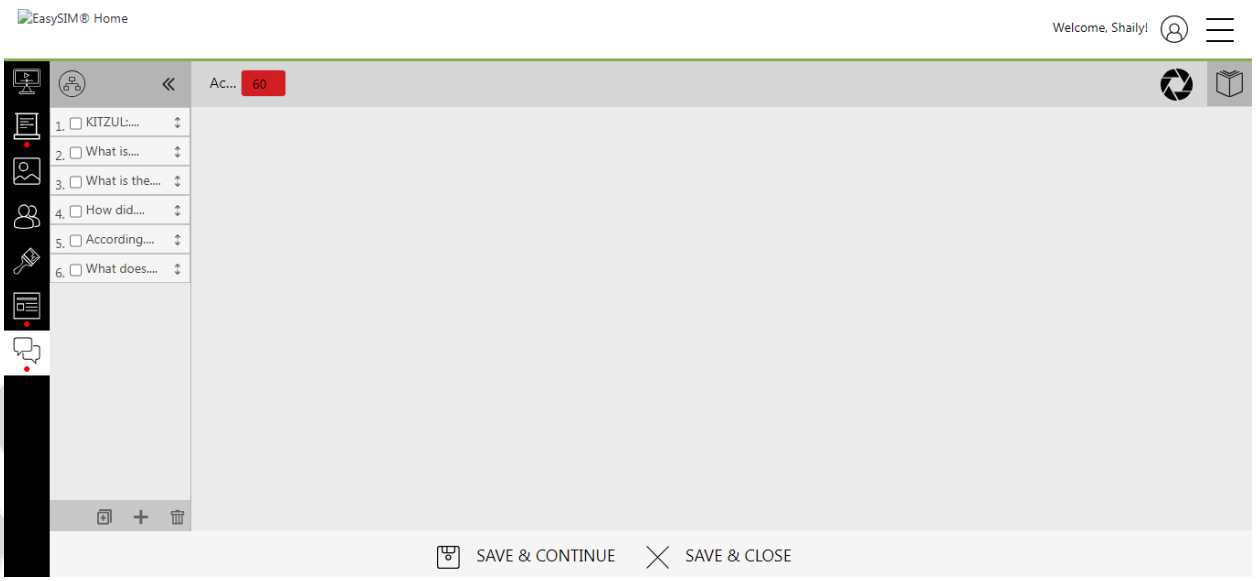
JS Functions Used

1. **checkFileType(event) {}**
   This function ensures that only files with specific extensions can be uploaded. If the file has an invalid extension, it alerts the user, clears the file input, and disables the upload button. If the file is valid, it enables the upload button.
2. **$('#upload_web').click(function(e) {})**
   This code handles the file upload process by disabling the upload button, sending the file via AJAX to *'includes/upload.php'*, and updating the UI based on the response from the server.
3. **$('.delete_wb').click(function(e) {})**
   This code handles the file deletion process by confirming with the user, sending an AJAX request to delete the file at *"includes/delete-file.php"*, and updating the UI based on the response from the server.
4. **$('#upload_cover_img').click(function(e) {})**
   This code handles the cover image upload process by validating the file type, sending the file via AJAX request to *'includes/upload.php'*, and updating the UI based on the server's response. If the file type is invalid or an error occurs, it displays appropriate messages and restores the button states.

5. **$('.delete_cover_img_file').click(function(e) {})**
   This code handles the cover image file deletion process by confirming with the user, sending an AJAX request at *"includes/delete-file.php"* to delete the file, and updating the UI based on the server's response. If an error occurs, it displays appropriate messages and restores the button states.

Edit Simulation > Hybrid Linear Classic

Template Name: hybrid-linear-classic-template.php



PHP Functions Used

1. **checkLogin()**
   Global function from "db.class.php". The checkLogin function ensures that a user is properly logged in by verifying that all essential session variables are set and not empty.
2. **getScenario($sim_id)**
   Global function from "db.class.php". This PHP function, "getScenario", retrieves scenario data from the database based on the provided scenario ID ($sim_id).
3. **getScenarioCompetency($sid)**
   Global function from "db.class.php". This function, "getScenarioCompetency", retrieves competency information associated with a specific scenario from the competency_tbl database table.
4. **getSimPage($sid)**
   Global function from "db.class.php". This PHP function, "getSimPage", retrieves simulation pages associated with a specific scenario ID from the sim_pages table in the database.
5. **getSimBranding($sid)**
   Global function from "db.class.php". This function, "getSimBranding", retrieves branding

information associated with a specific scenario from the sim_branding_tbl database table.

6. **getMultiScenarioType($id)**
   Global function from "db.class.php". The getMultiScenarioType function retrieves data from a database based on the provided scenario type IDs.
7. **SQL Queries**
   *$qsql = "SELECT * FROM question_tbl WHERE question_id = ?";*
   This query retrieves all columns from the question_tbl table for a specific question_id.

   $mcq_sql = "SELECT a., f.* FROM answer_tbl a LEFT JOIN feedback_tbl f ON f.answer_id = a.answer_id WHERE a.question_id = ?";*
   This SQL query string designed to select data from two tables, answer_tbl and feedback_tbl, using a LEFT JOIN as per the question_id passed in WHERE clause.

   *$ddq2_sql = "SELECT * FROM sub_options_tbl WHERE question_id = ?";*
   This SQL query selects data from the sub_options_tbl table based on a specific question_id.

   $ddqop_sql = "SELECT a., f.* FROM answer_tbl a LEFT JOIN feedback_tbl f ON a.answer_id = f.answer_id WHERE a.question_id = ?";*
   This SQL query selects data from two tables, answer_tbl and feedback_tbl, using a LEFT JOIN. This query retrieves all answers for a specific question along with any associated feedback.

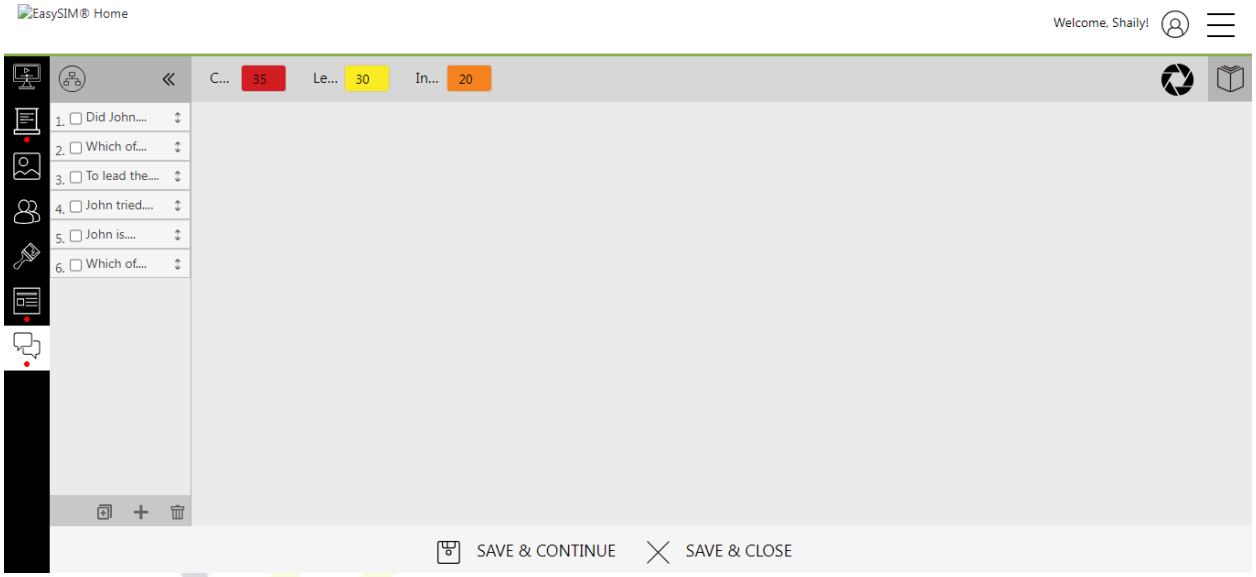
   $swa_sql = "SELECT a., f.* FROM answer_tbl a LEFT JOIN feedback_tbl f ON f.answer_id = a.answer_id WHERE a.question_id = ?";*
   This SQL query retrieves data from two tables, answer_tbl and feedback_tbl, utilizing a LEFT JOIN operation. This query aims to retrieve information about answers to a particular question along with any associated feedback.

JS Functions Used

1. **$(".Add_Drag").click(function(){})**
   This JavaScript code is part of a application where users can create questions with various features like multiple-choice questions (MCQs) and drag-and-drop (DND) questions.
2. **$(".MCQAdd").click(function(){})**
   This code dynamically adds new MCQ options when a certain button (MCQAdd) is clicked. It ensures that each new MCQ option is properly initialized with a text editor for text input and provides functionality to remove the MCQ option if needed.
3. **$(".swipAdd").click(function(){})**
   This code allows users to dynamically add form elements and upload files, with additional functionality for rich text editing and text-to-speech conversion.

Edit Simulation > Hybrid Linear Multiple

Template Name: hybrid-linear-multiple-template.php



PHP Functions Used

1. **checkLogin()**
   Global function from "db.class.php". The checkLogin function ensures that a user is properly logged in by verifying that all essential session variables are set and not empty.
2. **getScenario($sim_id)**
   Global function from "db.class.php". This PHP function, "getScenario", retrieves scenario data from the database based on the provided scenario ID ($sim_id).
3. **getScenarioCompetency($sid)**
   Global function from "db.class.php". This function, "getScenarioCompetency", retrieves competency information associated with a specific scenario from the competency_tbl database table.
4. **getSimPage($sid)**
   Global function from "db.class.php". This PHP function, "getSimPage", retrieves simulation pages associated with a specific scenario ID from the sim_pages table in the database.
5. **getSimBranding($sid)**
   Global function from "db.class.php". This function, "getSimBranding", retrieves branding information associated with a specific scenario from the sim_branding_tbl database table.
6. **getMultiScenarioType($id)**
   Global function from "db.class.php". The getMultiScenarioType function retrieves data from a database based on the provided scenario type IDs.
7. **SQL Queries**
   *$qsql = "SELECT * FROM question_tbl WHERE question_id = ?";*
   This query retrieves all columns from the question_tbl table for a specific question_id.

$mtf_choice_sql = "SELECT answer_id, choice_option, choice_order_no, match_order_no, match_option FROM answer_tbl WHERE question_id = ? AND choice_order_no != '0' AND match_order_no != '0'";
This SQL query selects specific columns from a table named answer_tbl, filtering the rows based on conditions related to question_id, choice_order_no, and match_order_no.

$mtf_choice_sql1 = "SELECT answer_id, choice_option, match_option FROM answer_tbl WHERE question_id = ? AND choice_order_no = '0' AND match_order_no = '0'";
This SQL query selects specific columns from a table named answer_tbl, filtering the rows based on conditions related to question_id, choice_order_no, and match_order_no. This query seems to target rows where both choice_order_no and match_order_no are equal to zero.

$sq_sql = "SELECT answer_id, choice_option FROM answer_tbl WHERE question_id = ?";
This SQL query retrieves the answer_id and choice_option columns from the answer_tbl table where the question_id matches a specific value.

$sort_sql = "SELECT answer_id, choice_option, match_option, match_sorting_item, image FROM answer_tbl WHERE question_id = ?";
This SQL query selects specific columns from a table named answer_tbl, filtering the rows based on the question_id column. This query retrieves data related to answers for a specific question identified by its question_id.

$sortop_sql = "SELECT sub_options_id, answer_id, sub_option FROM sub_options_tbl WHERE answer_id = ?";
This SQL query retrieves sub-options related to a specific answer identified by its answer_id. The sub-options are stored in the sub_options_tbl table.

$ddq_sql = "SELECT answer_id, choice_option, match_option, image, image2 FROM answer_tbl WHERE question_id = ?";
This SQL query retrieves answers related to a specific question identified by its question_id. The answers' details include the option choices, match options, and potentially images associated with those answers.

*$ddq2_sql = "SELECT * FROM sub_options_tbl WHERE question_id = ?";*
This SQL query retrieves all data from the sub_options_tbl table where the question_id matches a specified value. It's used to fetch sub-options associated with a particular question identified by its question_id.

*$ddqop_sql = "SELECT * FROM answer_tbl WHERE question_id = ?";*
This SQL query retrieves all data from the answer_tbl table where the question_id

matches a specified value. It's used to fetch answers associated with a particular question identified by its question_id.

$mcq_sql = "SELECT answer_id, choice_option FROM answer_tbl WHERE question_id = ?";
This SQL query retrieves the answer_id and choice_option columns from the answer_tbl table for rows where the question_id matches a specified value. It's used to fetch specific information about answers associated with a particular question identified by its question_id.

$mmcq_sql = "SELECT answer_id, choice_option FROM answer_tbl WHERE question_id = ?";
The query retrieves the answer_id and choice_option columns from the answer_tbl table for rows where the question_id matches a specific value.

$swa_sql = "SELECT answer_id, choice_option, image FROM answer_tbl WHERE question_id = ?";
This SQL query retrieves the answer_id, choice_option, and image columns from the answer_tbl table for rows where the question_id matches a specified value. It's used to fetch specific information about answers associated with a particular question identified by its question_id, including any images linked to those answers.

JS Functions Used

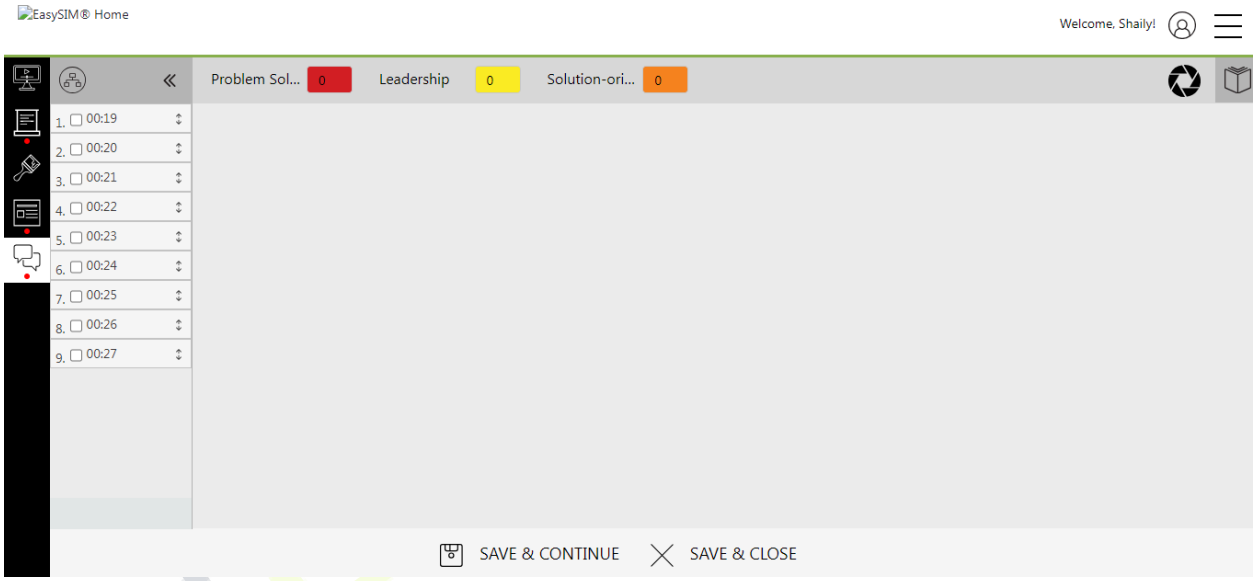
1. **$(".AddAnsICON").click(function(){})**
   This code dynamically adds new MTF options to a form when a certain button (AddAnsICON) is clicked. It ensures that the newly added options are properly initialized, including form validation and rich text editing functionality.
2. **$(".AddAnsICON1").click(function(){})**
   This code dynamically adds new MTF options to a form when a certain button (AddAnsICON1) is clicked. It ensures that the newly added options are properly initialized, including form validation and rich text editing functionality, similar to the previous snippet. However, these options might have different styling or behavior, as indicated by the different class used for their containers.
3. **$(".DNDAddAnsICON1").click(function(){})**
   This code dynamically adds new options for a Drag and Drop question type to a form when a certain button (DNDAddAnsICON1) is clicked. It ensures that the newly added options are properly initialized, including rich text editing functionality.
4. **$(".DNDAddAnsICON").click(function(){})**
   This code dynamically adds new drag and drop options to a form when a certain button (DNDAddAnsICON) is clicked. It ensures that the newly added options are properly initialized, including rich text editing functionality.
5. **$(".SeqAdd").click(function(){})**
   This code dynamically adds new sequence answer options to a form when a certain

button (SeqAdd) is clicked. It ensures that the newly added options are properly initialized, including rich text editing functionality.

6. **$(".ShortAddICON").click(function(){})**
   This code dynamically adds new options for a short answer question with drag-and-drop functionality when a certain button (ShortAddICON) is clicked. It ensures that the newly added options are properly initialized, including rich text editing functionality, and provides functionality to add and remove drag options dynamically.
7. **$('body').on('click', '.ShortDragAddICON', function() {})**
   This code dynamically adds new drag options within a specific sorting item for a short answer question when a certain button (ShortDragAddICON) is clicked. It ensures that the new drag option is properly initialized with rich text editing functionality and provides functionality to remove the drag option if needed.
8. **$(".MCQAdd").click(function(){})**
   This code dynamically adds new MCQ options when a certain button (MCQAdd) is clicked. It ensures that each new MCQ option is properly initialized with a text editor for text input and provides functionality to remove the MCQ option if needed.
9. **$(".MMCQAdd").click(function(){})**
   This function dynamically adds a new set of checkbox and textarea elements for multiple choice questions (MMCQ) each time the .MMCQAdd element is clicked, and initializes a rich text editor for the textarea element.
10. **$(".SwaAdd").click(function(){})**
    This function dynamically adds a new set of radio button, textarea, and image upload elements for single word answer questions (SWA) each time the .SwaAdd element is clicked, and initializes a rich text editor for the textarea element.
11. **$('body').on('click', '.removeOption', function(e) {})**
    This code provides a user-friendly way to delete options from a list and handles the deletion process asynchronously without refreshing the page. Ajax request is sent to the server using the $.ajax method. The request is sent to the URL specified (*includes/process.php*) with the data string containing the information about the answer to be deleted.
12. **$('body').on('click', '.removeSortingOption', function(e) {})**
    This code provides a user-friendly way to delete sorting options and handles the deletion process asynchronously without refreshing the page. Ajax request is sent to the server using the $.ajax method. The request is sent to the URL specified (*includes/process.php*) with the data string containing the information about the sorting answer to be deleted.
13. **$('body').on('click', '.removeSortingSubOption', function(e) {})**
    This code provides a user-friendly way to delete sorting sub-options and handles the deletion process asynchronously without refreshing the page. Ajax request is sent to the server using the $.ajax method. The request is sent to the URL specified (*includes/process.php*) with the data string containing the information about the sorting sub-answer to be deleted.

Edit Simulation > Hybrid Linear Classic Video

Template Name: hybrid-linear-classic-video-template.php



PHP Functions Used

1. **checkLogin()**
   Global function from "db.class.php". The checkLogin function ensures that a user is properly logged in by verifying that all essential session variables are set and not empty.
2. **getScenario($sim_id)**
   Global function from "db.class.php". This PHP function, "getScenario", retrieves scenario data from the database based on the provided scenario ID ($sim_id).
3. **getScenarioCompetency($sid)**
   Global function from "db.class.php". This function, "getScenarioCompetency", retrieves competency information associated with a specific scenario from the competency_tbl database table.
4. **getSimPage($sid)**
   Global function from "db.class.php". This PHP function, "getSimPage", retrieves simulation pages associated with a specific scenario ID from the sim_pages table in the database.
5. **getSimBranding($sid)**
   Global function from "db.class.php". This function, "getSimBranding", retrieves branding information associated with a specific scenario from the sim_branding_tbl database table.
6. **getMultiScenarioType($id)**
   Global function from "db.class.php". The getMultiScenarioType function retrieves data from a database based on the provided scenario type IDs.
7. **SQL Queries**
   *$qsql = "SELECT * FROM question_tbl WHERE question_id = ?";*
   This query retrieves all columns from the question_tbl table for a specific question_id.

$tqsql = "SELECT SUM(ques_val_1) AS ques1, SUM(ques_val_2) AS ques2, SUM(ques_val_3) AS ques3, SUM(ques_val_4) AS ques4, SUM(ques_val_5) AS ques5, SUM(ques_val_6) AS ques6, question_id, videoq_cue_point FROM question_tbl WHERE scenario_id = ?";
This SQL query calculate the total sums of specific question values (ques_val_1 to ques_val_6) for a given scenario, while also retaining some identifying information about the questions and their associated cue points in a video.

$mcq_sql = "SELECT a., f.* FROM answer_tbl a LEFT JOIN feedback_tbl f ON f.answer_id = a.answer_id WHERE a.question_id = ?";*
This SQL query string designed to select data from two tables, answer_tbl and feedback_tbl, using a LEFT JOIN as per the question_id passed in WHERE clause.

*$ddq2_sql = "SELECT * FROM sub_options_tbl WHERE question_id = ?";*
This SQL query selects data from the sub_options_tbl table based on a specific question_id.

$ddqop_sql = "SELECT a., f.* FROM answer_tbl a LEFT JOIN feedback_tbl f ON a.answer_id = f.answer_id WHERE a.question_id = ?";*
This SQL query selects data from two tables, answer_tbl and feedback_tbl, using a LEFT JOIN. This query retrieves all answers for a specific question along with any associated feedback.

$swa_sql = "SELECT a., f.* FROM answer_tbl a LEFT JOIN feedback_tbl f ON f.answer_id = a.answer_id WHERE a.question_id = ?";*
This SQL query retrieves data from two tables, answer_tbl and feedback_tbl, utilizing a LEFT JOIN operation. This query aims to retrieve information about answers to a particular question along with any associated feedback.

JS Functions Used

1. **convertDurationtoSeconds(duration){}**
   This JavaScript function, convertDurationtoSeconds, converts a duration given in the format MM:SS (minutes) into seconds.
2. **$('body').on('click', '.videocueAdd', function(){})**
   This code handles the addition of video cue points, validating the input, and sending the data to the server via an AJAX request. It uses SweetAlert for user notifications and a loading overlay during the AJAX request at *'includes/process.php'* to enhance the user experience.
3. **$('#upload_videoq_file').click(function() {})**
   This function handles the upload of a video file asynchronously without reloading the entire page, providing a smooth user experience. It also dynamically updates the UI

based on the upload result. It sends an AJAX request to a PHP script (*upload.php*) with the form data.

4. **$('body').on('click', '.delete_video_assets', function() {})**
   This function provides a smooth way for users to delete video assets and provides feedback on the success or failure of the deletion process. It sends an AJAX request to a PHP script (*delete-file.php*) with the data string.. It also utilizes a confirmation dialog to ensure users are aware of the action they are taking.
5. **$('#cloud_url').on('keyup', function(){})**
   This function seems to validate a video URL entered by the user, ensuring it matches a specified type (YouTube, Vimeo, or hosted video), enabling/disabling certain UI elements accordingly, and providing feedback to the user through warning messages.
6. **$('#cloud_multi_url').on('keyup', function(){})**
   This function appears to validate multiple video URLs entered by the user, ensuring they match a specified type (YouTube, Vimeo, or hosted video), enabling/disabling certain UI elements accordingly, and providing feedback to the user through warning messages.
7. **$('#add_cloud_video').on('click', function(){})**
   This function is responsible for embedding a cloud-hosted video (from YouTube, Vimeo, or a direct URL) into the webpage when a user clicks a specific button, providing a visual representation of the video content.
8. **$('#add_cloud_multi_video').on('click', function(){})**
   This function embeds a cloud-hosted video (from YouTube, Vimeo, or a direct URL) into the webpage and displays additional content related to the video when a user clicks a specific button. It also stores the URL of the video for future reference.
9. **$('body').on('click', '.delete_cloudv_assets', function() {})**
   This function provides a mechanism for users to delete cloud-hosted video assets, along with associated actions such as clearing the embedded video, resetting input fields, and adjusting the visibility and status of other elements accordingly.
10. **$('body').on('click', '.delete_cloud_multiv_assets', function() {})**
    This function provides a way for users to delete multiple cloud-hosted video assets, along with associated actions such as clearing the embedded video, resetting input fields, and adjusting the visibility and status of other elements accordingly.
11. **$('body').on('click', '.cloud_add_cue', function(){})**
    Ihis function allows users to add cue points to a cloud-hosted video by sending the necessary data to a server-side script for processing. It sends an AJAX POST request to a PHP script (*process.php*) with the constructed data. It provides feedback to the user based on the success or failure of the operation and updates the page accordingly.
12. **$(".Add_Drag").click(function(){})**
    This JavaScript code is part of a application where users can create questions with various features like multiple-choice questions (MCQs) and drag-and-drop (DND) questions.
13. **$(".MCQAdd").click(function(){})**
    This code dynamically adds new MCQ options when a certain button (MCQAdd) is clicked. It ensures that each new MCQ option is properly initialized with a text editor for text input and provides functionality to remove the MCQ option if needed.

14. **$(".SwaAdd").click(function(){})**
    This function dynamically adds a new set of radio button, textarea, and image upload elements for single word answer questions (SWA) each time the .SwaAdd element is clicked, and initializes a rich text editor for the textarea element.

Edit Simulation > Hybrid Linear Multiple Video

Template Name: hybrid-linear-multiple-video-template.php

ImageToBePlacedHere

PHP Functions Used

1. **checkLogin()**
   Global function from "db.class.php". The checkLogin function ensures that a user is properly logged in by verifying that all essential session variables are set and not empty.
2. **getScenario($sim_id)**
   Global function from "db.class.php". This PHP function, "getScenario", retrieves scenario data from the database based on the provided scenario ID ($sim_id).
3. **getScenarioCompetency($sid)**
   Global function from "db.class.php". This function, "getScenarioCompetency", retrieves competency information associated with a specific scenario from the competency_tbl database table.
4. **getSimPage($sid)**
   Global function from "db.class.php". This PHP function, "getSimPage", retrieves simulation pages associated with a specific scenario ID from the sim_pages table in the database.
5. **getSimBranding($sid)**
   Global function from "db.class.php". This function, "getSimBranding", retrieves branding information associated with a specific scenario from the sim_branding_tbl database table.
6. **getMultiScenarioType($id)**
   Global function from "db.class.php". The getMultiScenarioType function retrieves data from a database based on the provided scenario type IDs.
7. **SQL Queries**
   *$qsql = "SELECT * FROM question_tbl WHERE question_id = ?";*
   This query retrieves all columns from the question_tbl table for a specific question_id.

   *$feedsql = "SELECT * FROM feedback_tbl WHERE question_id =:question_id";*
   This SQL query retrieves all rows from the feedback_tbl table where the value of the question_id column matches the value provided for the :question_id parameter.

   $tqsql = "SELECT SUM(ques_val_1) AS ques1, SUM(ques_val_2) AS ques2, SUM(ques_val_3) AS ques3, SUM(ques_val_4) AS ques4, SUM(ques_val_5) AS ques5, SUM(ques_val_6) AS ques6, question_id, videoq_cue_point FROM question_tbl

WHERE scenario_id = ?";
This SQL query calculate the total sums of specific question values (ques_val_1 to ques_val_6) for a given scenario, while also retaining some identifying information about the questions and their associated cue points in a video.

$mtf_choice_sql = "SELECT answer_id, choice_option, choice_order_no, match_order_no, match_option FROM answer_tbl WHERE question_id = ? AND choice_order_no != '0' AND match_order_no != '0'";
This SQL query selects specific columns from a table named answer_tbl, filtering the rows based on conditions related to question_id, choice_order_no, and match_order_no.

$mtf_choice_sql1 = "SELECT answer_id, choice_option, match_option FROM answer_tbl WHERE question_id = ? AND choice_order_no = '0' AND match_order_no = '0'";
This SQL query selects specific columns from a table named answer_tbl, filtering the rows based on conditions related to question_id, choice_order_no, and match_order_no. This query seems to target rows where both choice_order_no and match_order_no are equal to zero.

$sq_sql = "SELECT answer_id, choice_option FROM answer_tbl WHERE question_id = ?";
This SQL query retrieves the answer_id and choice_option columns from the answer_tbl table where the question_id matches a specific value.

$sort_sql = "SELECT answer_id, choice_option, match_option, match_sorting_item, image FROM answer_tbl WHERE question_id = ?";
This SQL query selects specific columns from a table named answer_tbl, filtering the rows based on the question_id column. This query retrieves data related to answers for a specific question identified by its question_id.

$sortop_sql = "SELECT sub_options_id, answer_id, sub_option FROM sub_options_tbl WHERE answer_id = ?";
This SQL query retrieves sub-options related to a specific answer identified by its answer_id. The sub-options are stored in the sub_options_tbl table.

$ddq_sql = "SELECT answer_id, choice_option, match_option, image, image2 FROM answer_tbl WHERE question_id = ?";
This SQL query retrieves answers related to a specific question identified by its question_id. The answers' details include the option choices, match options, and potentially images associated with those answers.

*$ddq2_sql = "SELECT * FROM sub_options_tbl WHERE question_id = ?";*
This SQL query retrieves all data from the sub_options_tbl table where the question_id matches a specified value. It's used to fetch sub-options associated with a particular

question identified by its question_id.

*$ddqop_sql = "SELECT * FROM answer_tbl WHERE question_id = ?";*
This SQL query retrieves all data from the answer_tbl table where the question_id matches a specified value. It's used to fetch answers associated with a particular question identified by its question_id.

$mcq_sql = "SELECT answer_id, choice_option FROM answer_tbl WHERE question_id = ?";
This SQL query retrieves the answer_id and choice_option columns from the answer_tbl table for rows where the question_id matches a specified value. It's used to fetch specific information about answers associated with a particular question identified by its question_id.

$mmcq_sql = "SELECT answer_id, choice_option FROM answer_tbl WHERE question_id = ?";
The query retrieves the answer_id and choice_option columns from the answer_tbl table for rows where the question_id matches a specific value.

$swa_sql = "SELECT answer_id, choice_option, image FROM answer_tbl WHERE question_id = ?";
This SQL query retrieves the answer_id, choice_option, and image columns from the answer_tbl table for rows where the question_id matches a specified value. It's used to fetch specific information about answers associated with a particular question identified by its question_id, including any images linked to those answers.

JS Functions Used

1. **convertDurationtoSeconds(duration){}**
   This JavaScript function, convertDurationtoSeconds, converts a duration given in the format MM:SS (minutes) into seconds.
2. **$('body').on('click', '.videocueAdd', function(){})**
   This code handles the addition of video cue points, validating the input, and sending the data to the server via an AJAX request. It uses SweetAlert for user notifications and a loading overlay during the AJAX request at *'includes/process.php'* to enhance the user experience.
3. **$('#upload_videoq_file').click(function() {})**
   This function handles the upload of a video file asynchronously without reloading the entire page, providing a smooth user experience. It also dynamically updates the UI based on the upload result. It sends an AJAX request to a PHP script (*upload.php*) with the form data.
4. **$('body').on('click', '.delete_video_assets', function() {})**
   This function provides a smooth way for users to delete video assets and provides feedback on the success or failure of the deletion process. It sends an AJAX request to

a PHP script (*delete-file.php*) with the data string.. It also utilizes a confirmation dialog to ensure users are aware of the action they are taking.

5. **$('#cloud_url').on('keyup', function(){})**
   This function seems to validate a video URL entered by the user, ensuring it matches a specified type (YouTube, Vimeo, or hosted video), enabling/disabling certain UI elements accordingly, and providing feedback to the user through warning messages.
6. **$('#cloud_multi_url').on('keyup', function(){})**
   This function appears to validate multiple video URLs entered by the user, ensuring they match a specified type (YouTube, Vimeo, or hosted video), enabling/disabling certain UI elements accordingly, and providing feedback to the user through warning messages.
7. **$('#add_cloud_video').on('click', function(){})**
   This function is responsible for embedding a cloud-hosted video (from YouTube, Vimeo, or a direct URL) into the webpage when a user clicks a specific button, providing a visual representation of the video content.
8. **$('#add_cloud_multi_video').on('click', function(){})**
   This function embeds a cloud-hosted video (from YouTube, Vimeo, or a direct URL) into the webpage and displays additional content related to the video when a user clicks a specific button. It also stores the URL of the video for future reference.
9. **$('body').on('click', '.delete_cloudv_assets', function() {})**
   This function provides a mechanism for users to delete cloud-hosted video assets, along with associated actions such as clearing the embedded video, resetting input fields, and adjusting the visibility and status of other elements accordingly.
10. **$('body').on('click', '.delete_cloud_multiv_assets', function() {})**
    This function provides a way for users to delete multiple cloud-hosted video assets, along with associated actions such as clearing the embedded video, resetting input fields, and adjusting the visibility and status of other elements accordingly.
11. **$('body').on('click', '.cloud_add_cue', function(){})**
    Ihis function allows users to add cue points to a cloud-hosted video by sending the necessary data to a server-side script for processing. It sends an AJAX POST request to a PHP script (*process.php*) with the constructed data. It provides feedback to the user based on the success or failure of the operation and updates the page accordingly.
12. **$(".AddAnsICON").click(function(){})**
    This code dynamically adds new MTF options to a form when a certain button (AddAnsICON) is clicked. It ensures that the newly added options are properly initialized, including form validation and rich text editing functionality.
13. **$(".AddAnsICON1").click(function(){})**
    This code dynamically adds new MTF options to a form when a certain button (AddAnsICON1) is clicked. It ensures that the newly added options are properly initialized, including form validation and rich text editing functionality, similar to the previous snippet. However, these options might have different styling or behavior, as indicated by the different class used for their containers.
14. **$(".DNDAddAnsICON1").click(function(){})**
    This code dynamically adds new options for a Drag and Drop question type to a form

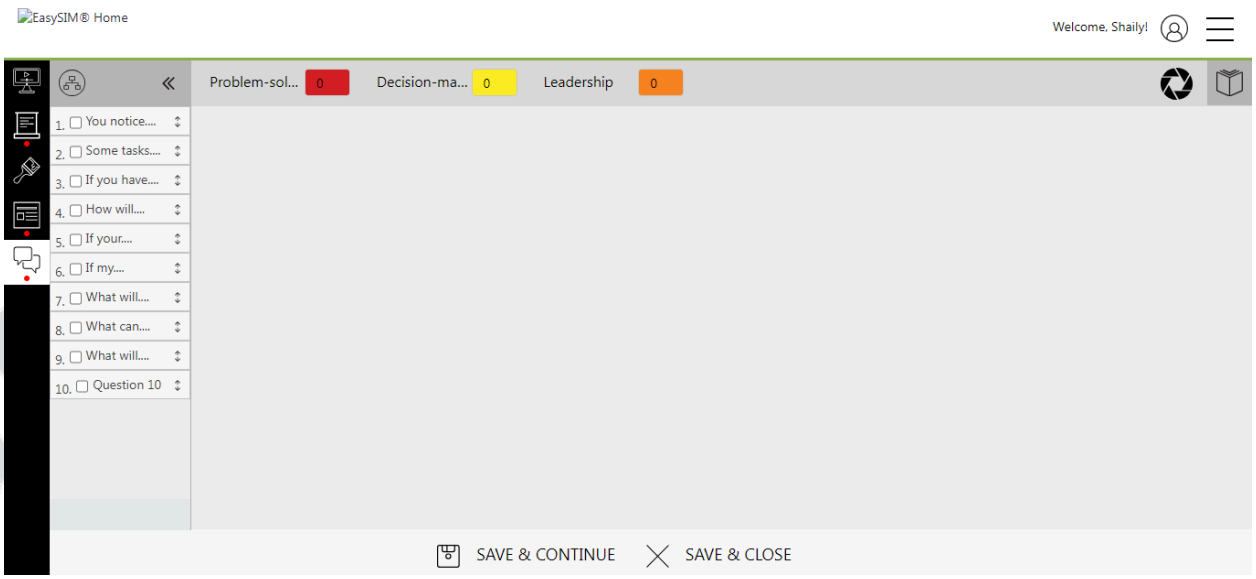
when a certain button (DNDAddAnsICON1) is clicked. It ensures that the newly added options are properly initialized, including rich text editing functionality.

15. **$(".DNDAddAnsICON").click(function(){})**
    This code dynamically adds new drag and drop options to a form when a certain button (DNDAddAnsICON) is clicked. It ensures that the newly added options are properly initialized, including rich text editing functionality.
16. **$(".SeqAdd").click(function(){})**
    This code dynamically adds new sequence answer options to a form when a certain button (SeqAdd) is clicked. It ensures that the newly added options are properly initialized, including rich text editing functionality.
17. **$(".ShortAddICON").click(function(){})**
    This code dynamically adds new options for a short answer question with drag-and-drop functionality when a certain button (ShortAddICON) is clicked. It ensures that the newly added options are properly initialized, including rich text editing functionality, and provides functionality to add and remove drag options dynamically.
18. **$('body').on('click', '.ShortDragAddICON', function() {})**
    This code dynamically adds new drag options within a specific sorting item for a short answer question when a certain button (ShortDragAddICON) is clicked. It ensures that the new drag option is properly initialized with rich text editing functionality and provides functionality to remove the drag option if needed.
19. **$(".MCQAdd").click(function(){})**
    This code dynamically adds new MCQ options when a certain button (MCQAdd) is clicked. It ensures that each new MCQ option is properly initialized with a text editor for text input and provides functionality to remove the MCQ option if needed.
20. **$(".MMCQAdd").click(function(){})**
    This function dynamically adds a new set of checkbox and textarea elements for multiple choice questions (MMCQ) each time the .MMCQAdd element is clicked, and initializes a rich text editor for the textarea element.
21. **$(".SwaAdd").click(function(){})**
    This function dynamically adds a new set of radio button, textarea, and image upload elements for single word answer questions (SWA) each time the .SwaAdd element is clicked, and initializes a rich text editor for the textarea element.
22. **$('body').on('click', '.removeOption', function(e) {})**
    This code provides a user-friendly way to delete options from a list and handles the deletion process asynchronously without refreshing the page. Ajax request is sent to the server using the $.ajax method. The request is sent to the URL specified (*includes/process.php*) with the data string containing the information about the answer to be deleted.
23. **$('body').on('click', '.removeSortingOption', function(e) {})**
    This code provides a user-friendly way to delete sorting options and handles the deletion process asynchronously without refreshing the page. Ajax request is sent to the server using the $.ajax method. The request is sent to the URL specified (*includes/process.php*) with the data string containing the information about the sorting answer to be deleted.

24. **$('body').on('click', '.removeSortingSubOption', function(e) {})**
    This code provides a user-friendly way to delete sorting sub-options and handles the deletion process asynchronously without refreshing the page. Ajax request is sent to the server using the $.ajax method. The request is sent to the URL specified (*includes/process.php*) with the data string containing the information about the sorting sub-answer to be deleted.

Edit Simulation > Linear Classic Video

Template Name: linear-classic-video-template.php



PHP Functions Used

1. **checkLogin()**
   Global function from "db.class.php". The checkLogin function ensures that a user is properly logged in by verifying that all essential session variables are set and not empty.
2. **getScenario($sim_id)**
   Global function from "db.class.php". This PHP function, "getScenario", retrieves scenario data from the database based on the provided scenario ID ($sim_id).
3. **getScenarioCompetency($sid)**
   Global function from "db.class.php". This function, "getScenarioCompetency", retrieves competency information associated with a specific scenario from the competency_tbl database table.
4. **getSimPage($sid)**
   Global function from "db.class.php". This PHP function, "getSimPage", retrieves simulation pages associated with a specific scenario ID from the sim_pages table in the database.
5. **getSimBranding($sid)**
   Global function from "db.class.php". This function, "getSimBranding", retrieves branding

information associated with a specific scenario from the sim_branding_tbl database table.

6. **getMultiScenarioType($id)**
   Global function from "db.class.php". The getMultiScenarioType function retrieves data from a database based on the provided scenario type IDs.
7. **SQL Queries**
   *$qsql = "SELECT * FROM question_tbl WHERE question_id = ?";*
   This query retrieves all columns from the question_tbl table for a specific question_id.

   $tqsql = "SELECT SUM(ques_val_1) AS ques1, SUM(ques_val_2) AS ques2, SUM(ques_val_3) AS ques3, SUM(ques_val_4) AS ques4, SUM(ques_val_5) AS ques5, SUM(ques_val_6) AS ques6, question_id, videoq_cue_point FROM question_tbl WHERE scenario_id = ?";
   This SQL query calculate the total sums of specific question values (ques_val_1 to ques_val_6) for a given scenario, while also retaining some identifying information about the questions and their associated cue points in a video.

   $mcq_sql = "SELECT a., f.* FROM answer_tbl a LEFT JOIN feedback_tbl f ON f.answer_id = a.answer_id WHERE a.question_id = ?";*
   This SQL query string designed to select data from two tables, answer_tbl and feedback_tbl, using a LEFT JOIN as per the question_id passed in WHERE clause.

   *$ddq2_sql = "SELECT * FROM sub_options_tbl WHERE question_id = ?";*
   This SQL query selects data from the sub_options_tbl table based on a specific question_id.

   $ddqop_sql = "SELECT a., f.* FROM answer_tbl a LEFT JOIN feedback_tbl f ON a.answer_id = f.answer_id WHERE a.question_id = ?";*
   This SQL query selects data from two tables, answer_tbl and feedback_tbl, using a LEFT JOIN. This query retrieves all answers for a specific question along with any associated feedback.

   $swa_sql = "SELECT a., f.* FROM answer_tbl a LEFT JOIN feedback_tbl f ON f.answer_id = a.answer_id WHERE a.question_id = ?";*
   This SQL query retrieves data from two tables, answer_tbl and feedback_tbl, utilizing a LEFT JOIN operation. This query aims to retrieve information about answers to a particular question along with any associated feedback.

JS Functions Used

1. **convertDurationtoSeconds(duration){}**
   This JavaScript function, convertDurationtoSeconds, converts a duration given in the format MM:SS (minutes) into seconds.

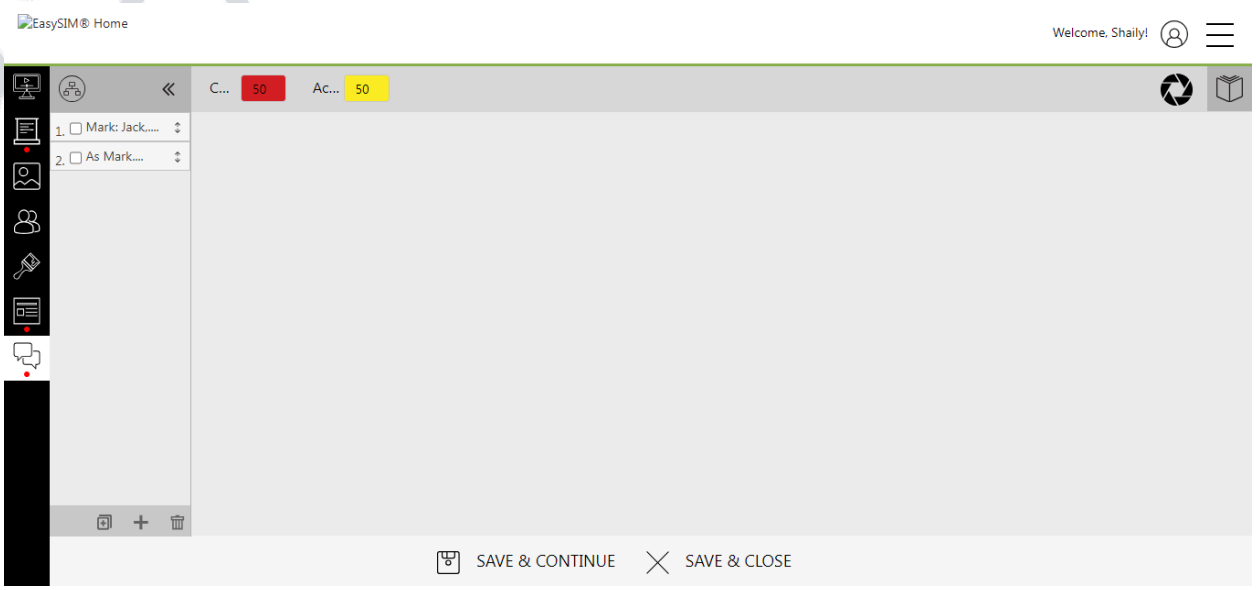
2. **$('body').on('click', '.videocueAdd', function(){})**
   This code handles the addition of video cue points, validating the input, and sending the data to the server via an AJAX request. It uses SweetAlert for user notifications and a loading overlay during the AJAX request at *'includes/process.php'* to enhance the user experience.
3. **$('#upload_videoq_file').click(function() {})**
   This function handles the upload of a video file asynchronously without reloading the entire page, providing a smooth user experience. It also dynamically updates the UI based on the upload result. It sends an AJAX request to a PHP script (*upload.php*) with the form data.
4. **$('body').on('click', '.delete_video_assets', function() {})**
   This function provides a smooth way for users to delete video assets and provides feedback on the success or failure of the deletion process. It sends an AJAX request to a PHP script (*delete-file.php*) with the data string.. It also utilizes a confirmation dialog to ensure users are aware of the action they are taking.
5. **$('#cloud_url').on('keyup', function(){})**
   This function seems to validate a video URL entered by the user, ensuring it matches a specified type (YouTube, Vimeo, or hosted video), enabling/disabling certain UI elements accordingly, and providing feedback to the user through warning messages.
6. **$('#cloud_multi_url').on('keyup', function(){})**
   This function appears to validate multiple video URLs entered by the user, ensuring they match a specified type (YouTube, Vimeo, or hosted video), enabling/disabling certain UI elements accordingly, and providing feedback to the user through warning messages.
7. **$('#add_cloud_video').on('click', function(){})**
   This function is responsible for embedding a cloud-hosted video (from YouTube, Vimeo, or a direct URL) into the webpage when a user clicks a specific button, providing a visual representation of the video content.
8. **$('#add_cloud_multi_video').on('click', function(){})**
   This function embeds a cloud-hosted video (from YouTube, Vimeo, or a direct URL) into the webpage and displays additional content related to the video when a user clicks a specific button. It also stores the URL of the video for future reference.
9. **$('body').on('click', '.delete_cloudv_assets', function() {})**
   This function provides a mechanism for users to delete cloud-hosted video assets, along with associated actions such as clearing the embedded video, resetting input fields, and adjusting the visibility and status of other elements accordingly.
10. **$('body').on('click', '.delete_cloud_multiv_assets', function() {})**
    This function provides a way for users to delete multiple cloud-hosted video assets, along with associated actions such as clearing the embedded video, resetting input fields, and adjusting the visibility and status of other elements accordingly.
11. **$('body').on('click', '.cloud_add_cue', function(){})**
    Ihis function allows users to add cue points to a cloud-hosted video by sending the necessary data to a server-side script for processing. It sends an AJAX POST request to a PHP script (*process.php*) with the constructed data. It provides feedback to the user based on the success or failure of the operation and updates the page accordingly.

12. **$(".AddAnsICON").click(function(){})**
    This code dynamically adds new MTF options to a form when a certain button (AddAnsICON) is clicked. It ensures that the newly added options are properly initialized, including form validation and rich text editing functionality.
13. **$(".AddAnsICON1").click(function(){})**
    This code dynamically adds new MTF options to a form when a certain button (AddAnsICON1) is clicked. It ensures that the newly added options are properly initialized, including form validation and rich text editing functionality, similar to the previous snippet. However, these options might have different styling or behavior, as indicated by the different class used for their containers.
14. **$(".DNDAddAnsICON1").click(function(){})**
    This code dynamically adds new options for a Drag and Drop question type to a form when a certain button (DNDAddAnsICON1) is clicked. It ensures that the newly added options are properly initialized, including rich text editing functionality.
15. **$(".DNDAddAnsICON").click(function(){})**
    This code dynamically adds new drag and drop options to a form when a certain button (DNDAddAnsICON) is clicked. It ensures that the newly added options are properly initialized, including rich text editing functionality.
16. **$(".SeqAdd").click(function(){})**
    This code dynamically adds new sequence answer options to a form when a certain button (SeqAdd) is clicked. It ensures that the newly added options are properly initialized, including rich text editing functionality.
17. **$(".ShortAddICON").click(function(){})**
    This code dynamically adds new options for a short answer question with drag-and-drop functionality when a certain button (ShortAddICON) is clicked. It ensures that the newly added options are properly initialized, including rich text editing functionality, and provides functionality to add and remove drag options dynamically.
18. **$('body').on('click', '.ShortDragAddICON', function() {})**
    This code dynamically adds new drag options within a specific sorting item for a short answer question when a certain button (ShortDragAddICON) is clicked. It ensures that the new drag option is properly initialized with rich text editing functionality and provides functionality to remove the drag option if needed.
19. **$(".MCQAdd").click(function(){})**
    This code dynamically adds new MCQ options when a certain button (MCQAdd) is clicked. It ensures that each new MCQ option is properly initialized with a text editor for text input and provides functionality to remove the MCQ option if needed.
20. **$(".MMCQAdd").click(function(){})**
    This function dynamically adds a new set of checkbox and textarea elements for multiple choice questions (MMCQ) each time the .MMCQAdd element is clicked, and initializes a rich text editor for the textarea element.
21. **$(".SwaAdd").click(function(){})**
    This function dynamically adds a new set of radio button, textarea, and image upload elements for single word answer questions (SWA) each time the .SwaAdd element is clicked, and initializes a rich text editor for the textarea element.

22. **$('body').on('click', '.removeOption', function(e) {})**
    This code provides a user-friendly way to delete options from a list and handles the deletion process asynchronously without refreshing the page. Ajax request is sent to the server using the $.ajax method. The request is sent to the URL specified (*includes/process.php*) with the data string containing the information about the answer to be deleted.
23. **$('body').on('click', '.removeSortingOption', function(e) {})**
    This code provides a user-friendly way to delete sorting options and handles the deletion process asynchronously without refreshing the page. Ajax request is sent to the server using the $.ajax method. The request is sent to the URL specified (*includes/process.php*) with the data string containing the information about the sorting answer to be deleted.
24. **$('body').on('click', '.removeSortingSubOption', function(e) {})**
    This code provides a user-friendly way to delete sorting sub-options and handles the deletion process asynchronously without refreshing the page. Ajax request is sent to the server using the $.ajax method. The request is sent to the URL specified (*includes/process.php*) with the data string containing the information about the sorting sub-answer to be deleted.

Edit Simulation > Linear Video Multiple

Template Name: linear-video-multiple-template.php



PHP Functions Used

1. **checkLogin()**
   Global function from "db.class.php". The checkLogin function ensures that a user is properly logged in by verifying that all essential session variables are set and not empty.

2. **getScenario($sim_id)**
   Global function from “db.class.php”. This PHP function, "getScenario", retrieves scenario data from the database based on the provided scenario ID ($sim_id).
3. **getScenarioCompetency($sid)**
   Global function from “db.class.php”. This function, "getScenarioCompetency", retrieves competency information associated with a specific scenario from the competency_tbl database table.
4. **getSimPage($sid)**
   Global function from “db.class.php”. This PHP function, "getSimPage", retrieves simulation pages associated with a specific scenario ID from the sim_pages table in the database.
5. **getSimBranding($sid)**
   Global function from “db.class.php”. This function, “getSimBranding”, retrieves branding information associated with a specific scenario from the sim_branding_tbl database table.
6. **getMultiScenarioType($id)**
   Global function from “db.class.php”. The getMultiScenarioType function retrieves data from a database based on the provided scenario type IDs.
7. **SQL Queries**
   *$qsql = "SELECT * FROM question_tbl WHERE question_id = ?";*
   This query retrieves all columns from the question_tbl table for a specific question_id.

   *$feedsql = "SELECT * FROM feedback_tbl WHERE question_id =:question_id";*
   This SQL query retrieves all rows from the feedback_tbl table where the value of the question_id column matches the value provided for the :question_id parameter.

   $tqsql = "SELECT SUM(ques_val_1) AS ques1, SUM(ques_val_2) AS ques2, SUM(ques_val_3) AS ques3, SUM(ques_val_4) AS ques4, SUM(ques_val_5) AS ques5, SUM(ques_val_6) AS ques6, question_id, videoq_cue_point FROM question_tbl WHERE scenario_id = ?";
   This SQL query calculate the total sums of specific question values (ques_val_1 to ques_val_6) for a given scenario, while also retaining some identifying information about the questions and their associated cue points in a video.

   $mtf_choice_sql = "SELECT answer_id, choice_option, choice_order_no, match_order_no, match_option FROM answer_tbl WHERE question_id = ? AND choice_order_no != '0' AND match_order_no != '0'";
   This SQL query selects specific columns from a table named answer_tbl, filtering the rows based on conditions related to question_id, choice_order_no, and match_order_no.

   $mtf_choice_sql1 = "SELECT answer_id, choice_option, match_option FROM answer_tbl WHERE question_id = ? AND choice_order_no = '0' AND match_order_no = '0'";
   This SQL query selects specific columns from a table named answer_tbl, filtering the

rows based on conditions related to question_id, choice_order_no, and match_order_no. This query seems to target rows where both choice_order_no and match_order_no are equal to zero.

$sq_sql = "SELECT answer_id, choice_option FROM answer_tbl WHERE question_id = ?";
This SQL query retrieves the answer_id and choice_option columns from the answer_tbl table where the question_id matches a specific value.

$sort_sql = "SELECT answer_id, choice_option, match_option, match_sorting_item, image FROM answer_tbl WHERE question_id = ?";
This SQL query selects specific columns from a table named answer_tbl, filtering the rows based on the question_id column. This query retrieves data related to answers for a specific question identified by its question_id.

$ddq_sql = "SELECT answer_id, choice_option, match_option, image, image2 FROM answer_tbl WHERE question_id = ?";
This SQL query retrieves answers related to a specific question identified by its question_id. The answers' details include the option choices, match options, and potentially images associated with those answers.

*$ddq2_sql = "SELECT * FROM sub_options_tbl WHERE question_id = ?";*
This SQL query selects data from the sub_options_tbl table based on a specific question_id.

*$ddqop_sql = "SELECT * FROM answer_tbl WHERE question_id = ?";*
This SQL query retrieves all data from the answer_tbl table where the question_id matches a specified value. It's used to fetch answers associated with a particular question identified by its question_id.

$mcq_sql = "SELECT answer_id, choice_option FROM answer_tbl WHERE question_id = ?";
This SQL query retrieves the answer_id and choice_option columns from the answer_tbl table for rows where the question_id matches a specified value. It's used to fetch specific information about answers associated with a particular question identified by its question_id.

$mmcq_sql = "SELECT answer_id, choice_option FROM answer_tbl WHERE question_id = ?";
The query retrieves the answer_id and choice_option columns from the answer_tbl table for rows where the question_id matches a specific value.

$swa_sql = "SELECT answer_id, choice_option, image FROM answer_tbl WHERE question_id = ?";

This SQL query retrieves the answer_id, choice_option, and image columns from the answer_tbl table for rows where the question_id matches a specified value. It's used to fetch specific information about answers associated with a particular question identified by its question_id, including any images linked to those answers.

JS Functions Used

1. **convertDurationtoSeconds(duration){}**
   This JavaScript function, convertDurationtoSeconds, converts a duration given in the format MM:SS (minutes) into seconds.
2. **$('body').on('click', '.videocueAdd', function(){})**
   This code handles the addition of video cue points, validating the input, and sending the data to the server via an AJAX request. It uses SweetAlert for user notifications and a loading overlay during the AJAX request at 'includes/process.php' to enhance the user experience.
3. **$('#upload_videoq_file').click(function() {})**
   This function handles the upload of a video file asynchronously without reloading the entire page, providing a smooth user experience. It also dynamically updates the UI based on the upload result. It sends an AJAX request to a PHP script (*upload.php*) with the form data.
4. **$('body').on('click', '.delete_video_assets', function() {})**
   This function provides a smooth way for users to delete video assets and provides feedback on the success or failure of the deletion process. It sends an AJAX request to a PHP script (*delete-file.php*) with the data string.. It also utilizes a confirmation dialog to ensure users are aware of the action they are taking.
5. **$('#cloud_url').on('keyup', function(){})**
   This function seems to validate a video URL entered by the user, ensuring it matches a specified type (YouTube, Vimeo, or hosted video), enabling/disabling certain UI elements accordingly, and providing feedback to the user through warning messages.
6. **$('#cloud_multi_url').on('keyup', function(){})**
   This function appears to validate multiple video URLs entered by the user, ensuring they match a specified type (YouTube, Vimeo, or hosted video), enabling/disabling certain UI elements accordingly, and providing feedback to the user through warning messages.
7. **$('#add_cloud_video').on('click', function(){})**
   This function is responsible for embedding a cloud-hosted video (from YouTube, Vimeo, or a direct URL) into the webpage when a user clicks a specific button, providing a visual representation of the video content.
8. **$('#add_cloud_multi_video').on('click', function(){})**
   This function embeds a cloud-hosted video (from YouTube, Vimeo, or a direct URL) into the webpage and displays additional content related to the video when a user clicks a specific button. It also stores the URL of the video for future reference.
9. **$('body').on('click', '.delete_cloudv_assets', function() {})**
   This function provides a mechanism for users to delete cloud-hosted video assets, along

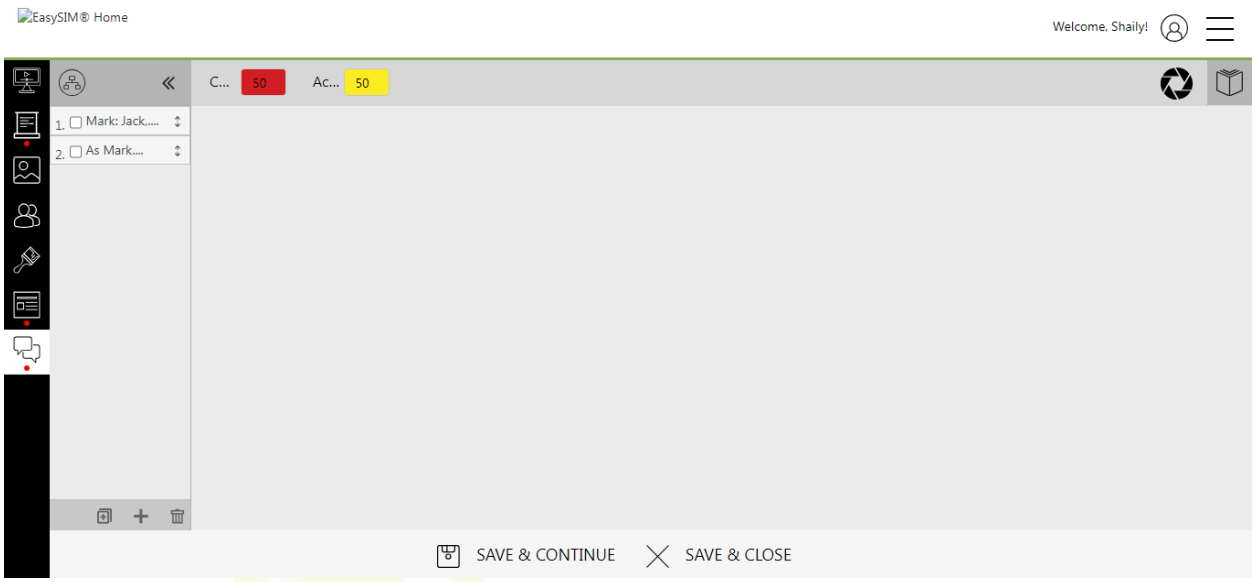
with associated actions such as clearing the embedded video, resetting input fields, and adjusting the visibility and status of other elements accordingly.

10. **$('body').on('click', '.delete_cloud_multiv_assets', function() {})**
    This function provides a way for users to delete multiple cloud-hosted video assets, along with associated actions such as clearing the embedded video, resetting input fields, and adjusting the visibility and status of other elements accordingly.
11. **$('body').on('click', '.cloud_add_cue', function(){})**
    Ihis function allows users to add cue points to a cloud-hosted video by sending the necessary data to a server-side script for processing. It sends an AJAX POST request to a PHP script (*process.php*) with the constructed data. It provides feedback to the user based on the success or failure of the operation and updates the page accordingly.
12. **$(".AddAnsICON").click(function(){})**
    This code dynamically adds new MTF options to a form when a certain button (AddAnsICON) is clicked. It ensures that the newly added options are properly initialized, including form validation and rich text editing functionality.
13. **$(".AddAnsICON1").click(function(){})**
    This code dynamically adds new MTF options to a form when a certain button (AddAnsICON1) is clicked. It ensures that the newly added options are properly initialized, including form validation and rich text editing functionality, similar to the previous snippet. However, these options might have different styling or behavior, as indicated by the different class used for their containers.
14. **$(".DNDAddAnsICON1").click(function(){})**
    This code dynamically adds new options for a Drag and Drop question type to a form when a certain button (DNDAddAnsICON1) is clicked. It ensures that the newly added options are properly initialized, including rich text editing functionality.
15. **$(".DNDAddAnsICON").click(function(){})**
    This code dynamically adds new drag and drop options to a form when a certain button (DNDAddAnsICON) is clicked. It ensures that the newly added options are properly initialized, including rich text editing functionality.
16. **$(".SeqAdd").click(function(){})**
    This code dynamically adds new sequence answer options to a form when a certain button (SeqAdd) is clicked. It ensures that the newly added options are properly initialized, including rich text editing functionality.
17. **$(".ShortAddICON").click(function(){})**
    This code dynamically adds new options for a short answer question with drag-and-drop functionality when a certain button (ShortAddICON) is clicked. It ensures that the newly added options are properly initialized, including rich text editing functionality, and provides functionality to add and remove drag options dynamically.
18. **$('body').on('click', '.ShortDragAddICON', function() {})**
    This code dynamically adds new drag options within a specific sorting item for a short answer question when a certain button (ShortDragAddICON) is clicked. It ensures that the new drag option is properly initialized with rich text editing functionality and provides functionality to remove the drag option if needed.

19. **$(".MCQAdd").click(function(){})**
    This code dynamically adds new MCQ options when a certain button (MCQAdd) is clicked. It ensures that each new MCQ option is properly initialized with a text editor for text input and provides functionality to remove the MCQ option if needed.
20. **$(".MMCQAdd").click(function(){})**
    This function dynamically adds a new set of checkbox and textarea elements for multiple choice questions (MMCQ) each time the .MMCQAdd element is clicked, and initializes a rich text editor for the textarea element.
21. **$(".SwaAdd").click(function(){})**
    This function dynamically adds a new set of radio button, textarea, and image upload elements for single word answer questions (SWA) each time the .SwaAdd element is clicked, and initializes a rich text editor for the textarea element.
22. **$('body').on('click', '.removeOption', function(e) {})**
    This code provides a user-friendly way to delete options from a list and handles the deletion process asynchronously without refreshing the page. Ajax request is sent to the server using the $.ajax method. The request is sent to the URL specified (*includes/process.php*) with the data string containing the information about the answer to be deleted.
23. **$('body').on('click', '.removeSortingOption', function(e) {})**
    This code provides a user-friendly way to delete sorting options and handles the deletion process asynchronously without refreshing the page. Ajax request is sent to the server using the $.ajax method. The request is sent to the URL specified (*includes/process.php*) with the data string containing the information about the sorting answer to be deleted.
24. **$('body').on('click', '.removeSortingSubOption', function(e) {})**
    This code provides a user-friendly way to delete sorting sub-options and handles the deletion process asynchronously without refreshing the page. Ajax request is sent to the server using the $.ajax method. The request is sent to the URL specified (*includes/process.php*) with the data string containing the information about the sorting sub-answer to be deleted.

Edit Simulation > Linear Video Multiple

Template Name: linear-classic-template.php



PHP Functions Used

1. **checkLogin()**
   Global function from "db.class.php". The checkLogin function ensures that a user is properly logged in by verifying that all essential session variables are set and not empty.
2. **getScenario($sim_id)**
   Global function from "db.class.php". This PHP function, "getScenario", retrieves scenario data from the database based on the provided scenario ID ($sim_id).
3. **getScenarioCompetency($sid)**
   Global function from "db.class.php". This function, "getScenarioCompetency", retrieves competency information associated with a specific scenario from the competency_tbl database table.
4. **getSimPage($sid)**
   Global function from "db.class.php". This PHP function, "getSimPage", retrieves simulation pages associated with a specific scenario ID from the sim_pages table in the database.
5. **getSimBranding($sid)**
   Global function from "db.class.php". This function, "getSimBranding", retrieves branding information associated with a specific scenario from the sim_branding_tbl database table.
6. **getMultiScenarioType($id)**
   Global function from "db.class.php". The getMultiScenarioType function retrieves data from a database based on the provided scenario type IDs.
7. **SQL Queries**
   *$qsql = "SELECT * FROM question_tbl WHERE question_id = ?";*
   This query retrieves all columns from the question_tbl table for a specific question_id.

   $tqsql = "SELECT SUM(ques_val_1) AS ques1, SUM(ques_val_2) AS ques2, SUM(ques_val_3) AS ques3, SUM(ques_val_4) AS ques4, SUM(ques_val_5) AS ques5,

SUM(ques_val_6) AS ques6 FROM question_tbl WHERE scenario_id = ?";
This SQL query retrieves the sum of values in six different columns (ques_val_1 to ques_val_6) from the question_tbl table for rows where the scenario_id matches a specific value. Each sum is given an alias (ques1 to ques6) for easier reference in the output. The actual value for scenario_id will be supplied at runtime when the query is executed.

$mcq_sql = "SELECT a., f.* FROM answer_tbl a LEFT JOIN feedback_tbl f ON f.answer_id = a.answer_id WHERE a.question_id = ?";*
This SQL query string designed to select data from two tables, answer_tbl and feedback_tbl, using a LEFT JOIN as per the question_id passed in WHERE clause.

*$ddq2_sql = "SELECT * FROM sub_options_tbl WHERE question_id = ?";*
This SQL query selects data from the sub_options_tbl table based on a specific question_id.

$ddqop_sql = "SELECT a., f.* FROM answer_tbl a LEFT JOIN feedback_tbl f ON a.answer_id = f.answer_id WHERE a.question_id = ?";*
This SQL query selects data from two tables, answer_tbl and feedback_tbl, using a LEFT JOIN. This query retrieves all answers for a specific question along with any associated feedback.

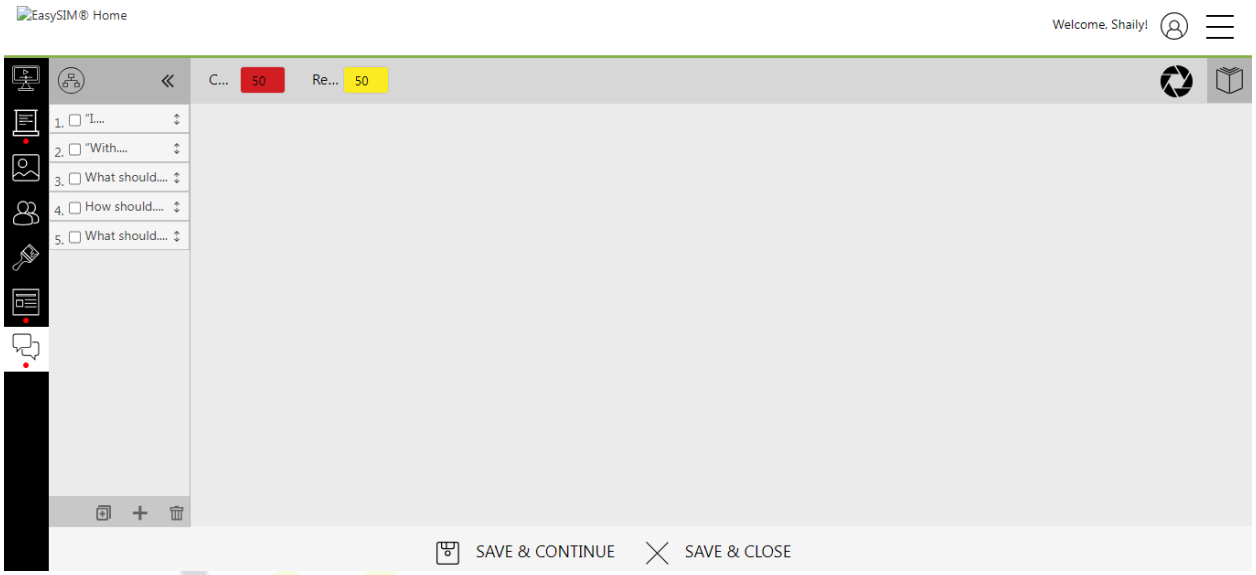
$swa_sql = "SELECT a., f.* FROM answer_tbl a LEFT JOIN feedback_tbl f ON f.answer_id = a.answer_id WHERE a.question_id = ?";*
This SQL query retrieves data from two tables, answer_tbl and feedback_tbl, utilizing a LEFT JOIN operation. This query aims to retrieve information about answers to a particular question along with any associated feedback.

JS Functions Used

1. **$(".Add_Drag").click(function(){})**
   This JavaScript code is part of a application where users can create questions with various features like multiple-choice questions (MCQs) and drag-and-drop (DND) questions.
2. **$(".MCQAdd").click(function(){})**
   This code dynamically adds new MCQ options when a certain button (MCQAdd) is clicked. It ensures that each new MCQ option is properly initialized with a text editor for text input and provides functionality to remove the MCQ option if needed.
3. **$(".swipAdd").click(function(){})**
   This code allows users to dynamically add form elements and upload files, with additional functionality for rich text editing and text-to-speech conversion.

Edit Simulation > Linear Multiple

Template Name: linear-multiple-template.php



PHP Functions Used

1. **checkLogin()**
   Global function from "db.class.php". The checkLogin function ensures that a user is properly logged in by verifying that all essential session variables are set and not empty.
2. **getScenario($sim_id)**
   Global function from "db.class.php". This PHP function, "getScenario", retrieves scenario data from the database based on the provided scenario ID ($sim_id).
3. **getScenarioCompetency($sid)**
   Global function from "db.class.php". This function, "getScenarioCompetency", retrieves competency information associated with a specific scenario from the competency_tbl database table.
4. **getSimPage($sid)**
   Global function from "db.class.php". This PHP function, "getSimPage", retrieves simulation pages associated with a specific scenario ID from the sim_pages table in the database.
5. **getSimBranding($sid)**
   Global function from "db.class.php". This function, "getSimBranding", retrieves branding information associated with a specific scenario from the sim_branding_tbl database table.
6. **getMultiScenarioType($id)**
   Global function from "db.class.php". The getMultiScenarioType function retrieves data from a database based on the provided scenario type IDs.
7. **SQL Queries**
   *$qsql = "SELECT * FROM question_tbl WHERE question_id = ?";*
   This query retrieves all columns from the question_tbl table for a specific question_id.

*$feedsql = "SELECT * FROM feedback_tbl WHERE question_id = :question_id";*
This SQL query retrieves all columns from the feedback_tbl table where the question_id column matches a specific value.

$tqsql = "SELECT SUM(ques_val_1) AS ques1, SUM(ques_val_2) AS ques2, SUM(ques_val_3) AS ques3, SUM(ques_val_4) AS ques4, SUM(ques_val_5) AS ques5, SUM(ques_val_6) AS ques6 FROM question_tbl WHERE scenario_id = ?";
This SQL query retrieves the sum of values in six different columns (ques_val_1 to ques_val_6) from the question_tbl table for rows where the scenario_id matches a specific value. Each sum is given an alias (ques1 to ques6) for easier reference in the output. The actual value for scenario_id will be supplied at runtime when the query is executed.

$mtf_choice_sql = "SELECT answer_id, choice_option, choice_order_no, match_order_no, match_option FROM answer_tbl WHERE question_id = ? AND choice_order_no != '0' AND match_order_no != '0'";
This SQL query selects specific columns from a table named answer_tbl, filtering the rows based on conditions related to question_id, choice_order_no, and match_order_no.

$mtf_choice_sql1 = "SELECT answer_id, choice_option, match_option FROM answer_tbl WHERE question_id = ? AND choice_order_no = '0' AND match_order_no = '0'";
This SQL query selects specific columns from a table named answer_tbl, filtering the rows based on conditions related to question_id, choice_order_no, and match_order_no. This query seems to target rows where both choice_order_no and match_order_no are equal to zero.

$sq_sql = "SELECT answer_id, choice_option FROM answer_tbl WHERE question_id = ?";
This SQL query retrieves the answer_id and choice_option columns from the answer_tbl table where the question_id matches a specific value.

$sort_sql = "SELECT answer_id, choice_option, match_option, match_sorting_item, image FROM answer_tbl WHERE question_id = ?";
This SQL query selects specific columns from a table named answer_tbl, filtering the rows based on the question_id column. This query retrieves data related to answers for a specific question identified by its question_id.

$sortop_sql = "SELECT sub_options_id, answer_id, sub_option FROM sub_options_tbl WHERE answer_id = ?";
This SQL query retrieves sub-options related to a specific answer identified by its answer_id. The sub-options are stored in the sub_options_tbl table.

$ddq_sql = "SELECT answer_id, choice_option, match_option, image, image2 FROM answer_tbl WHERE question_id = ?";
This SQL query retrieves answers related to a specific question identified by its question_id. The answers' details include the option choices, match options, and potentially images associated with those answers.

*$ddqop_sql = "SELECT * FROM answer_tbl WHERE question_id = ?";*
This SQL query retrieves all data from the answer_tbl table where the question_id matches a specified value. It's used to fetch answers associated with a particular question identified by its question_id.

$mcq_sql = "SELECT answer_id, choice_option FROM answer_tbl WHERE question_id = ?";
This SQL query retrieves the answer_id and choice_option columns from the answer_tbl table for rows where the question_id matches a specified value. It's used to fetch specific information about answers associated with a particular question identified by its question_id.

$mmcq_sql = "SELECT answer_id, choice_option FROM answer_tbl WHERE question_id = ?";
The query retrieves the answer_id and choice_option columns from the answer_tbl table for rows where the question_id matches a specific value.

$swa_sql = "SELECT answer_id, choice_option, image FROM answer_tbl WHERE question_id = ?";
This SQL query retrieves the answer_id, choice_option, and image columns from the answer_tbl table for rows where the question_id matches a specified value. It's used to fetch specific information about answers associated with a particular question identified by its question_id, including any images linked to those answers.

JS Functions Used

1. **$('.qtype').on('click', function(){})**
   Clicking on any element with the class qtype would set the value of the hidden input field with the ID question_type to the value of its data-qtype attribute. The get_question_type input field can be used to retain the original question type value if needed.
2. **$('.mtf_shuffle').on('click', function(){})**
   This code seems is a part of a functionality related to managing shuffling options for a Match The Following type of question in a form. Depending on the value of the clicked element, it toggles the required attribute for certain elements accordingly, presumably to enforce form validation rules.
3. **$(".AddAnsICON").click(function(){})**
   This code dynamically adds new MTF options to a form when a certain button

(AddAnsICON) is clicked. It ensures that the newly added options are properly initialized, including form validation and rich text editing functionality.

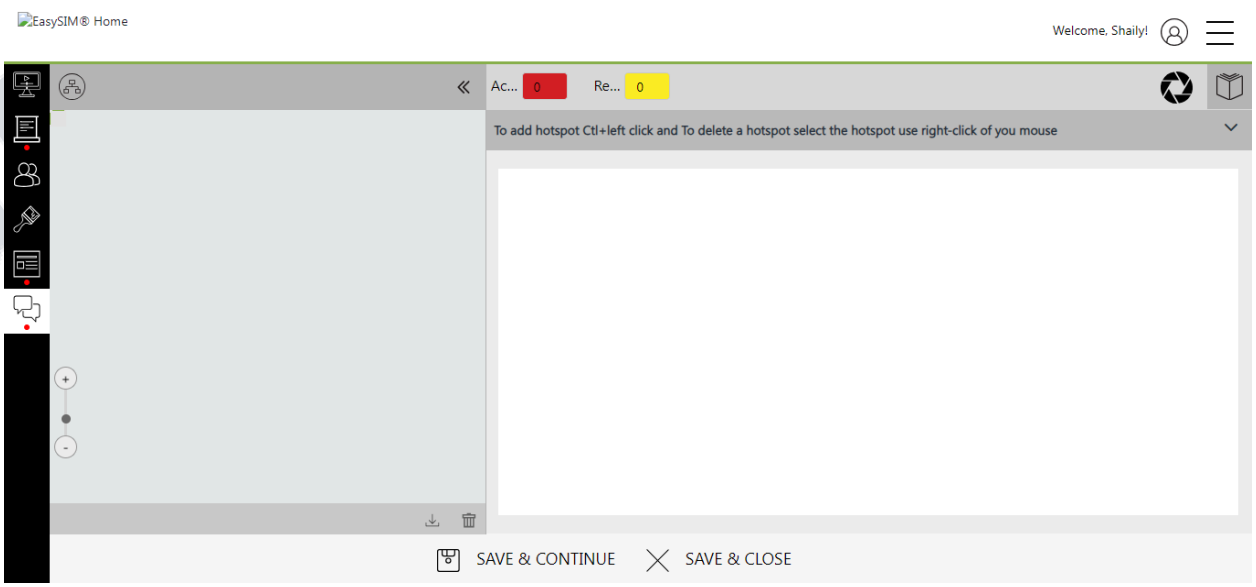
4. **$(".AddAnsICON1").click(function(){})**
   This code dynamically adds new MTF options to a form when a certain button (AddAnsICON1) is clicked. It ensures that the newly added options are properly initialized, including form validation and rich text editing functionality, similar to the previous snippet. However, these options might have different styling or behavior, as indicated by the different class used for their containers.
5. **$(".DNDAddAnsICON1").click(function(){})**
   This code dynamically adds new options for a Drag and Drop question type to a form when a certain button (DNDAddAnsICON1) is clicked. It ensures that the newly added options are properly initialized, including rich text editing functionality.
6. **$(".DNDAddAnsICON").click(function(){})**
   This code dynamically adds new drag and drop options to a form when a certain button (DNDAddAnsICON) is clicked. It ensures that the newly added options are properly initialized, including rich text editing functionality.
7. **$(".SeqAdd").click(function(){})**
   This code dynamically adds new sequence answer options to a form when a certain button (SeqAdd) is clicked. It ensures that the newly added options are properly initialized, including rich text editing functionality.
8. **$(".ShortAddICON").click(function(){})**
   This code dynamically adds new options for a short answer question with drag-and-drop functionality when a certain button (ShortAddICON) is clicked. It ensures that the newly added options are properly initialized, including rich text editing functionality, and provides functionality to add and remove drag options dynamically.
9. **$('body').on('click', '.ShortDragAddICON', function() {})**
   This code dynamically adds new drag options within a specific sorting item for a short answer question when a certain button (ShortDragAddICON) is clicked. It ensures that the new drag option is properly initialized with rich text editing functionality and provides functionality to remove the drag option if needed.
10. **$(".MCQAdd").click(function(){})**
    This code dynamically adds new MCQ options when a certain button (MCQAdd) is clicked. It ensures that each new MCQ option is properly initialized with a text editor for text input and provides functionality to remove the MCQ option if needed.
11. **$(".MMCQAdd").click(function(){})**
    This function dynamically adds a new set of checkbox and textarea elements for multiple choice questions (MMCQ) each time the .MMCQAdd element is clicked, and initializes a rich text editor for the textarea element.
12. **$(".SwaAdd").click(function(){})**
    This function dynamically adds a new set of radio button, textarea, and image upload elements for single word answer questions (SWA) each time the .SwaAdd element is clicked, and initializes a rich text editor for the textarea element.
13. **$('body').on('click', '.removeAnswer', function(e) {})**
    This code snippet is a jQuery-based script for handling the click event on elements with

the class .removeAnswer. It includes a confirmation dialog, an AJAX request at *"includes/process.php"* to delete an answer, and error handling

14. **$('body').on('click', '.removeSortingOption', function(e) {})**
    This code provides a user-friendly way to delete sorting options and handles the deletion process asynchronously without refreshing the page. Ajax request is sent to the server using the $.ajax method. The request is sent to the URL specified (*includes/process.php*) with the data string containing the information about the sorting answer to be deleted.
15. **$('body').on('click', '.removeSortingSubOption', function(e) {})**
    This code provides a user-friendly way to delete sorting sub-options and handles the deletion process asynchronously without refreshing the page. Ajax request is sent to the server using the $.ajax method. The request is sent to the URL specified (*includes/process.php*) with the data string containing the information about the sorting sub-answer to be deleted.

Edit Simulation > 360 Branching

Template Name: 360-branching-template.php



PHP Functions Used

1. **checkLogin()**
   Global function from "db.class.php". The checkLogin function ensures that a user is properly logged in by verifying that all essential session variables are set and not empty.
2. **getScenario($sim_id)**
   Global function from "db.class.php". This PHP function, "getScenario", retrieves scenario data from the database based on the provided scenario ID ($sim_id).
3. **getScenarioCompetency($sid)**
   Global function from "db.class.php". This function, "getScenarioCompetency", retrieves

competency information associated with a specific scenario from the competency_tbl database table.

4. **getSimPage($sid)**
   Global function from “db.class.php”. This PHP function, "getSimPage", retrieves simulation pages associated with a specific scenario ID from the sim_pages table in the database.
5. **getSimBranding($sid)**
   Global function from “db.class.php”. This function, “getSimBranding”, retrieves branding information associated with a specific scenario from the sim_branding_tbl database table.
6. **getMultiScenarioType($id)**
   Global function from “db.class.php”. The getMultiScenarioType function retrieves data from a database based on the provided scenario type IDs.
7. **SQL Queries**
   *$qsql = "SELECT * FROM question_tbl WHERE question_id = ?";*
   This query retrieves all columns from the question_tbl table for a specific question_id.

   *$tqsql = "SELECT * FROM question_tbl WHERE scenario_id = ?";*
   This SQL query retrieves all rows from the question_tbl table where the value of the scenario_id column matches the value provided for the placeholder ?.

   $questionListSql = "SELECT question_id, questions AS ques FROM question_tbl WHERE scenario_id = ? AND question_id != ?";
   This SQL query selects specific data from the question_tbl table in a database based on the scenario_id and question_id passed into it.

   $mcq_sql = "SELECT a., f.* FROM answer_tbl a LEFT JOIN feedback_tbl f ON f.answer_id = a.answer_id WHERE a.question_id = ?";*
   This SQL query string designed to select data from two tables, answer_tbl and feedback_tbl, using a LEFT JOIN as per the question_id passed in WHERE clause.

   $mappedListSql = "SELECT End_Sim, nextQid FROM answer_tbl WHERE question_id = ? AND answer_id = ?";
   This SQL query selects specific data from the answer_tbl table based on given question_id and answer_id values.

   *$ddq2_sql = "SELECT * FROM sub_options_tbl WHERE question_id = ?";*
   This SQL query selects data from the sub_options_tbl table based on a specific question_id.

   $ddqop_sql = "SELECT a., f.* FROM answer_tbl a LEFT JOIN feedback_tbl f ON a.answer_id = f.answer_id WHERE a.question_id = ?";*
   This SQL query selects data from two tables, answer_tbl and feedback_tbl, using a LEFT JOIN. This query retrieves all answers for a specific question along with any

associated feedback.

$swa_sql = "SELECT a., f.* FROM answer_tbl a LEFT JOIN feedback_tbl f ON f.answer_id = a.answer_id WHERE a.question_id = ?";*
This SQL query retrieves data from two tables, answer_tbl and feedback_tbl, utilizing a LEFT JOIN operation. This query aims to retrieve information about answers to a particular question along with any associated feedback.

$mappedListSql = "SELECT End_Sim, nextQid FROM answer_tbl WHERE question_id = ? AND answer_id = ?";
This SQL query selects specific data from the answer_tbl table based on given question_id and answer_id values.

Third Party JS Library

1. InputMask
2. Spectrum
3. Fancybox
4. GoJS
5. HyperlinkText
6. ZoomSlider
7. Polyfill
8. JSPdf
9. HTML2Canvas

JS Functions Used

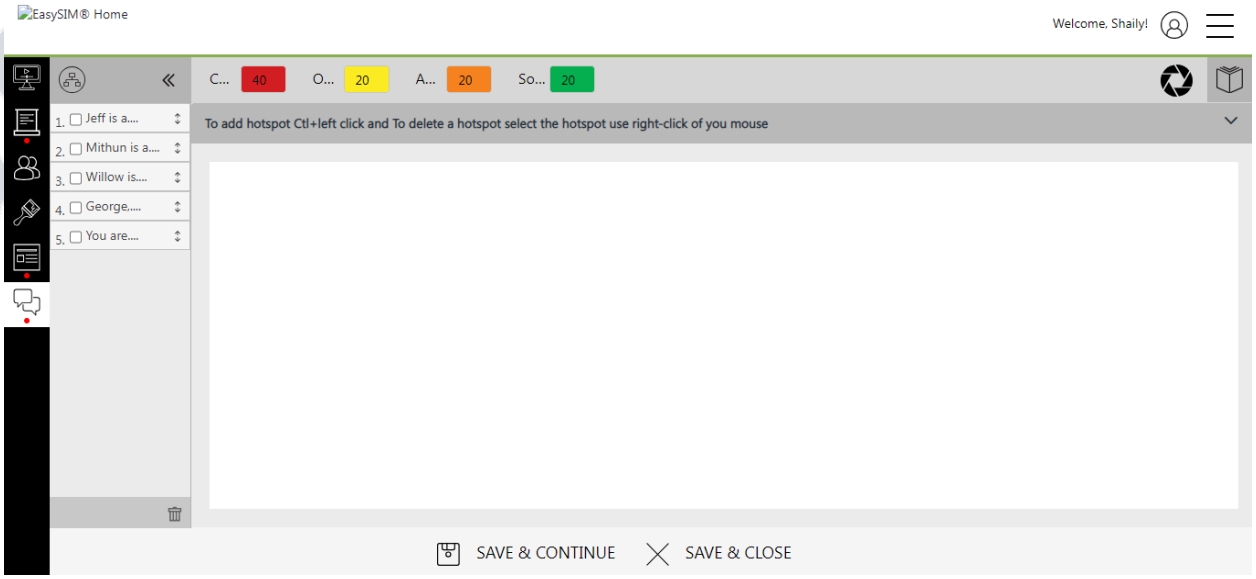
1. **getPDF(id){}**
   This JavaScript function, getPDF(id), converts an HTML element specified by its ID into a PDF document.
2. **$('.DelIcon360').on('click', function(e){})**
   This script handles the click event on elements with the .DelIcon360 class. When clicked, it retrieves the question ID from an input field. If a question ID is provided, it sends an AJAX POST request to delete the question, shows a loading overlay, and handles the server response. If the deletion is successful, it redirects the user to a new location after a brief delay.
3. **$('#upload_360_file').click(function(){})**
   This script handles the click event on the element with the ID upload_360_file. When clicked, it retrieves data attributes and file information from the file input element, shows a loading overlay, and sends an AJAX POST request at *"includes/upload.php"* to upload the file. If the upload is successful, it updates the UI with the response data and disables certain elements. If the upload fails, it shows appropriate error or warning messages.
4. **$('.delete_360_assets').click(function() {})**
   This script handles the click event on elements with the .delete_360_assets class. When

clicked, it retrieves data attributes from the clicked element, shows a confirmation dialog, and sends an AJAX POST request at *"includes/delete-file.php"* to delete the specified file if confirmed. It provides visual feedback using a loading overlay and SweetAlert dialogs, and updates the UI based on the server's response.

5. **$(".MCQadd").click(function(){})**
   This function adds a new MCQ form dynamically when a specific element is clicked, and it also initializes jQuery Text Editors on the text areas within the newly added MCQ form.
6. **$(".Add_Drag").click(function(){})**
   This JavaScript code is part of a application where users can create questions with various features like multiple-choice questions (MCQs) and drag-and-drop (DND) questions.
7. **$(".swipAdd").click(function(){})**
   This code allows users to dynamically add form elements and upload files, with additional functionality for rich text editing and text-to-speech conversion.

Edit Simulation > 360

Template Name: 360-template.php



PHP Functions Used

1. **checkLogin()**
   Global function from "db.class.php". The checkLogin function ensures that a user is properly logged in by verifying that all essential session variables are set and not empty.
2. **getScenario($sim_id)**
   Global function from "db.class.php". This PHP function, "getScenario", retrieves scenario data from the database based on the provided scenario ID ($sim_id).
3. **getScenarioCompetency($sid)**
   Global function from "db.class.php". This function, "getScenarioCompetency", retrieves

competency information associated with a specific scenario from the competency_tbl database table.

4. **getSimPage($sid)**
   Global function from “db.class.php”. This PHP function, "getSimPage", retrieves simulation pages associated with a specific scenario ID from the sim_pages table in the database.
5. **getSimBranding($sid)**
   Global function from “db.class.php”. This function, “getSimBranding”, retrieves branding information associated with a specific scenario from the sim_branding_tbl database table.
6. **getMultiScenarioType($id)**
   Global function from “db.class.php”. The getMultiScenarioType function retrieves data from a database based on the provided scenario type IDs.
7. **SQL Queries**
   *$qsql = "SELECT * FROM question_tbl WHERE question_id = ?";*
   This query retrieves all columns from the question_tbl table for a specific question_id.

   $tqsql = "SELECT SUM(ques_val_1) AS ques1, SUM(ques_val_2) AS ques2, SUM(ques_val_3) AS ques3, SUM(ques_val_4) AS ques4, SUM(ques_val_5) AS ques5, SUM(ques_val_6) AS ques6 FROM question_tbl WHERE scenario_id = ?";
   This SQL query retrieves the sum of values in six different columns (ques_val_1 to ques_val_6) from the question_tbl table for rows where the scenario_id matches a specific value. Each sum is given an alias (ques1 to ques6) for easier reference in the output. The actual value for scenario_id will be supplied at runtime when the query is executed.

   $mcq_sql = "SELECT a., f.* FROM answer_tbl a LEFT JOIN feedback_tbl f ON f.answer_id = a.answer_id WHERE a.question_id = ?";*
   This SQL query string designed to select data from two tables, answer_tbl and feedback_tbl, using a LEFT JOIN as per the question_id passed in WHERE clause.

   *$ddq2_sql = "SELECT * FROM sub_options_tbl WHERE question_id = ?";*
   This SQL query selects data from the sub_options_tbl table based on a specific question_id.

   $ddqop_sql = "SELECT a., f.* FROM answer_tbl a LEFT JOIN feedback_tbl f ON a.answer_id = f.answer_id WHERE a.question_id = ?";*
   This SQL query selects data from two tables, answer_tbl and feedback_tbl, using a LEFT JOIN. This query retrieves all answers for a specific question along with any associated feedback.

   $swa_sql = "SELECT a., f.* FROM answer_tbl a LEFT JOIN feedback_tbl f ON f.answer_id = a.answer_id WHERE a.question_id = ?";*
   This SQL query retrieves data from two tables, answer_tbl and feedback_tbl, utilizing a

LEFT JOIN operation. This query aims to retrieve information about answers to a particular question along with any associated feedback.

Third Party JS Library

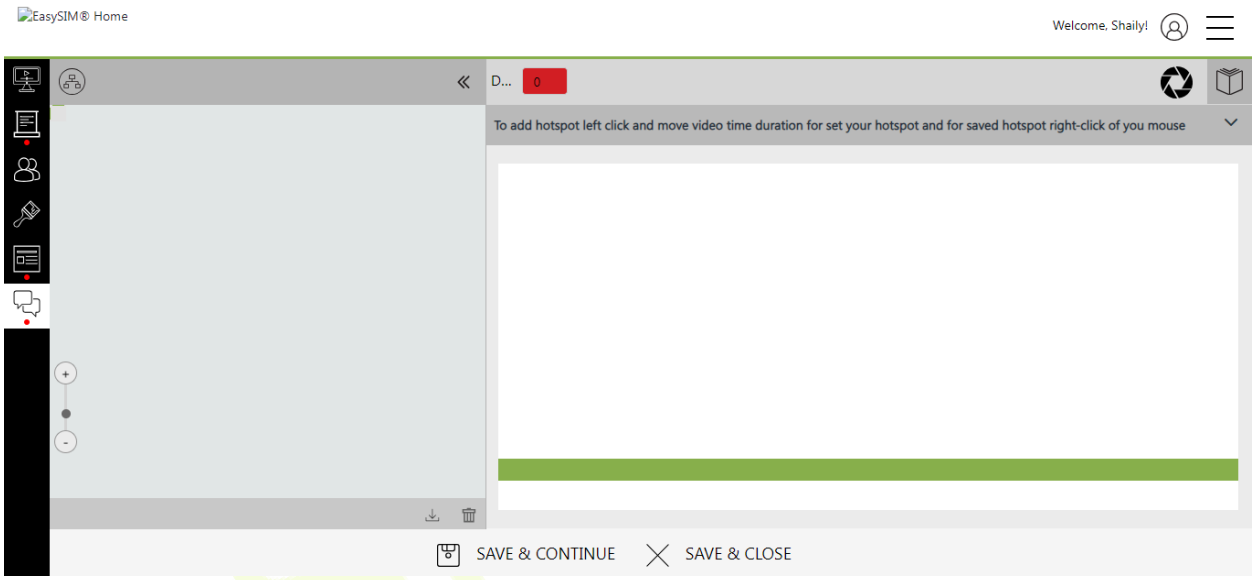
1. InputMask
2. Spectrum
3. Fancybox
4. JQueryUI

JS Functions Used

1. **$('.DelIcon360').on('click', function(e){})**
   This script handles the click event on elements with the .DelIcon360 class. When clicked, it retrieves the IDs of checked questions, shows a loading overlay, and sends an AJAX POST request at *'includes/process.php'* to delete the selected questions. It provides visual feedback using a loading overlay and SweetAlert dialogs.
2. **$('#upload_360_file').click(function(){})**
   This script handles the click event on the element with the ID upload_360_file. When clicked, it retrieves data attributes and file information from the file input element, shows a loading overlay, and sends an AJAX POST request at *"includes/upload.php"* to upload the file. If the upload is successful, it updates the UI with the response data and disables certain elements.
3. **$('.delete_360_assets').click(function() {})**
   This script handles the click event on elements with the .delete_360_assets class. When clicked, it retrieves data attributes from the clicked element, shows a confirmation dialog using SweetAlert, and, if confirmed, sends an AJAX POST request at *"includes/delete-file.php"* to delete the specified file. It provides visual feedback using a loading overlay and SweetAlert dialogs.
4. **$(".MCQadd").click(function(){})**
   This function adds a new MCQ form dynamically when a specific element is clicked, and it also initializes jQuery Text Editors on the text areas within the newly added MCQ form.
5. **$(".Add_Drag").click(function(){})**
   This JavaScript code is part of a application where users can create questions with various features like multiple-choice questions (MCQs) and drag-and-drop (DND) questions.
6. **$(".swipAdd").click(function(){})**
   This code allows users to dynamically add form elements and upload files, with additional functionality for rich text editing and text-to-speech conversion.

Edit Simulation > 360 Video Branching

Template Name: 360-video-branching-template.php



PHP Functions Used

1. **checkLogin()**
   Global function from “db.class.php”. The checkLogin function ensures that a user is properly logged in by verifying that all essential session variables are set and not empty.
2. **getScenario($sim_id)**
   Global function from “db.class.php”. This PHP function, "getScenario", retrieves scenario data from the database based on the provided scenario ID ($sim_id).
3. **getScenarioCompetency($sid)**
   Global function from “db.class.php”. This function, "getScenarioCompetency", retrieves competency information associated with a specific scenario from the competency_tbl database table.
4. **getSimPage($sid)**
   Global function from “db.class.php”. This PHP function, "getSimPage", retrieves simulation pages associated with a specific scenario ID from the sim_pages table in the database.
5. **getSimBranding($sid)**
   Global function from “db.class.php”. This function, “getSimBranding”, retrieves branding information associated with a specific scenario from the sim_branding_tbl database table.
6. **getMultiScenarioType($id)**
   Global function from “db.class.php”. The getMultiScenarioType function retrieves data from a database based on the provided scenario type IDs.
7. **SQL Queries**
   *$qsql = "SELECT * FROM question_tbl WHERE question_id = ?";*
   This query retrieves all columns from the question_tbl table for a specific question_id.

   *$tqsql = "SELECT * FROM question_tbl WHERE scenario_id = ?";*
   This SQL query retrieves all rows from the question_tbl table where the value of the

scenario_id column matches the value provided for the placeholder ?.

$feedSql = "SELECT question_id, nextQid FROM answer_tbl WHERE question_id = ?";
This query selects specific data from the answer_tbl table in a database based on the question_id passed.

$questionListSql = "SELECT question_id, questions AS ques FROM question_tbl WHERE scenario_id = ? AND question_id != ?";
This SQL query selects question_id and questions (aliased as ques) from a table named question_tbl, filtering the rows based on specific conditions related to scenario_id and question_id.

$mcq_sql = "SELECT a., f.* FROM answer_tbl a LEFT JOIN feedback_tbl f ON f.answer_id = a.answer_id WHERE a.question_id = ?";*
This SQL query string designed to select data from two tables, answer_tbl and feedback_tbl, using a LEFT JOIN as per the question_id passed in WHERE clause.

$mappedListSql = "SELECT End_Sim, nextQid FROM answer_tbl WHERE question_id = ? AND answer_id = ?";
This SQL query selects specific data from the answer_tbl table based on given question_id and answer_id values.

*$ddq2_sql = "SELECT * FROM sub_options_tbl WHERE question_id = ?";*
This SQL query selects data from the sub_options_tbl table based on a specific question_id.

$ddqop_sql = "SELECT a., f.* FROM answer_tbl a LEFT JOIN feedback_tbl f ON a.answer_id = f.answer_id WHERE a.question_id = ?";*
This SQL query selects data from two tables, answer_tbl and feedback_tbl, using a LEFT JOIN. This query retrieves all answers for a specific question along with any associated feedback.

$swa_sql = "SELECT a., f.* FROM answer_tbl a LEFT JOIN feedback_tbl f ON f.answer_id = a.answer_id WHERE a.question_id = ?";*
This SQL query retrieves data from two tables, answer_tbl and feedback_tbl, utilizing a LEFT JOIN operation. This query aims to retrieve information about answers to a particular question along with any associated feedback.

Third Party JS Library

1. InputMask
2. Spectrum
3. Fancybox

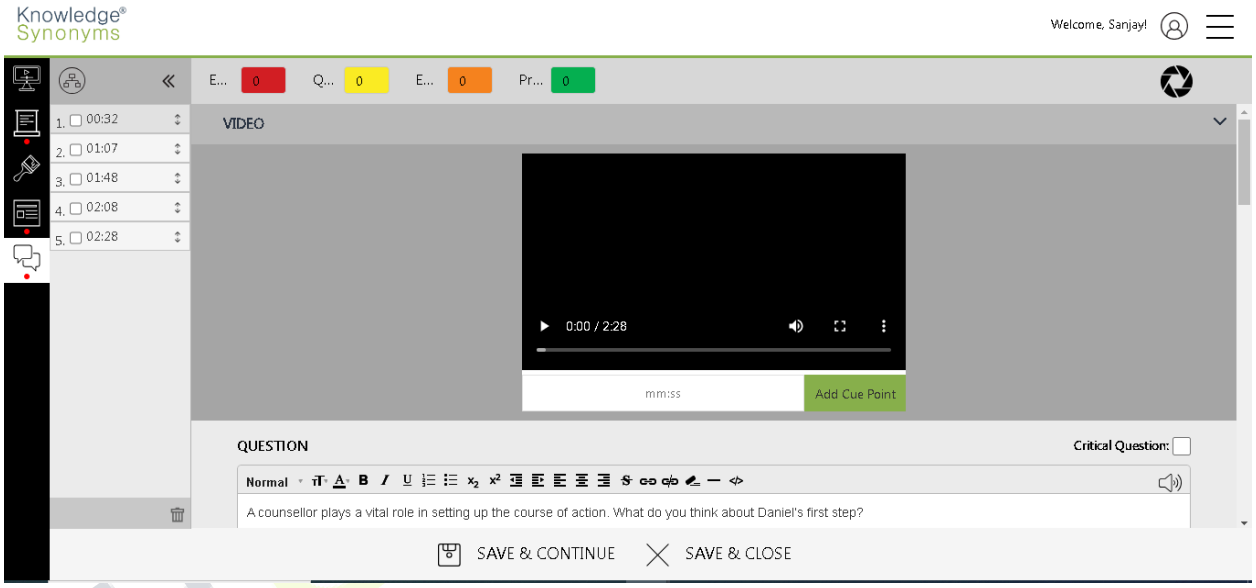
4. GoJS
5. HyperlinkText
6. ZoomSlider
7. Polyfill
8. JQueryUI
9. JSPdf
10. HTML2Canvas

JS Functions Used

1. **getPDF(id){}**
   This JavaScript function, getPDF(id), converts an HTML element specified by its ID into a PDF document.
2. **$('.DelIcon360').on('click', function(e){})**
   This script handles the click event on elements with the .DelIcon360 class. When clicked, it retrieves the IDs of checked questions, shows a loading overlay, and sends an AJAX POST request at *'includes/process.php'* to delete the selected questions. It provides visual feedback using a loading overlay and SweetAlert dialogs.
3. **$('#upload_360_file').click(function(){})**
   This script handles the click event on the element with the ID upload_360_file. When clicked, it retrieves data attributes and file information from the file input element, shows a loading overlay, and sends an AJAX POST request at *"includes/upload.php"* to upload the file. If the upload is successful, it updates the UI with the response data and disables certain elements.
4. **$('.delete_360_assets').click(function() {})**
   This script handles the click event on elements with the .delete_360_assets class. When clicked, it retrieves data attributes from the clicked element, shows a confirmation dialog using SweetAlert, and, if confirmed, sends an AJAX POST request at *"includes/delete-file.php"* to delete the specified file. It provides visual feedback using a loading overlay and SweetAlert dialogs.
5. **$(".MCQadd").click(function(){})**
   This function adds a new MCQ form dynamically when a specific element is clicked, and it also initializes jQuery Text Editors on the text areas within the newly added MCQ form.
6. **$(".Add_Drag").click(function(){})**
   This JavaScript code is part of a application where users can create questions with various features like multiple-choice questions (MCQs) and drag-and-drop (DND) questions.
7. **$(".swipAdd").click(function(){})**
   This code allows users to dynamically add form elements and upload files, with additional functionality for rich text editing and text-to-speech conversion.

Edit Simulation > Single Video

Template Name: single-video-template.php



PHP Functions Used

1. **checkLogin()**
   Global function from “db.class.php”. The checkLogin function ensures that a user is properly logged in by verifying that all essential session variables are set and not empty.
2. **getScenario($sim_id)**
   Global function from “db.class.php”. This PHP function, "getScenario", retrieves scenario data from the database based on the provided scenario ID ($sim_id).
3. **getScenarioCompetency($sid)**
   Global function from “db.class.php”. This function, "getScenarioCompetency", retrieves competency information associated with a specific scenario from the competency_tbl database table.
4. **getSimPage($sid)**
   Global function from “db.class.php”. This PHP function, "getSimPage", retrieves simulation pages associated with a specific scenario ID from the sim_pages table in the database.
5. **getSimBranding($sid)**
   Global function from “db.class.php”. This function, “getSimBranding”, retrieves branding information associated with a specific scenario from the sim_branding_tbl database table.
6. **getMultiScenarioType($id)**
   Global function from “db.class.php”. The getMultiScenarioType function retrieves data from a database based on the provided scenario type IDs.
7. **SQL Queries**
   *$qsql = "SELECT * FROM question_tbl WHERE question_id = ?";*
   This query retrieves all columns from the question_tbl table for a specific question_id.

$tqsql = "SELECT SUM(ques_val_1) AS ques1, SUM(ques_val_2) AS ques2, SUM(ques_val_3) AS ques3, SUM(ques_val_4) AS ques4, SUM(ques_val_5) AS ques5, SUM(ques_val_6) AS ques6 FROM question_tbl WHERE scenario_id = ?";
This SQL query retrieves the sum of values in six different columns (ques_val_1 to ques_val_6) from the question_tbl table for rows where the scenario_id matches a specific value. Each sum is given an alias (ques1 to ques6) for easier reference in the output. The actual value for scenario_id will be supplied at runtime when the query is executed.

Third Party JS Library

1. InputMask
2. Spectrum
3. Fancybox
4. JQueryUI

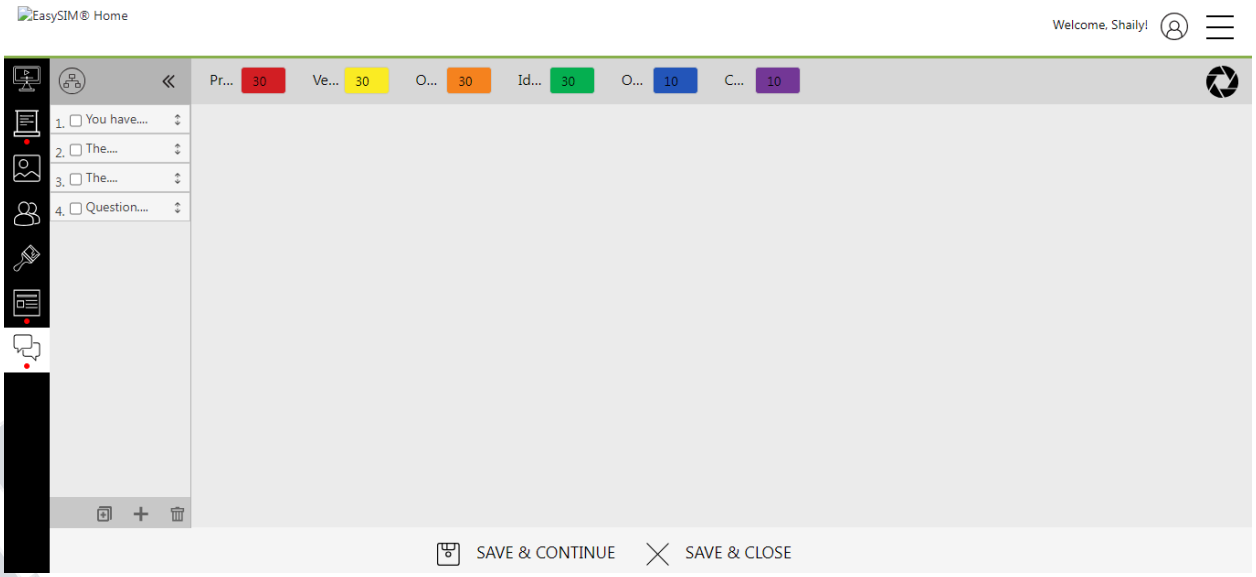
JS Functions Used

1. **$('#upload_videoq_file').click(function() {})**
   This script handles the click event on the element with the ID upload_videoq_file. When clicked, it retrieves data attributes and file information from the file input element, shows a loading overlay, and sends an AJAX POST request at *"includes/upload.php"* to upload the file. If the upload is successful, it updates the UI with the response data, sets the new video source, and reloads the video. If the upload fails, it shows appropriate error or warning messages.
2. **$('body').on('click', '.delete_video_assets', function() {})**
   This code handles the deletion of video assets from the UI with a confirmation step and provides feedback to the user on success or failure. If the user confirms the deletion, it sends an AJAX request at *"includes/delete-file.php"* to the server to delete the file.
3. **$('body').on('click', '.delete_tts', function() {})**
   This JavaScript code is another event handler attached to the click event of elements with the class "delete_tts" that are children of the "body" element. When such an element is clicked, it displays a confirmation dialog using the swal() function from the SweetAlert library.
4. **$('body').on('click', '.videocueAdd', function(){})**
   This code handles the addition of video cue points, validating the input, and sending the data to the server via an AJAX request. It uses SweetAlert for user notifications and a loading overlay during the AJAX request at *'includes/process.php'* to enhance the user experience.
5. **$('body').on('click', '.close_cue', function(){})**
   This code handles the deletion of a cue point from the UI with a confirmation step and updates the UI accordingly upon deletion.

6. **$('body').on('click', '.videoAddoptions', function(){})**
   This code dynamically generates a form for adding options for a video question, providing a comprehensive interface for users to input various types of data related to each option.

Edit Simulation > Open Response

Template Name: open-response-template.php



PHP Functions Used

1. **checkLogin()**
   Global function from “db.class.php”. The checkLogin function ensures that a user is properly logged in by verifying that all essential session variables are set and not empty.
2. **getScenario($sim_id)**
   Global function from “db.class.php”. This PHP function, "getScenario", retrieves scenario data from the database based on the provided scenario ID ($sim_id).
3. **getScenarioCompetency($sid)**
   Global function from “db.class.php”. This function, "getScenarioCompetency", retrieves competency information associated with a specific scenario from the competency_tbl database table.
4. **getSimPage($sid)**
   Global function from “db.class.php”. This PHP function, "getSimPage", retrieves simulation pages associated with a specific scenario ID from the sim_pages table in the database.
5. **getSimBranding($sid)**
   Global function from “db.class.php”. This function, “getSimBranding”, retrieves branding information associated with a specific scenario from the sim_branding_tbl database table.

6. **getMultiScenarioType($id)**
   Global function from “db.class.php”. The getMultiScenarioType function retrieves data from a database based on the provided scenario type IDs.
7. **SQL Queries**
   *$qsql = "SELECT * FROM question_tbl WHERE question_id = ?";*
   This query retrieves all columns from the question_tbl table for a specific question_id.

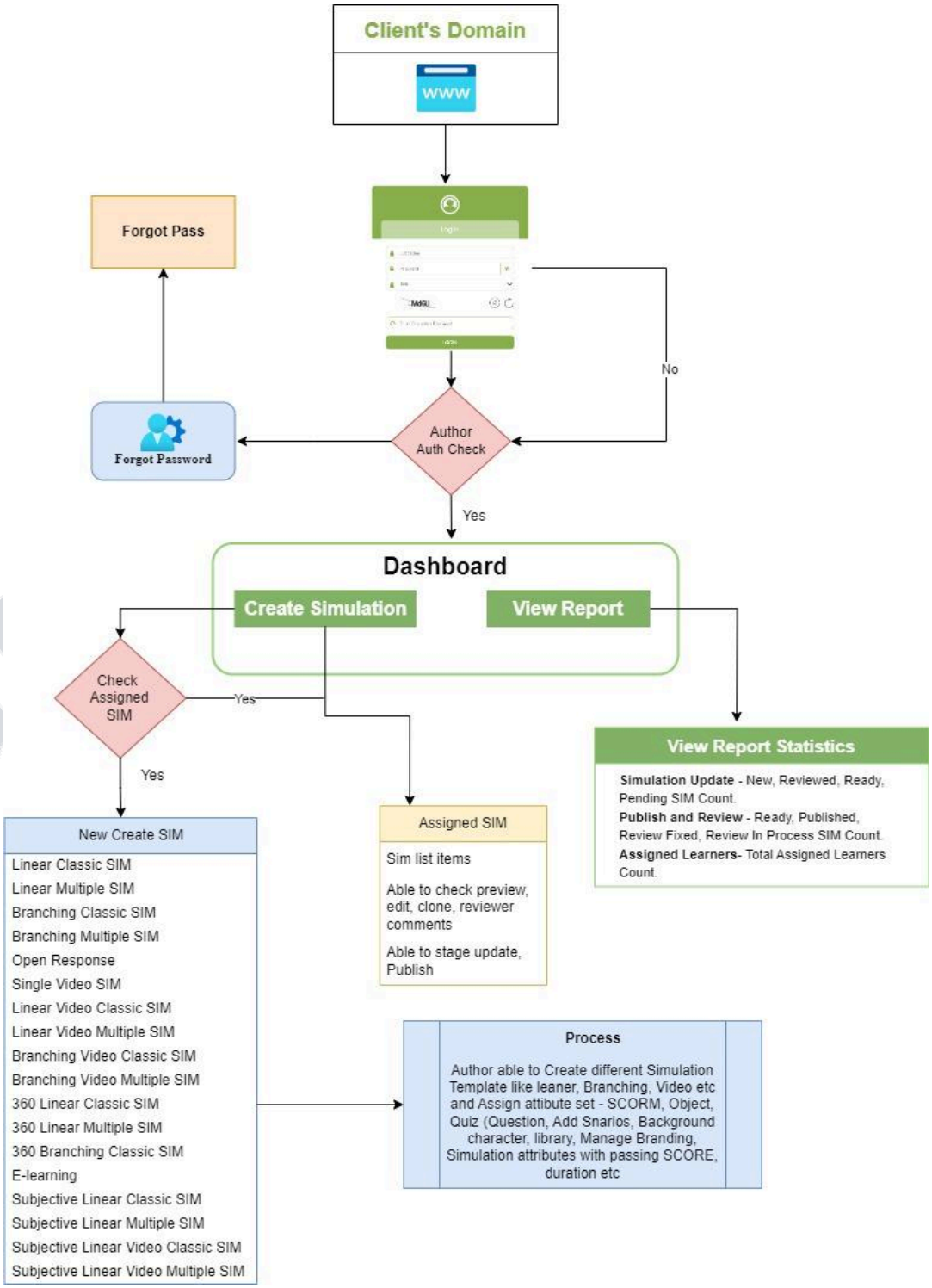
   $tqsql = "SELECT SUM(ques_val_1) AS ques1, SUM(ques_val_2) AS ques2, SUM(ques_val_3) AS ques3, SUM(ques_val_4) AS ques4, SUM(ques_val_5) AS ques5, SUM(ques_val_6) AS ques6 FROM question_tbl WHERE scenario_id = ?";
   This SQL query retrieves the sum of values in six different columns (ques_val_1 to ques_val_6) from the question_tbl table for rows where the scenario_id matches a specific value. Each sum is given an alias (ques1 to ques6) for easier reference in the output. The actual value for scenario_id will be supplied at runtime when the query is executed.

Third Party JS Library

1. InputMask
2. Spectrum
3. Fancybox
4. JQueryUI

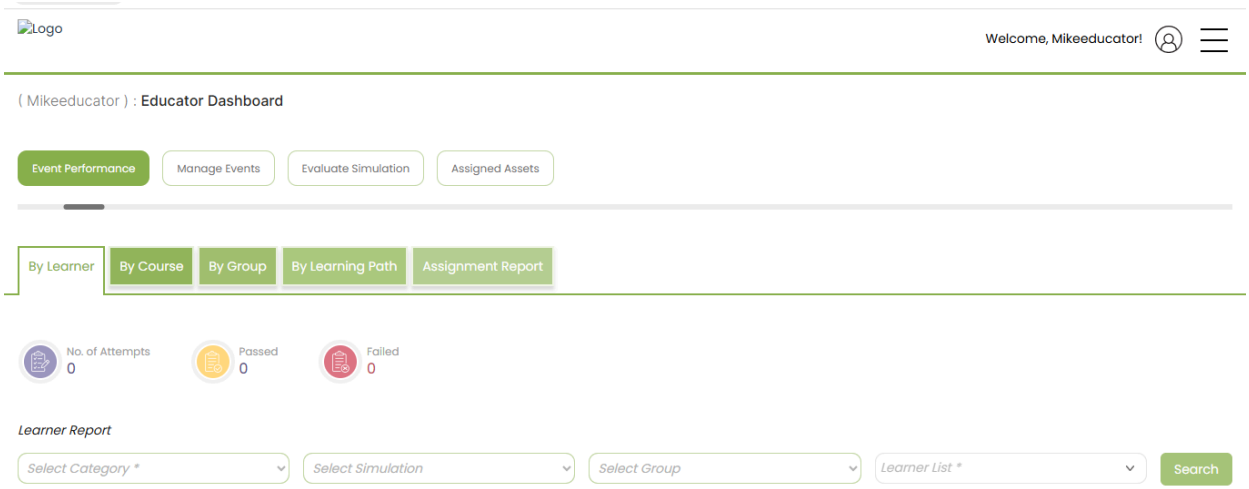
EasyShift®

Author DFD


Client's Domain
www
Forgot Pass
No
Forgot Password
Author
Auth Check
Yes
Dashboard
Create Simulation
View Report
Check
Assigned
SIM
Yes
Yes
View Report Statistics
Simulation Update - New, Reviewed, Ready, Pending SIM Count.
Publish and Review - Ready, Published, Review Fixed, Review In Process SIM Count.
Assigned Learners- Total Assigned Learners Count.
New Create SIM
Linear Classic SIM
Linear Multiple SIM
Branching Classic SIM
Branching Multiple SIM
Open Response
Single Video SIM
Linear Video Classic SIM
Linear Video Multiple SIM
Branching Video Classic SIM
Branching Video Multiple SIM
360 Linear Classic SIM
360 Linear Multiple SIM
360 Branching Classic SIM
E-learning
Subjective Linear Classic SIM
Subjective Linear Multiple SIM
Subjective Linear Video Classic SIM
Subjective Linear Video Multiple SIM
Assigned SIM
Sim list items
Able to check preview, edit, clone, reviewer comments
Able to stage update, Publish
Process
Author able to Create different Simulation Template like leaner, Branching, Video etc and Assign attibute set - SCORM, Object, Quiz (Question, Add Snarios, Background character, library, Manage Branding, Simulation attributes with passing SCORE, duration etc

Educator

Overview



This domain will only be accessible to users with Educator credentials. It will allow management of Event Performance, Manage Events, Evaluate Simulation, and Assigned Assets on the Dashboard page.

1. Event Performance: Contains "By Learner," "By Course," "By Group," "By Learning Path," and "Assignment Report."
    a. By Learner:-It shows the Learner Report here. First, select a category, then select a simulation, select a group, and finally select the learner list. After that, click the "Search" button.
    b. By Course:-Select a category and a simulation, then select a date. After that, choose one option from system wide, group, location, or unit, etc. Then search for the report.
    c. By Group:- Choose an option and then click the search button. It will search based on the selected group.
    d. By Learning Path:-Select a learning path and a simulation, then select a date. After that, choose one option from system wide, group, location, or unit, etc. Then search for the report.
    e. Assignment Report:-The educator can search for an assignment report based on either a simulation or a learning path. The educator can choose one simulation option or a learning path option, select a simulation, and then search for the report.
2. Manage Events: Contains "Events" and "Events Attendance Report."
    a. Events:- Here, you can create, manage, edit, view, and delete events.
    b. Event Attendance Report:- The Events Attendance Report is shown here. Select an option and then search for the report.

3. Evaluate Simulation: Contains "Evaluate Simulation List."
    a. Evaluate Simulation list:- "In this page, the Evaluate Simulation List is displayed."
4. Assigned Assets: Contains "Assigned Assets List."
    a. Assigned Assets List:- "When the admin uploads an asset to them, it is then displayed as the assigned asset on this page."

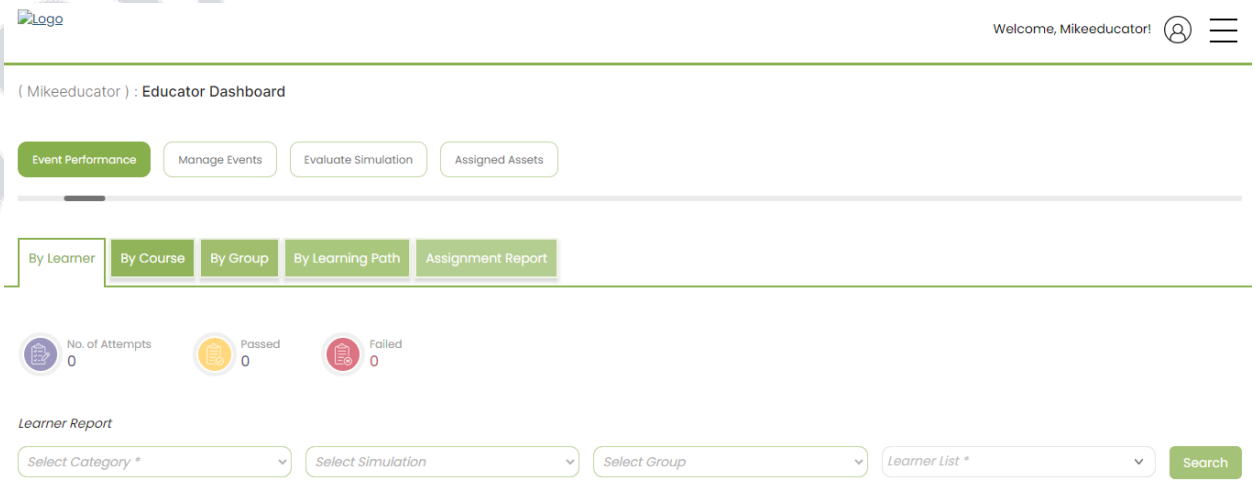
There are 4 tabs in the **Educator**.

I. Event Performance
II. Manage Events
III. Evaluate Simulation
IV. Assigned Assets

Code Stuff

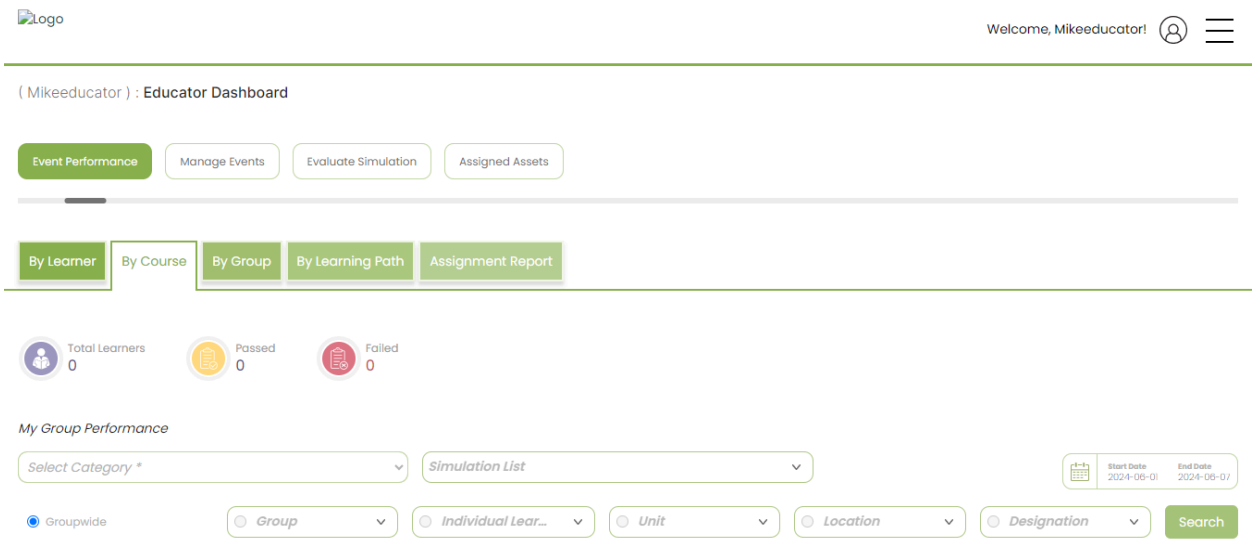
Event Performance > By Learner

Template Name: educator-assigned-assets.php



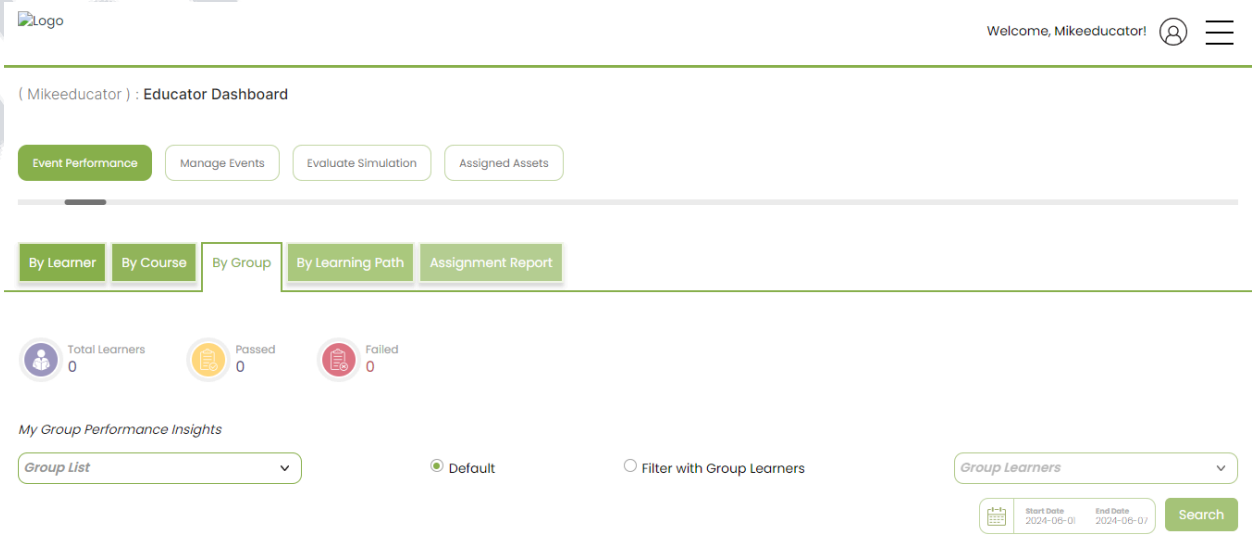
Event Performance > By Course

Template Name: group-performance-educator-new.php



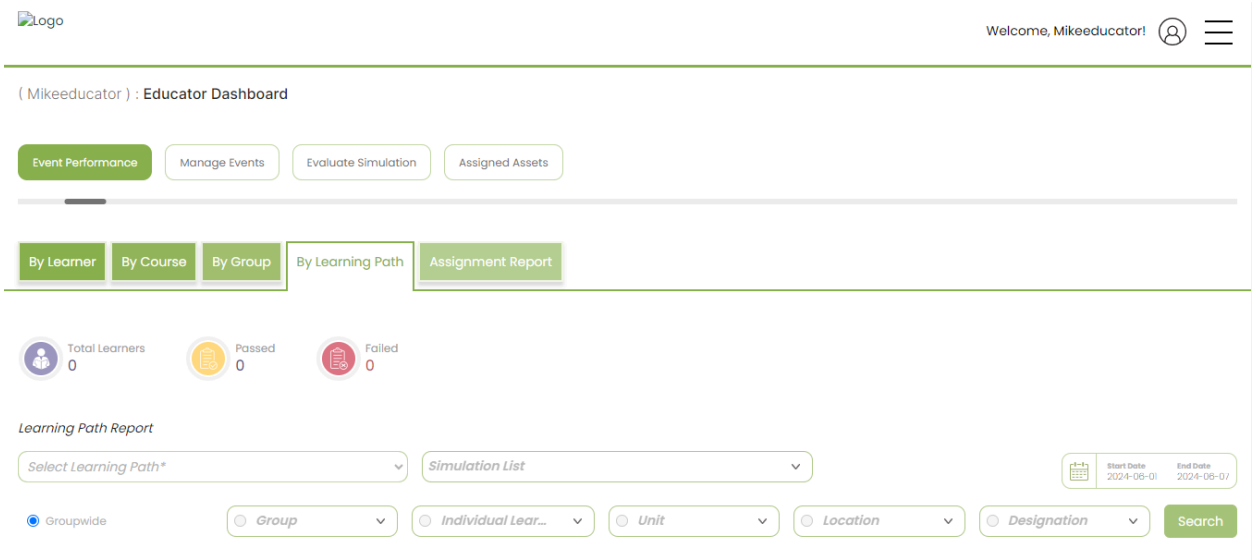
Event Performance > By Group

Template Name: group-performance-insights-educator-report.php



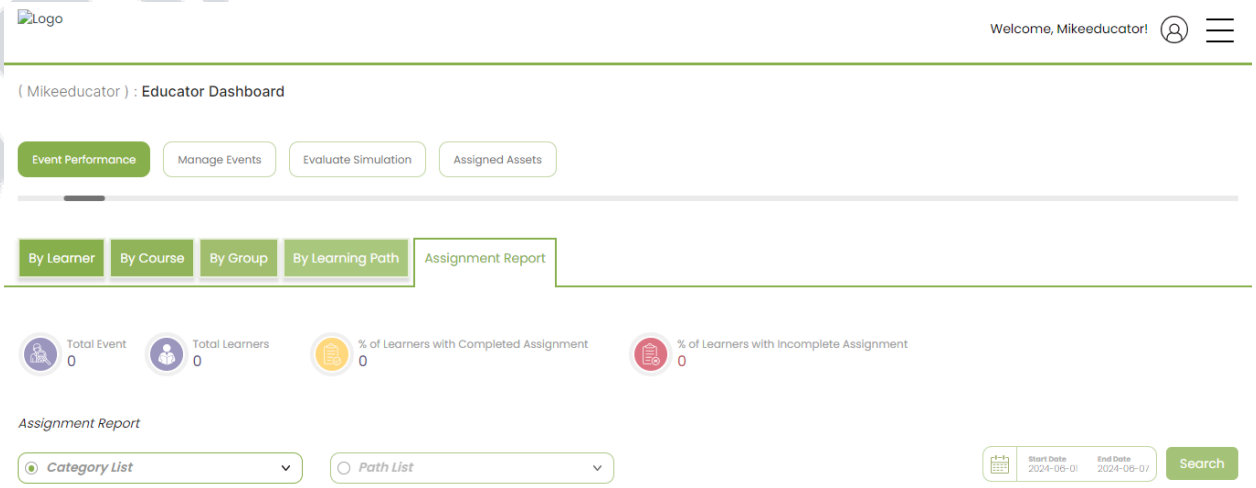
Event Performance > By Learning Path

Template Name: learner-learning-path-report.php



Event Performance > Assignment Report

Template Name: assignment-educator-report.php



Event Performance > Common Functions & Files Used

PHP Functions Used

1. **getAssigneeName($userid)**
   Global function from "db.class.php". This function will return all the data for user as per the User ID passed in the argument.

Third Party JS Library

1. Datatables
2. CanvasJS
3. DateRangePicker

JS File Used

File Path: content/js/educator/educator-overview-dash.js

1. **getPercentage = (x, y, html = false) => {}**
   This JavaScript code defines a function named "getPercentage" that calculates the percentage of x relative to y, optionally returning the result as a formatted string with a percentage sign.
2. **cb(start, end) {}**
   This JavaScript code defines a function named "cb" that updates the values of specific input elements with formatted date strings based on the given start and end dates.
3. **$('#reportrange, #group_report_range').daterangepicker({})**
   This code sets up a DateRangePicker plugin for two elements and configures various options such as initial date range, date format, positioning behavior, predefined ranges, and a callback function for handling date range changes.
4. **CircleBar = () => {}**
   This JavaScript code defines a function named "CircleBar" that dynamically updates the style of elements representing circular progress bars on a webpage.
5. **progressBar = () => {}**
   The “progressBar” function iterates over each element with the class progressBar, adds a span element inside it, and sets the width of the span to match the value of the data-progress attribute of the original element
6. **getUrlVars = () => {}**
   The "getUrlVars" function extracts query parameters from the current URL and returns them as an array where each parameter is both an item and a property of the array.
7. **getEvaluateSim = () => {}**
   This JavaScript function, getEvaluateSim, is responsible for retrieving evaluation simulation data asynchronously from *'includes/educator/overview-dashboard-data.php'* and updating various elements on the page based on the received data.
8. **learnerReport = (sim_id, learner) => {}**
   This function asynchronously retrieves a learner's report data from *'includes/educator/overview-dashboard-data.php'*, updates UI elements with the received data, and handles loading indicators and errors appropriately. It seems to be part of a web application's functionality to dynamically display learner reports in a structured format.
9. **resetTable = () => {}**
   The resetTable function is a JavaScript function designed to reset and clear specific DataTables on a webpage, as well as remove certain elements.
10. **transformData = (filterData) => {}**
    This function is useful for converting an array of user data objects into a format suitable

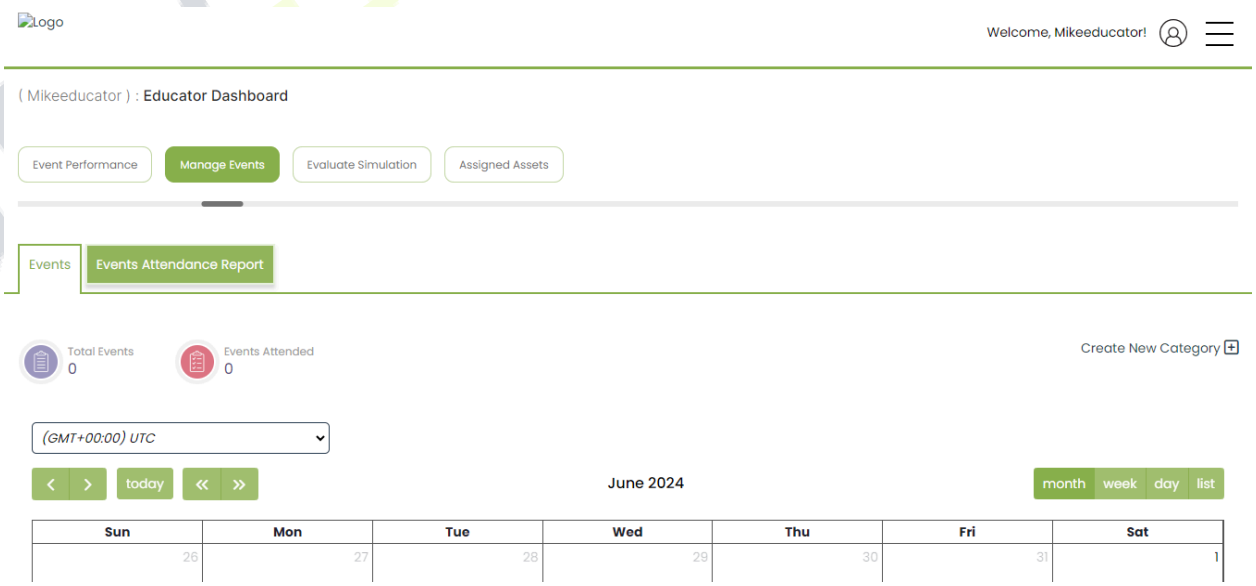
for tabular display or further manipulation, providing consistency and flexibility in handling the data.

11. **transformCategoryPathAssignmentData = (filterData, report_type) => {}**
    This transformCategoryPathAssignmentData function takes two parameters: filterData, which is an object containing filtered data, and report_type, which specifies the type of report being generated. The function transforms the input data into a specific format based on the report type.
12. **groupPerformanceTbl = (columns = [], reportData = []) => {}**
    This function is used to create and update a DataTable displaying performance data for a group. It first clears any existing DataTable, transforms the raw data, and then initializes a new DataTable with specified columns and options. Additionally, it sets up customizations such as pagination, scrolling, and export buttons.
13. **learningPathReport = (columns = [], reportData = []) => {}**
    The learningPathReport function is a JavaScript function that initializes and configures a DataTable on an HTML element with the ID #path_report_tbl.
14. **resetCircleBar = () => {}**
    This function ensures that the circular progress bars and their associated text indicators are reset to zero before potentially updating them with new data.
15. **genCategoryPathReportDataTable = (columns = [], reportData = [], dataTableId, report_type) => {}**
    This function genCategoryPathReportDataTable generates a data table for displaying reports related to category or path assignments.
16. **$('#category, #fcategory').on('change', function(e) {})**
    This code handles the event triggered when a selection is made in a select element, populating other select elements based on the selected value using AJAX to *'includes/ajax.php'*.
17. **$('#scenario, #fscenario').on('change', function(e) {})**
    This code handles the event triggered when a selection is made in one of the specified select elements, populating other select elements based on the selected value using AJAX to *'includes/ajax.php'*.
18. **$('#group').on('change', function(e) {})**
    This code handles the event when a selection is made in the #group select element, sends an AJAX request to *'includes/educator/overview-dashboard-data.php'* to the server to fetch data based on the selected group and scenario IDs, and updates the page with the received data.
19. **$('.filter_type').on('change', function() {})**
    This code dynamically enables or disables multiselect options based on the selected type and performs additional operations for specific types, such as resetting tables and enabling/disabling multiselects.
20. **$('#gr_category').on('change', function() {})**
    This code dynamically updates from url "*'includes/ajax.php'*+ '?getEducatorScenario=true&catid='+ id" the options in the #gr_scenario multiselect based on the selected category ID and performs various reset operations when the category changes.

21. **$('#gr_scenario').on('change', function(){})**
    This code dynamically updates the options in multiple multiselect elements based on the selected scenario ID and handles various UI states based on the success or failure of the AJAX request to *'includes/ajax.php'* + '?group_performance_educator=true&sim_id='+ id.
22. **$("#group_performance").submit(function(e) {})**
    This code submits form data asynchronously to *'includes/educator/group-performance-report-data.php'*, handles the server response, and updates the UI accordingly.
23. **$("#edu_assignment_report").submit(function(e) {})**
    On submission of the form with the ID #edu_assignment_report, it makes AJAX request to the server at *'includes/educator/group-performance-report-data.php'* and on successful response updates the data on the page.

Manage Events > Events

Template Name: educator-events.php



PHP Functions Used

1. **getAssigneeName($userid)**
   Global function from “db.class.php”. This function will return all the data for user as per the User ID passed in the argument.

Third Party JS Library

1. CanvasJS
2. DateRangePicker

3. FullCalender
4. TooltipJS
5. SweetAlert
6. Fancybox

JS Files Used

File Path: content/js/educator/educator-overview-dash.js

1. **getPercentage = (x, y, html = false) => {}**
   This JavaScript code defines a function named "getPercentage" that calculates the percentage of x relative to y, optionally returning the result as a formatted string with a percentage sign.
2. **cb(start, end) {}**
   This JavaScript code defines a function named "cb" that updates the values of specific input elements with formatted date strings based on the given start and end dates.
3. **$('#reportrange, #group_report_range').daterangepicker({})**
   This code sets up a DateRangePicker plugin for two elements and configures various options such as initial date range, date format, positioning behavior, predefined ranges, and a callback function for handling date range changes.
4. **CircleBar = () => {}**
   This JavaScript code defines a function named "CircleBar" that dynamically updates the style of elements representing circular progress bars on a webpage.
5. **progressBar = () => {}**
   The “progressBar” function iterates over each element with the class progressBar, adds a span element inside it, and sets the width of the span to match the value of the data-progress attribute of the original element
6. **getUrlVars = () => {}**
   The "getUrlVars" function extracts query parameters from the current URL and returns them as an array where each parameter is both an item and a property of the array.
7. **getEvaluateSim = () => {}**
   This JavaScript function, getEvaluateSim, is responsible for retrieving evaluation simulation data asynchronously from *'includes/educator/overview-dashboard-data.php'* and updating various elements on the page based on the received data.
8. **learnerReport = (sim_id, learner) => {}**
   This function asynchronously retrieves a learner's report data from *'includes/educator/overview-dashboard-data.php'*, updates UI elements with the received data, and handles loading indicators and errors appropriately. It seems to be part of a web application's functionality to dynamically display learner reports in a structured format.
9. **resetTable = () => {}**
   The resetTable function is a JavaScript function designed to reset and clear specific DataTables on a webpage, as well as remove certain elements.
10. **transformData = (filterData) => {}**
    This function is useful for converting an array of user data objects into a format suitable

for tabular display or further manipulation, providing consistency and flexibility in handling the data.

11. **transformCategoryPathAssignmentData = (filterData, report_type) => {}**
    This transformCategoryPathAssignmentData function takes two parameters: filterData, which is an object containing filtered data, and report_type, which specifies the type of report being generated. The function transforms the input data into a specific format based on the report type.
12. **groupPerformanceTbl = (columns = [], reportData = []) => {}**
    This function is used to create and update a DataTable displaying performance data for a group. It first clears any existing DataTable, transforms the raw data, and then initializes a new DataTable with specified columns and options. Additionally, it sets up customizations such as pagination, scrolling, and export buttons.
13. **learningPathReport = (columns = [], reportData = []) => {}**
    The learningPathReport function is a JavaScript function that initializes and configures a DataTable on an HTML element with the ID #path_report_tbl.
14. **resetCircleBar = () => {}**
    This function ensures that the circular progress bars and their associated text indicators are reset to zero before potentially updating them with new data.
15. **genCategoryPathReportDataTable = (columns = [], reportData = [], dataTableId, report_type) => {}**
    This function genCategoryPathReportDataTable generates a data table for displaying reports related to category or path assignments.
16. **$('#category, #fcategory').on('change', function(e) {})**
    This code handles the event triggered when a selection is made in a select element, populating other select elements based on the selected value using AJAX to *'includes/ajax.php'*.
17. **$('#scenario, #fscenario').on('change', function(e) {})**
    This code handles the event triggered when a selection is made in one of the specified select elements, populating other select elements based on the selected value using AJAX to *'includes/ajax.php'*.
18. **$('#group').on('change', function(e) {})**
    This code handles the event when a selection is made in the #group select element, sends an AJAX request to *'includes/educator/overview-dashboard-data.php'* to the server to fetch data based on the selected group and scenario IDs, and updates the page with the received data.
19. **$('.filter_type').on('change', function() {})**
    This code dynamically enables or disables multiselect options based on the selected type and performs additional operations for specific types, such as resetting tables and enabling/disabling multiselects.
20. **$('#gr_category').on('change', function() {})**
    This code dynamically updates from url "*'includes/ajax.php'*+ '?getEducatorScenario=true&catid='+ id" the options in the #gr_scenario multiselect based on the selected category ID and performs various reset operations when the category changes.

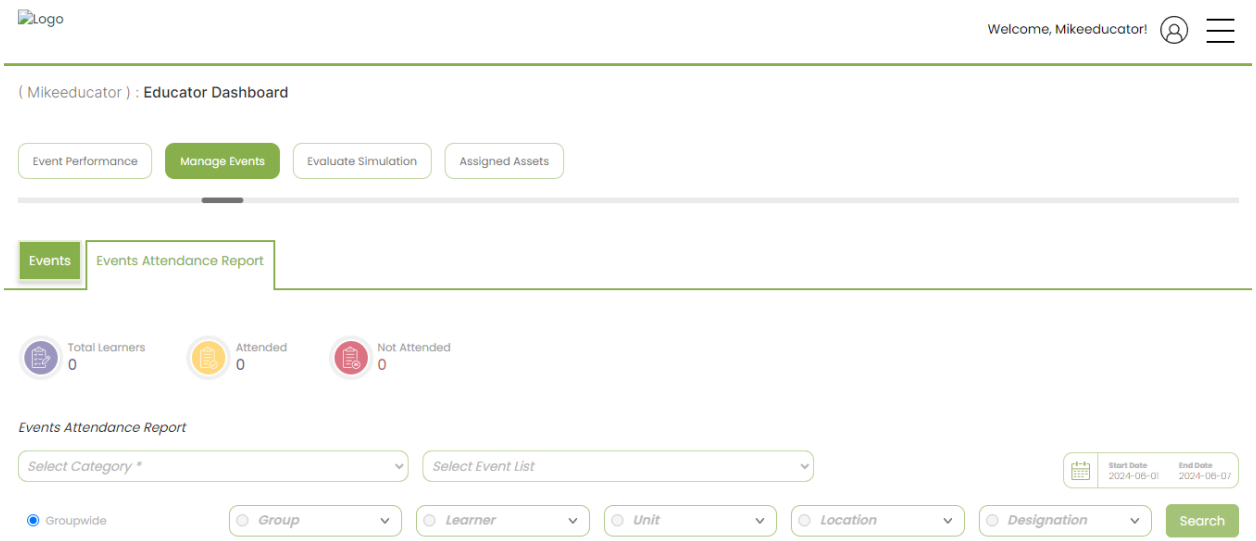
21. **$('#gr_scenario').on('change', function(){})**
    This code dynamically updates the options in multiple multiselect elements based on the selected scenario ID and handles various UI states based on the success or failure of the AJAX request to *'includes/ajax.php'* + '?group_performance_educator=true&sim_id='+ id.
22. **$("#group_performance").submit(function(e) {})**
    This code submits form data asynchronously to *'includes/educator/group-performance-report-data.php'*, handles the server response, and updates the UI accordingly.
23. **$("#edu_assignment_report").submit(function(e) {})**
    On submission of the form with the ID #edu_assignment_report, it makes AJAX request to the server at *'includes/educator/group-performance-report-data.php'* and on successful response updates the data on the page.

FilePath: content/js/educator/educator-events.js

1. **baseUrl = (path = '') => {}**
   The "getBaseUrl" function generates the base URL for a web page, optionally appending a specific page path.
2. **eventsTemp = (data = false, start = false) => {}**
   This function eventsTemp dynamically generates HTML content for adding or editing events based on the provided event data and start time. It includes form elements for event details, attachment uploads, cover image uploads, and group assignments.
3. **viewEventTemp = (info) => {}**
   This function viewEventTemp dynamically generates HTML content to display detailed information about an event, including its title, category, assigned groups, geographical location, description, creator details, and links to the event URL, attachment, and cover image (if available).
4. **eventsAttended = (array) => {}**
   This JavaScript function eventsAttended sends a POST request to the server at *'includes/educator/events.php'* to retrieve the number of events attended by a user or a group of users.
5. **calendar = new FullCalendar.Calendar(calendarEl, {})**
   This JavaScript code initializes a FullCalendar instance with various configurations and event handlers.

Manage Events > Events Attendance Report

Template Name: educator-events-report.php



PHP Functions Used

1. **getAssigneeName($userid)**
   Global function from "db.class.php". This function will return all the data for user as per the User ID passed in the argument.

Third Party JS Library

1. CanvasJS
2. DateRangePicker
3. Datatables

JS Files Used

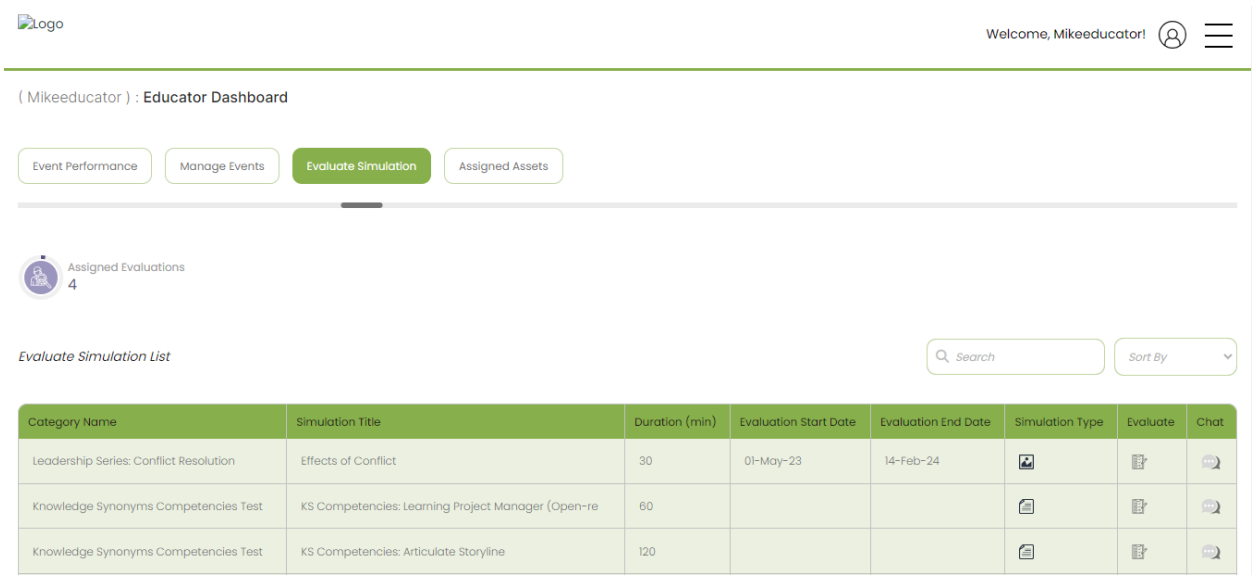
File Path: content/js/educator/event-report.js

1. **cb(start, end) {}**
   The function cb(start, end) is designed to update the content of a specific HTML element with the ID reportrange by setting the inner HTML of a span element within it to display a formatted date range.
2. **$('#group_report_range').daterangepicker({})**
   This code sets up a date range picker with various options and predefined date ranges, allowing users to easily select date ranges using a graphical interface.
3. **$('.multiselection').multiselect({})**
   This code snippet initializes a multi-select dropdown menu for elements with the class .multiselection.
4. **$('.filter_type').on('change', function() {})**
   This code controls the behavior of multiselect dropdowns based on the selected option in another dropdown, allowing users to filter data based on different criteria.

5. **resetTable = () => {}**
   The resetTable function is a JavaScript function designed to reset and clear specific DataTables on a webpage, as well as remove certain elements.
6. **$('#category').on('change', function() {})**
   This code snippet sets up an event listener for changes on a select element with the ID category. When the selected option changes, it triggers a series of actions to update another element with the ID scenario based on the selected category.
7. **$('#event').on('change', function(){})**
   The provided code is a jQuery script that attaches an event listener to an HTML element with the ID #event. This listener responds to changes in the element's value, typically a dropdown menu or input field.
8. **$("#group_performance").submit(function(e) {})**
   The provided code is a jQuery script that handles the submission of a form with the ID #group_performance. It prevents the default form submission behavior, sends an AJAX request to *'includes/educator/events.php'* to fetch data, and processes the response to populate a DataTable.
9. **$('body').on('click', '.mark_attendance', function() {})**
   This code snippet is an event listener attached to the body element that listens for clicks on elements with the class mark_attendance. It sends a POST request to the server at *'includes/ajax.php'*. It shows a success message using swal(), indicating that the attendance has been added.

Evaluate Simulation

Template Name: educator-overview-dashboard.php



PHP Functions Used

1. **getAssigneeName($userid)**
   Global function from “db.class.php”. This function will return all the data for user as per the User ID passed in the argument.
2. **getClientPermission($client_id)**
   Global function from “db.class.php”. This function returns all the data for the permissions from table client_permissions for the give client_id.
3. **SQL Queries**
   $sqlr = "SELECT DISTINCT s.scenario_id, s.sim_temp_type, s.Scenario_title, LEFT(s.Scenario_title, 50) AS stitle, s.scenario_category, s.duration, s.evaluate_sdate, s.evaluate_edate, DATE_FORMAT(s.evaluate_sdate, '%d-%b-%y') AS sdate, DATE_FORMAT(s.evaluate_edate, '%d-%b-%y') AS edate, c.category, t.type_name, t.type_icon FROM scenario_master AS s LEFT JOIN category_tbl AS c ON c.cat_id = s.scenario_category LEFT JOIN scenario_type AS t ON t.st_id = s.author_scenario_type WHERE s.author_scenario_type != 0 AND s.sim_temp_type IN (7,12,16,17,18,19) AND s.scenario_id IN ($sim_id) AND (s.sim_temp_type <> 12 OR EXISTS (SELECT 1 FROM question_tbl AS q WHERE q.scenario_id = s.scenario_id AND q.question_type = 9)) $orderby";
   It selects various columns (s.scenario_id, s.sim_temp_type, etc.) from the scenario_master table, with aliases and formatting applied to some columns. The query involves joining tables category_tbl and scenario_type using the LEFT JOIN clause with several WHERE clause.

Third Party JS Library

1. CanvasJS
2. DateRangePicker
3. ChatApp

JS File Used

File Path: content/js/educator/educator-overview-dash.js

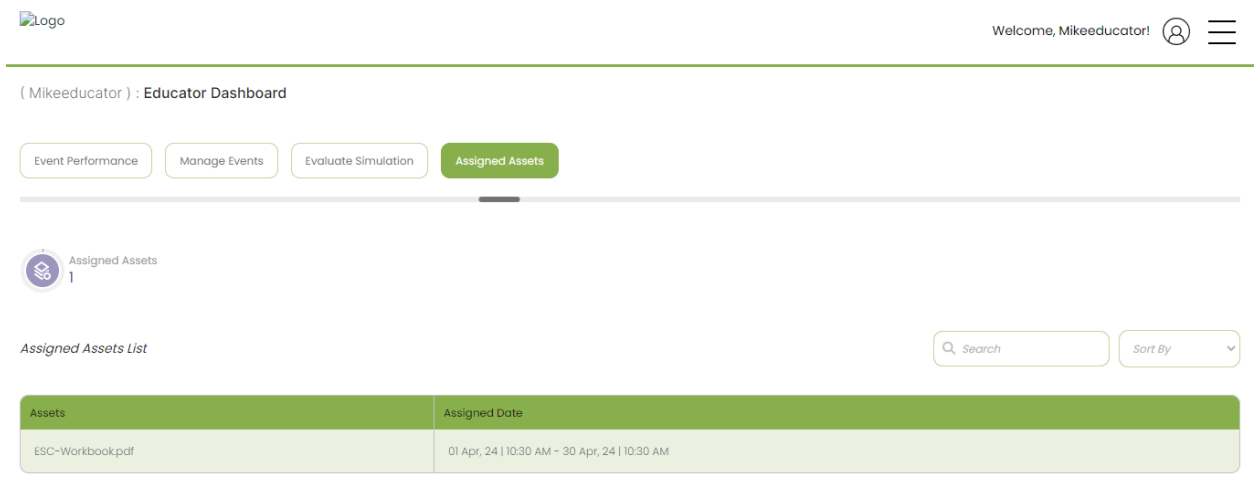
1. **getPercentage = (x, y, html = false) => {}**
   This JavaScript code defines a function named "getPercentage" that calculates the percentage of x relative to y, optionally returning the result as a formatted string with a percentage sign.
2. **cb(start, end) {}**
   This JavaScript code defines a function named "cb" that updates the values of specific input elements with formatted date strings based on the given start and end dates.
3. **$('#reportrange, #group_report_range').daterangepicker({})**
   This code sets up a DateRangePicker plugin for two elements and configures various options such as initial date range, date format, positioning behavior, predefined ranges, and a callback function for handling date range changes.

4. **CircleBar = () => {}**
   This JavaScript code defines a function named "CircleBar" that dynamically updates the style of elements representing circular progress bars on a webpage.
5. **progressBar = () => {}**
   The “progressBar” function iterates over each element with the class progressBar, adds a span element inside it, and sets the width of the span to match the value of the data-progress attribute of the original element
6. **getUrlVars = () => {}**
   The "getUrlVars" function extracts query parameters from the current URL and returns them as an array where each parameter is both an item and a property of the array.
7. **getEvaluateSim = () => {}**
   This JavaScript function, getEvaluateSim, is responsible for retrieving evaluation simulation data asynchronously from *'includes/educator/overview-dashboard-data.php'* and updating various elements on the page based on the received data.
8. **learnerReport = (sim_id, learner) => {}**
   This function asynchronously retrieves a learner's report data from *'includes/educator/overview-dashboard-data.php'*, updates UI elements with the received data, and handles loading indicators and errors appropriately. It seems to be part of a web application's functionality to dynamically display learner reports in a structured format.
9. **resetTable = () => {}**
   The resetTable function is a JavaScript function designed to reset and clear specific DataTables on a webpage, as well as remove certain elements.
10. **transformData = (filterData) => {}**
    This function is useful for converting an array of user data objects into a format suitable for tabular display or further manipulation, providing consistency and flexibility in handling the data.
11. **transformCategoryPathAssignmentData = (filterData, report_type) => {}**
    This transformCategoryPathAssignmentData function takes two parameters: filterData, which is an object containing filtered data, and report_type, which specifies the type of report being generated. The function transforms the input data into a specific format based on the report type.
12. **groupPerformanceTbl = (columns = [], reportData = []) => {}**
    This function is used to create and update a DataTable displaying performance data for a group. It first clears any existing DataTable, transforms the raw data, and then initializes a new DataTable with specified columns and options. Additionally, it sets up customizations such as pagination, scrolling, and export buttons.
13. **learningPathReport = (columns = [], reportData = []) => {}**
    The learningPathReport function is a JavaScript function that initializes and configures a DataTable on an HTML element with the ID #path_report_tbl.
14. **resetCircleBar = () => {}**
    This function ensures that the circular progress bars and their associated text indicators are reset to zero before potentially updating them with new data.

15. **genCategoryPathReportDataTable = (columns = [], reportData = [], dataTableId, report_type) => {}**
    This function genCategoryPathReportDataTable generates a data table for displaying reports related to category or path assignments.
16. **$('#category, #fcategory').on('change', function(e) {})**
    This code handles the event triggered when a selection is made in a select element, populating other select elements based on the selected value using AJAX to 'includes/ajax.php'.
17. **$('#scenario, #fscenario').on('change', function(e) {})**
    This code handles the event triggered when a selection is made in one of the specified select elements, populating other select elements based on the selected value using AJAX to *'includes/ajax.php'*.
18. **$('#group').on('change', function(e) {})**
    This code handles the event when a selection is made in the #group select element, sends an AJAX request to *'includes/educator/overview-dashboard-data.php'* to the server to fetch data based on the selected group and scenario IDs, and updates the page with the received data.
19. **$('.filter_type').on('change', function() {})**
    This code dynamically enables or disables multiselect options based on the selected type and performs additional operations for specific types, such as resetting tables and enabling/disabling multiselects.
20. **$('#gr_category').on('change', function() {})**
    This code dynamically updates from url "*'includes/ajax.php'*+ '?getEducatorScenario=true&catid='+ id" the options in the #gr_scenario multiselect based on the selected category ID and performs various reset operations when the category changes.
21. **$('#gr_scenario').on('change', function(){})**
    This code dynamically updates the options in multiple multiselect elements based on the selected scenario ID and handles various UI states based on the success or failure of the AJAX request to *'includes/ajax.php'* + '?group_performance_educator=true&sim_id='+ id.
22. **$("#group_performance").submit(function(e) {})**
    This code submits form data asynchronously to *'includes/educator/group-performance-report-data.php'*, handles the server response, and updates the UI accordingly.
23. **$("#edu_assignment_report").submit(function(e) {})**
    On submission of the form with the ID #edu_assignment_report, it makes AJAX request to the server at *'includes/educator/group-performance-report-data.php'* and on successful response updates the data on the page.

Assigned Assets

Template Name: educator-assigned-assets.php



PHP Functions Used

1. **getAssigneeName($userid)**
   Global function from "db.class.php". This function will return all the data for user as per the User ID passed in the argument.
2. **getClientPermission($client_id)**
   Global function from "db.class.php". This function returns all the data for the permissions from table client_permissions for the give client_id.
3. **SQL Queries**
   $sqlr = "SELECT id, assets, DATE_FORMAT(start_date, '%d %b, %y | %H:%i %p') AS sdate, DATE_FORMAT(end_date, '%d %b, %y | %H:%i %p') AS edate, CASE WHEN (start_date IS NOT NULL AND end_date IS NOT NULL AND NOW() BETWEEN start_date AND end_date) OR (start_date IS NULL AND end_date IS NULL) THEN 'Enabled' ELSE 'Disabled' END AS download_statusFROM assigned_assets WHERE status = 1 AND FIND_IN_SET(?, educator) $orderby";
   It selects specific columns (id, assets, start_date, end_date) from the assigned_assets table. The DATE_FORMAT function is used to format the start_date and end_date columns in a specific date and time format. The query includes a CASE statement to conditionally determine the download_status based on the conditions. The WHERE clause filters the rows based on the conditions status = 1 and FIND_IN_SET(?, educator).

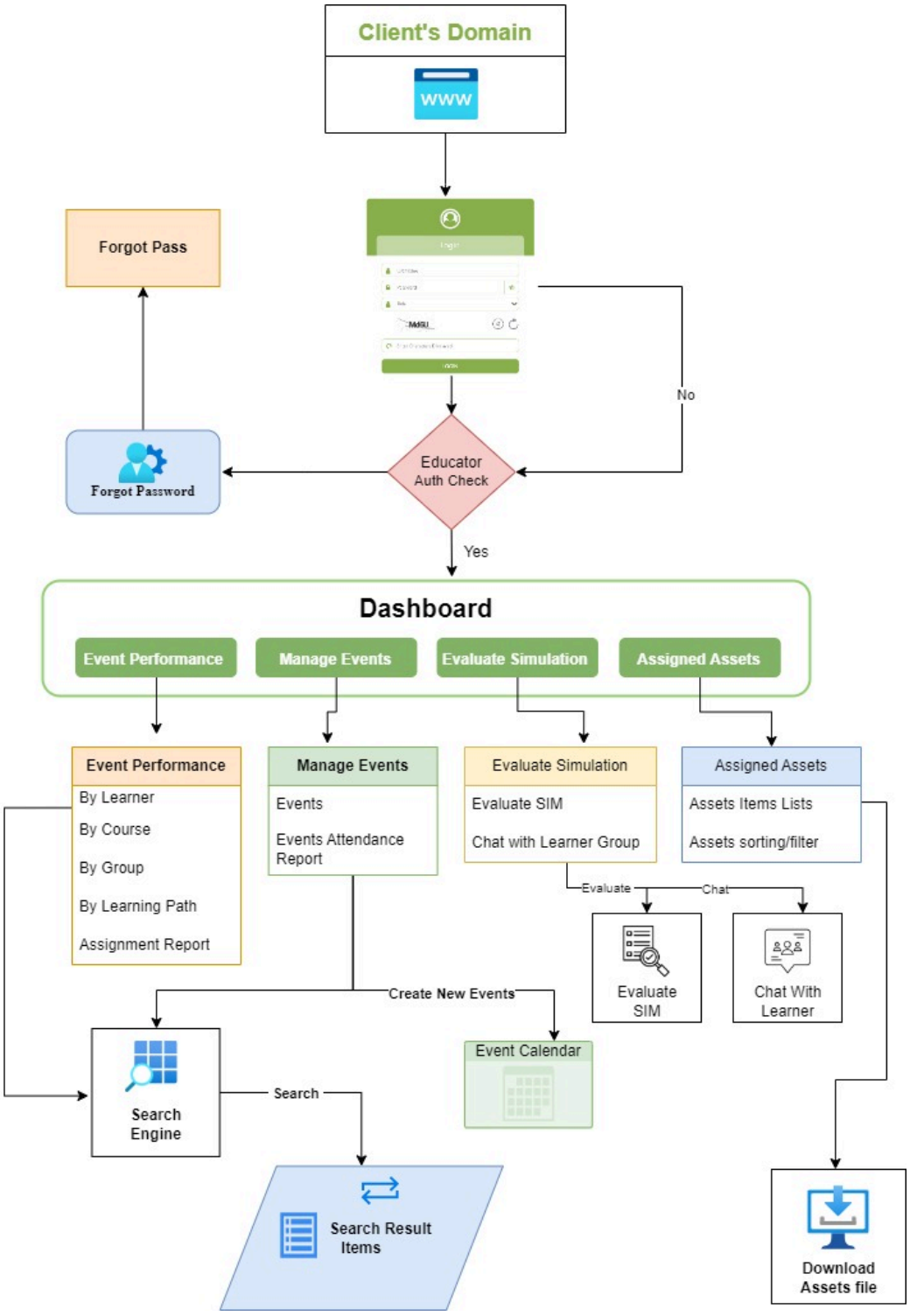
Third Party JS Library

1. SweetAlert

JS Functions Used

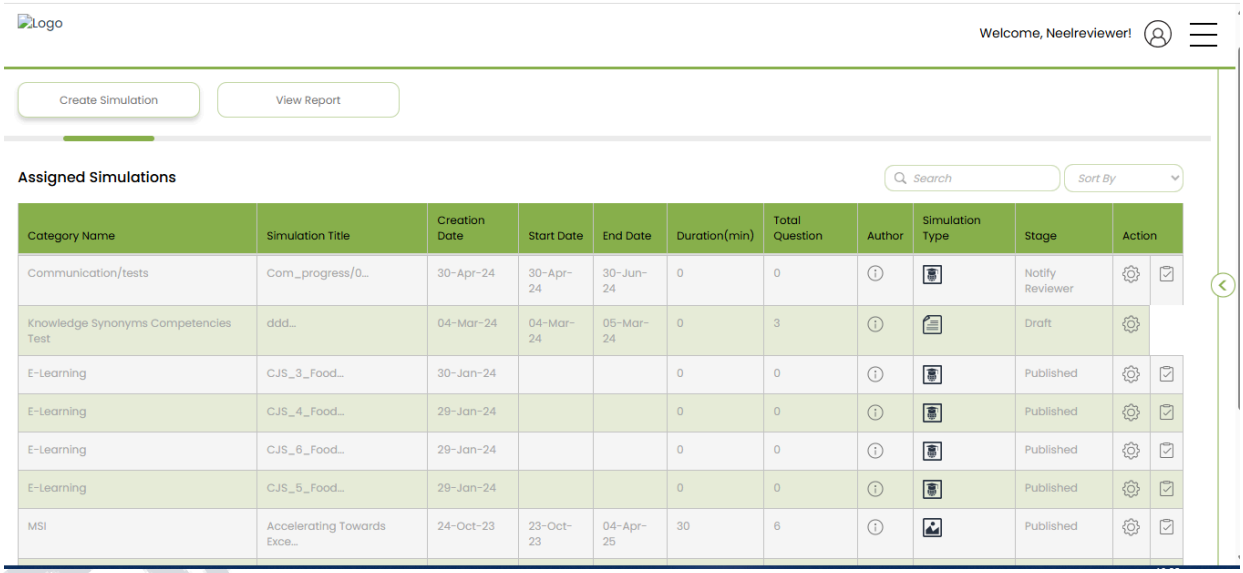
1. **getUrlVars = () => {}**
   The "getUrlVars" function extracts query parameters from the current URL and returns them as an array where each parameter is both an item and a property of the array.
2. **CircleBar = () => {}**
   The "CircleBar" function initializes circular progress bars on elements with specific classes by setting the "stroke-dashoffset" CSS property based on a data attribute.
3. **saveDownloadTrack = (id) => {}**
   This JavaScript function saveDownloadTrack sends a POST request to a server-side PHP script (*includes/process.php*) to save download tracking information.
4. **downloadFile = (filePath, fileName) => {}**
   This downloadFile function in JavaScript facilitates the downloading of a file by creating a temporary anchor element and simulating a click on it.
5. **$('.download_assets').on('click', async function(){})**
   This script sets up a click event listener for elements with the class download_assets. When clicked, it initiates a download process after displaying a confirmation dialog with terms and conditions.

Educator DFD



Reviewer

Overview



Category Name	Simulation Title	Creation Date	Start Date	End Date	Duration(min)	Total Question	Author	Simulation Type	Stage	Action
Communication/tests	Com_progress/0...	30-Apr-24	30-Apr-24	30-Jun-24	0	0			Notify Reviewer	
Knowledge Synonyms Competencies Test	ddd...	04-Mar-24	04-Mar-24	05-Mar-24	0	3			Draft	
E-Learning	CJS_3_Food...	30-Jan-24			0	0			Published	
E-Learning	CJS_4_Food...	29-Jan-24			0	0			Published	
E-Learning	CJS_6_Food...	29-Jan-24			0	0			Published	
E-Learning	CJS_5_Food...	29-Jan-24			0	0			Published	
MSI	Accelerating Towards Exce...	24-Oct-23	23-Oct-23	04-Apr-25	30	6			Published	

This domain will only be accessible to users with Reviewer credentials. It will allow management of Assigned Simulations on the Dashboard's "Create Simulation" tab. Another tab, named "View Report," will enable users to view all reports related to "Simulation Update," "Publish & Review," and "Assigned Learner."

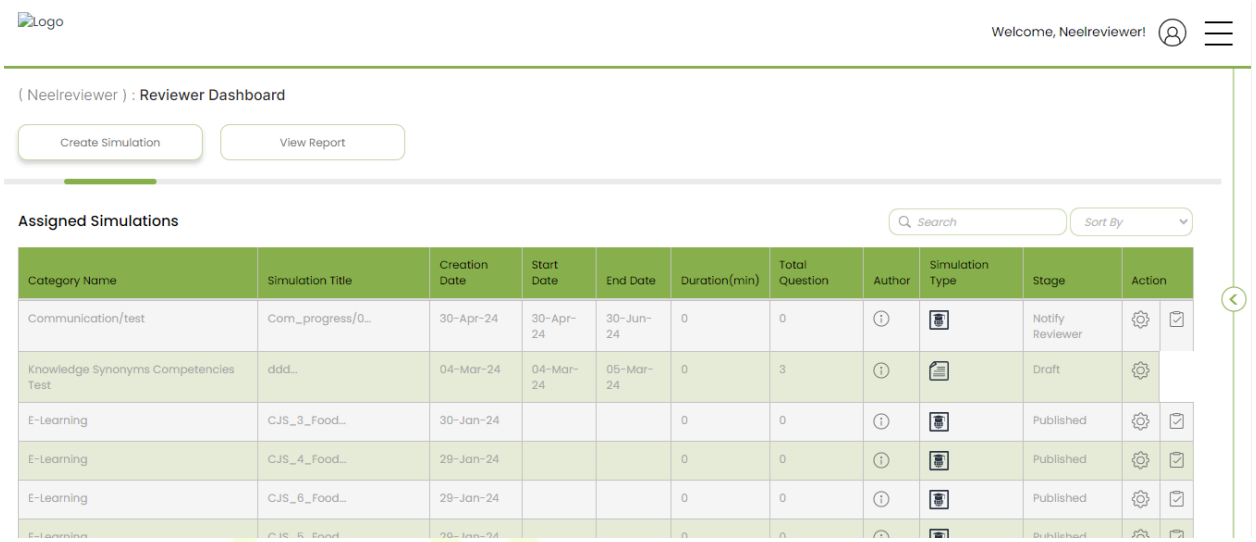
Reviewer dashboard has two tab
1. Create Simulation
2. View Report

1. Create Simulation:- "Reviewers evaluate all simulations assigned by the admin and update the settings, indicating whether the simulation has been reviewed or not."
2. View Report:- "It contains all simulation reports, including the number of simulation updates, total number of simulations, the count of reviewed simulations for publishing, and the number of assigned learners."

Code Stuff

Create Simulation

Template Name: reviewer-overview-dashboard.php



PHP Functions Used

1. **getRole()**
   Global function from "db.class.php". This PHP function getRole() is designed to retrieve roles from a database table named "role" and return them as an associative array.
2. **getAssigneeName($userid)**
   Global function from "db.class.php". This function will return all the data for user as per the User ID passed in the argument.
3. **SQL Queries**
   $sqlr = "SELECT DISTINCT s.scenario_id, s.Scenario_title, LEFT(s.Scenario_title, 25) AS title, s.assign_start_date, s.assign_end_date, DATE_FORMAT(s.assign_start_date, '%d-%b-%y') AS assign_sdate, DATE_FORMAT(s.assign_end_date, '%d-%b-%y') AS assign_edate, s.duration, s.scenario_type, s.scenario_category, s.author_scenario_type, s.assign_author, s.sim_ques_type, s.sim_temp_type, DATE_FORMAT(s.created_on, '%d-%b-%y') AS created, s.update_sim, s.status, c.category, st.status_name, t.type_name, t.type_icon FROM scenario_master s INNER JOIN group_tbl g LEFT JOIN category_tbl c ON c.cat_id = s.scenario_category LEFT JOIN scenario_status st ON st.sid = s.status LEFT JOIN scenario_type t ON t.st_id = s.author_scenario_type WHERE s.author_scenario_type != '0' AND (CASE WHEN s.assign_reviewer != '' THEN FIND_IN_SET($userid, s.assign_reviewer) WHEN s.assign_group != '' THEN FIND_IN_SET($userid, g.reviewer) END) $orderby";
   This query retrieves a comprehensive list of scenarios with various attributes and their associated categories, status, and types. It uses several joins to combine data from multiple tables and includes conditional filtering based on the user's role in the scenario assignment.

Third Party JS Library

1. TooltipJS

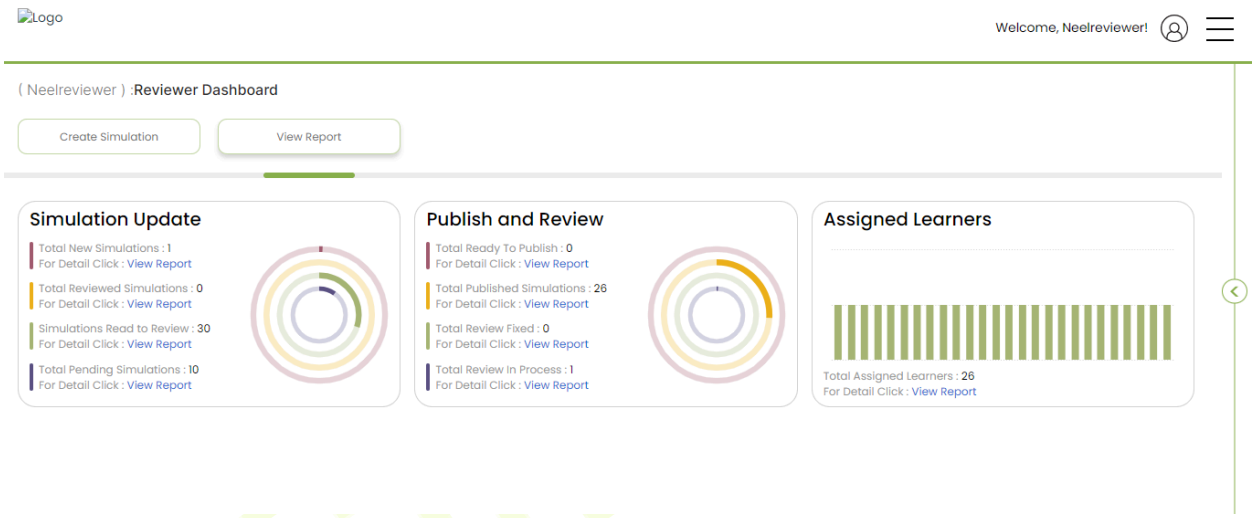
JS File Used

File Path: content/js/reviewer-overview-dash.js

1. **getPercentage = (x, y, html = false) => {}**
   This JavaScript code defines a function named "getPercentage" that calculates the percentage of x relative to y, optionally returning the result as a formatted string with a percentage sign.
2. **CircleBar = () => {}**
   This JavaScript code defines a function named "CircleBar" that dynamically updates the style of elements representing circular progress bars on a webpage.
3. **getUrlVars = () => {}**
   The "getUrlVars" function extracts query parameters from the current URL and returns them as an array where each parameter is both an item and a property of the array.
4. **overview = () => {}**
   The overview function is designed to update a dashboard with various metrics. It sends an AJAX POST request at *'includes/reviewer/overview-dashboard-data.php'* to a server, retrieves data, and then updates the UI elements based on the response.
5. **report = () => {}**
   The report function fetches data via an AJAX POST request at *'includes/reviewer/overview-dashboard-data.php'*, processes the response, and updates various dashboard elements with statistical information about simulations.
6. **$('.update_stage').click(function () {})**
   This code snippet is designed to handle updating the stage of a scenario. When a user clicks on an element with the class update_stage, it retrieves the data-scenario-id and data-status-id attributes from the clicked element. It then shows a loading overlay, sends an AJAX request to load content from *'edit-scenario-stage-modal.php'* into a modal, and finally hides the loading overlay. If the response is valid, the modal is displayed; otherwise, an error message is shown using SweetAlert.

View Report

Template Name: reviewer-report-new.php



PHP Functions Used

1. **getAssigneeName($userid)**
   Global function from "db.class.php". This function will return all the data for user as per the User ID passed in the argument.
2. **isReviewer()**
   Global function from "db.class.php". The isReviewer function in the provided code snippet checks if the current user's role is authorized to access a particular page. If the user's role is not in the list of allowed roles, the function displays an error message and redirects the user to a different page after a brief delay.

Third Party JS Library

1. CanvasJS

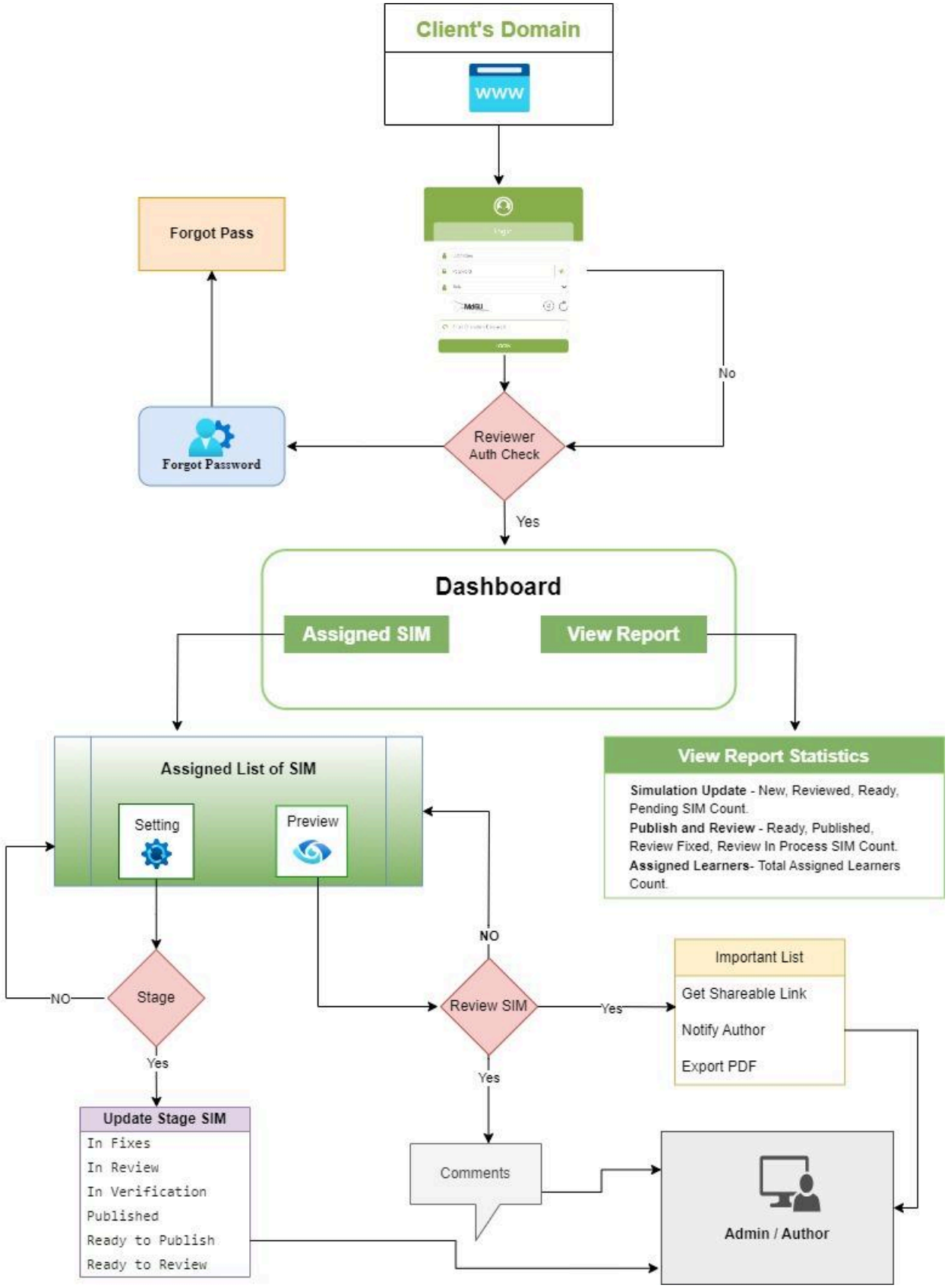
JS File Used

File Path: content/js/reviewer-overview-dash.js

1. **getPercentage = (x, y, html = false) => {}**
   This JavaScript code defines a function named "getPercentage" that calculates the percentage of x relative to y, optionally returning the result as a formatted string with a percentage sign.
2. **CircleBar = () => {}**
   This JavaScript code defines a function named "CircleBar" that dynamically updates the style of elements representing circular progress bars on a webpage.
3. **getUrlVars = () => {}**
   The "getUrlVars" function extracts query parameters from the current URL and returns them as an array where each parameter is both an item and a property of the array.

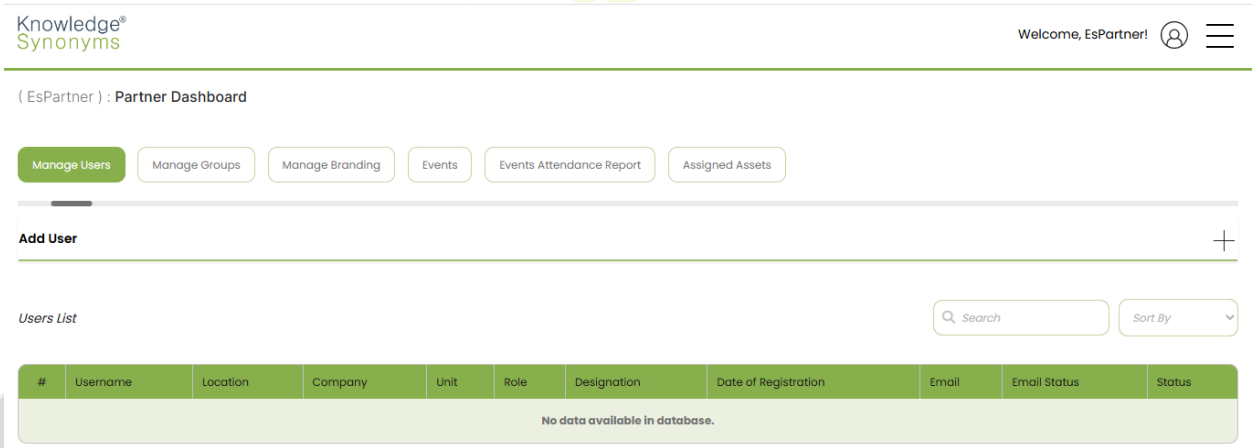
4. **overview = () => {}**
   The overview function is designed to update a dashboard with various metrics. It sends an AJAX POST request at *'includes/reviewer/overview-dashboard-data.php'* to a server, retrieves data, and then updates the UI elements based on the response.
5. **report = () => {}**
   The report function fetches data via an AJAX POST request at *'includes/reviewer/overview-dashboard-data.php'*, processes the response, and updates various dashboard elements with statistical information about simulations.
6. **$('.update_stage').click(function () {})**
   This code snippet is designed to handle updating the stage of a scenario. When a user clicks on an element with the class update_stage, it retrieves the data-scenario-id and data-status-id attributes from the clicked element. It then shows a loading overlay, sends an AJAX request to load content from *'edit-scenario-stage-modal.php'* into a modal, and finally hides the loading overlay. If the response is valid, the modal is displayed; otherwise, an error message is shown using SweetAlert.

Reviewer DFD


Client's Domain
www
Forgot Pass
Forgot Password
No
Reviewer Auth Check
Yes
Dashboard
Assigned SIM
View Report
Assigned List of SIM
Setting
Preview
View Report Statistics
Simulation Update - New, Reviewed, Ready, Pending SIM Count.
Publish and Review - Ready, Published, Review Fixed, Review In Process SIM Count.
Assigned Learners- Total Assigned Learners Count.
NO
Stage
NO
Review SIM
Yes
Important List
Get Shareable Link
Notify Author
Export PDF
Yes
Yes
Update Stage SIM
In Fixes
In Review
In Verification
Published
Ready to Publish
Ready to Review
Comments
Admin / Author

Partner

Overview



This domain will only be accessible to users with Partner credentials. It will allow management of "Add User" and "User List" on the Dashboard's "Manage Users" tab. The next tab, named "Manage Groups," will enable users to view "Group Lists." The next tab, named "Manage Branding," will enable users to upload a "Logo." The next tab, named "Events," will enable users to create and view "Events." The next tab, named "Events Attendance Report," will enable users to view the "Attendance Report." The next tab, named "Assigned Assets," will enable users to view the "Assigned Assets List.".

The Partner Landing page has 6 tabs: Partner is a part of any group who can be assigned by admin.

- Manage Users
  - Click on "Add User" tab after that partner add own users
- Manage Groups
  - If the admin assigns any group, then the partner adds their own user to this group.
- Manage Branding
  - Partner can change their own logo on this page
- Events
  - Partners can create events for their own users.
- Events Attendance Report
  - Partners can download the Events Attendance Report.
- Assigned Assets

- When the admin uploads an asset to them, it then shows the assigned asset on this page.

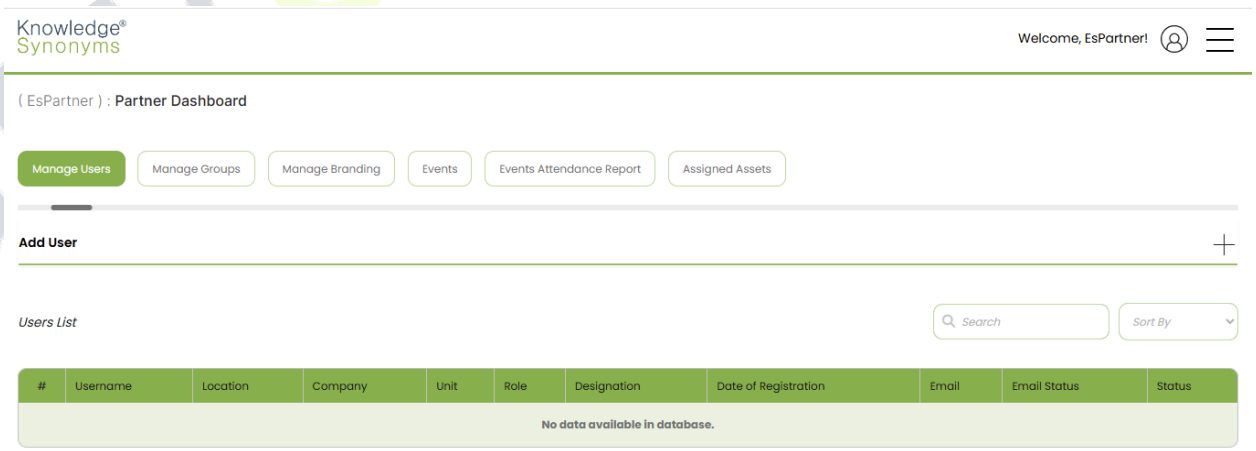
There are 6 tabs in the **Partner**.

I. Manage Users
II. Manage Groups
III. Manage Branding
IV. Events
V. Events Attendance Report
VI. Assigned Assets

Code Stuff

Manage Users

Template Name: partner-overview-dashboard.php



PHP Functions Used

1. **SQL Queries**
   *$sqlr = "SELECT * FROM users WHERE created_by=? $orderby";*
   This SQL query is designed to select all columns from the users table where the created_by column matches a specific value.

Third Party JS Library

1. Bootstrap-DateTimePicker

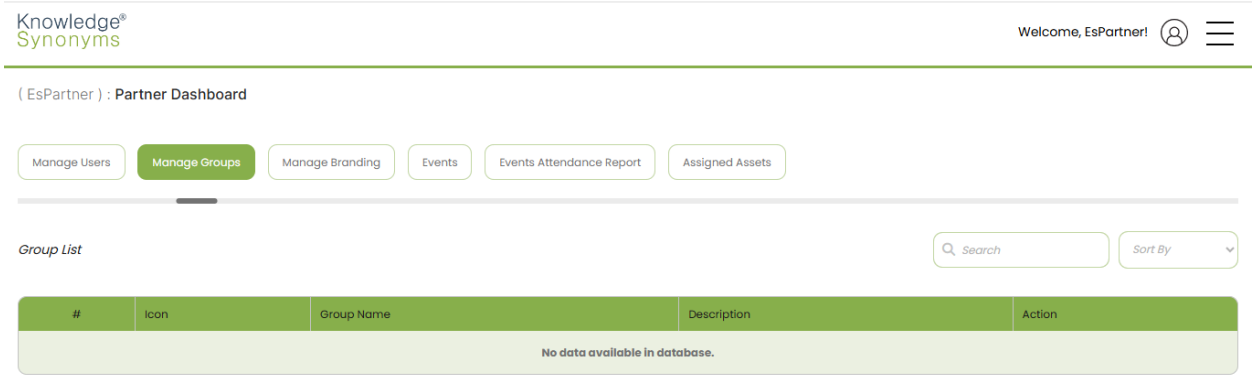
JS File Used

File Path: content/js/partner/overview.js

1. **baseUrl = (path = '') => {}**
   The "getBaseUrl" function generates the base URL for a web page, optionally appending a specific page path.
2. **CircleBar = () => {}**
   This JavaScript code defines a function named "CircleBar" that dynamically updates the style of elements representing circular progress bars on a webpage.
3. **progressBar = () => {}**
   The “progressBar” function iterates over each element with the class progressBar, adds a span element inside it, and sets the width of the span to match the value of the data-progress attribute of the original element
4. **$('.form_datetime').datetimepicker({})**
   This code sets up a datetime picker for HTML elements with the class ".form_datetime".
5. **validateUserName = () => {}**
   The validateUserName function ensures that the username entered by the user only contains letters and numbers.
6. **updateStatus = (status, id) => {}**
   The updateStatus function is designed to update the status of a user in a web application, either setting them as 'Active' or 'Inactive'. The function uses SweetAlert for confirmation dialogs and jQuery for making an AJAX request to *'includes/process.php'*, with parameters as { user_update_status: true, status: status, uid: id }.
7. **('.add_inst_learner').on('click', function(){})**
   This jQuery code sets up an event handler for clicking elements with the class add_inst_learner. It performs an AJAX request to *'add-user-group-partner-modal.php'* to load content into a modal.
8. **$('#upload_partner_logo').click(function() {})**
   This jQuery code snippet sets up an event handler for clicking the element with the ID upload_partner_logo. It handles uploading a partner logo file using AJAX request to *'includes/upload.php'*.
9. **$('#delete_partner_logo').click(function(e) {})**
   This jQuery code snippet sets up an event handler for clicking the element with the ID delete_partner_logo. It handles deleting a partner logo file using AJAX request to *"includes/delete-file.php"*.

Manage Groups

Template Name: partner-manage-group.php



PHP Functions Used

1. **SQL Queries**
   *$sqlr = "SELECT * FROM group_tbl WHERE FIND_IN_SET(?, `partner`) $orderby";*
   This SQL query retrieves all records from the group_tbl table where a specified value is found within a comma-separated list in the partner column.

JS File Used

File Path: content/js/partner/overview.js

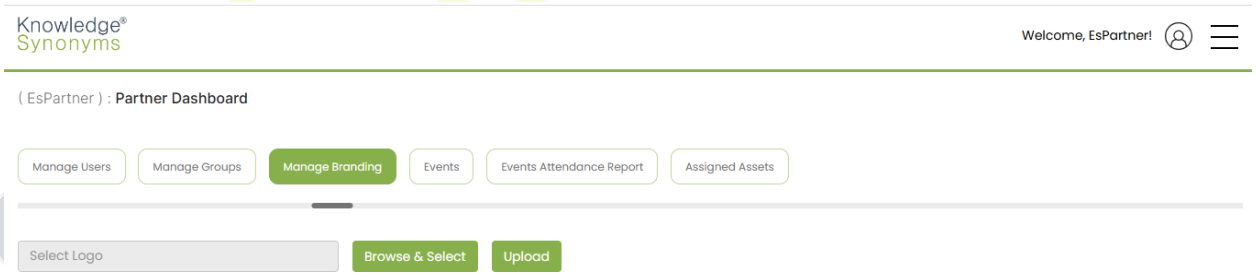
1. **baseUrl = (path = '') => {}**
   The "getBaseUrl" function generates the base URL for a web page, optionally appending a specific page path.
2. **CircleBar = () => {}**
   This JavaScript code defines a function named "CircleBar" that dynamically updates the style of elements representing circular progress bars on a webpage.
3. **progressBar = () => {}**
   The “progressBar” function iterates over each element with the class progressBar, adds a span element inside it, and sets the width of the span to match the value of the data-progress attribute of the original element
4. **$('.form_datetime').datetimepicker({})**
   This code sets up a datetime picker for HTML elements with the class ".form_datetime".
5. **validateUserName = () => {}**
   The validateUserName function ensures that the username entered by the user only contains letters and numbers.
6. **updateStatus = (status, id) => {}**
   The updateStatus function is designed to update the status of a user in a web application, either setting them as 'Active' or 'Inactive'. The function uses SweetAlert for confirmation dialogs and jQuery for making an AJAX request to *'includes/process.php'*, with parameters as { user_update_status: true, status: status, uid: id }.
7. **('.add_inst_learner').on('click', function(){})**
   This jQuery code sets up an event handler for clicking elements with the class

add_inst_learner. It performs an AJAX request to *'add-user-group-partner-modal.php'* to load content into a modal.

8. **$('#upload_partner_logo').click(function() {})**
   This jQuery code snippet sets up an event handler for clicking the element with the ID upload_partner_logo. It handles uploading a partner logo file using AJAX request to *'includes/upload.php'*.
9. **$('#delete_partner_logo').click(function(e) {})**
   This jQuery code snippet sets up an event handler for clicking the element with the ID delete_partner_logo. It handles deleting a partner logo file using AJAX request to *"includes/delete-file.php"*.

Manage Branding

Template Name: partner-manage-branding.php



Third Party JS Library

1. Fancybox

JS File Used

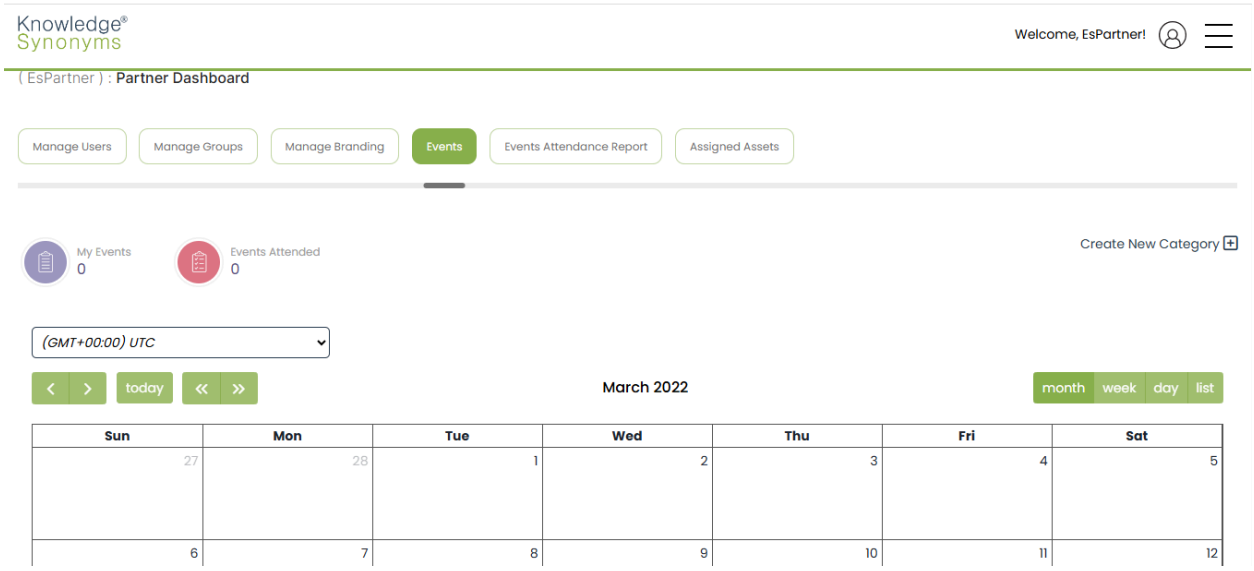
File Path: content/js/partner/overview.js

1. **baseUrl = (path = '') => {}**
   The "getBaseUrl" function generates the base URL for a web page, optionally appending a specific page path.
2. **CircleBar = () => {}**
   This JavaScript code defines a function named "CircleBar" that dynamically updates the style of elements representing circular progress bars on a webpage.
3. **progressBar = () => {}**
   The “progressBar” function iterates over each element with the class progressBar, adds a span element inside it, and sets the width of the span to match the value of the data-progress attribute of the original element
4. **$('.form_datetime').datetimepicker({})**
   This code sets up a datetime picker for HTML elements with the class ".form_datetime".

5. **validateUserName = () => {}**
   The validateUserName function ensures that the username entered by the user only contains letters and numbers.
6. **updateStatus = (status, id) => {}**
   The updateStatus function is designed to update the status of a user in a web application, either setting them as 'Active' or 'Inactive'. The function uses SweetAlert for confirmation dialogs and jQuery for making an AJAX request to *'includes/process.php'*, with parameters as { user_update_status: true, status: status, uid: id }.
7. **('.add_inst_learner').on('click', function(){})**
   This jQuery code sets up an event handler for clicking elements with the class add_inst_learner. It performs an AJAX request to *'add-user-group-partner-modal.php'* to load content into a modal.
8. **$('#upload_partner_logo').click(function() {})**
   This jQuery code snippet sets up an event handler for clicking the element with the ID upload_partner_logo. It handles uploading a partner logo file using AJAX request to *'includes/upload.php'*.
9. **$('#delete_partner_logo').click(function(e) {})**
   This jQuery code snippet sets up an event handler for clicking the element with the ID delete_partner_logo. It handles deleting a partner logo file using AJAX request to *"includes/delete-file.php"*.

Events

Template Name: partner-events.php



Third Party JS Library

1. CanvasJs
2. DateRangePicker

3. SweetAlert
4. Fancybox
5. FullCalender
6. TooltipJs

JS File Used

File Path: content/js/partner/overview.js

1. **baseUrl = (path = '') => {}**
   The "getBaseUrl" function generates the base URL for a web page, optionally appending a specific page path.
2. **CircleBar = () => {}**
   This JavaScript code defines a function named "CircleBar" that dynamically updates the style of elements representing circular progress bars on a webpage.
3. **progressBar = () => {}**
   The “progressBar” function iterates over each element with the class progressBar, adds a span element inside it, and sets the width of the span to match the value of the data-progress attribute of the original element
4. **$('.form_datetime').datetimepicker({})**
   This code sets up a datetime picker for HTML elements with the class ".form_datetime".
5. **validateUserName = () => {}**
   The validateUserName function ensures that the username entered by the user only contains letters and numbers.
6. **updateStatus = (status, id) => {}**
   The updateStatus function is designed to update the status of a user in a web application, either setting them as 'Active' or 'Inactive'. The function uses SweetAlert for confirmation dialogs and jQuery for making an AJAX request to *'includes/process.php'*, with parameters as { user_update_status: true, status: status, uid: id }.
7. **('.add_inst_learner').on('click', function(){})**
   This jQuery code sets up an event handler for clicking elements with the class add_inst_learner. It performs an AJAX request to *'add-user-group-partner-modal.php'* to load content into a modal.
8. **$('#upload_partner_logo').click(function() {})**
   This jQuery code snippet sets up an event handler for clicking the element with the ID upload_partner_logo. It handles uploading a partner logo file using AJAX request to *'includes/upload.php'*.
9. **$('#delete_partner_logo').click(function(e) {})**
   This jQuery code snippet sets up an event handler for clicking the element with the ID delete_partner_logo. It handles deleting a partner logo file using AJAX request to *"includes/delete-file.php"*.

File Path: content/js/partner-events.js

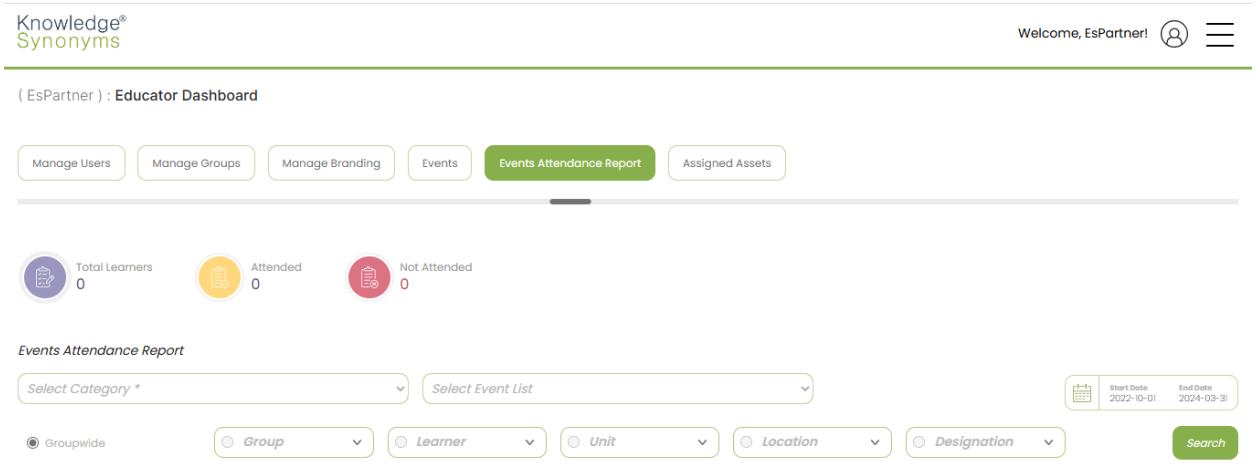
1. **eventsTemp = (data = false, start = false) => {}**
   This function eventsTemp dynamically generates HTML content for adding or editing

events based on the provided event data and start time. It includes form elements for event details, attachment uploads, cover image uploads, and group assignments.

2. **viewEventTemp = (info) => {}**
   This function viewEventTemp dynamically generates HTML content to display detailed information about an event, including its title, category, assigned groups, geographical location, description, creator details, and links to the event URL, attachment, and cover image (if available).
3. **eventsAttended = (array) => {}**
   This JavaScript function eventsAttended sends a POST request to the server at *'includes/educator/events.php'* to retrieve the number of events attended by a user or a group of users.
4. **calendar = new FullCalendar.Calendar(calendarEl, {})**
   This JavaScript code initializes a FullCalendar instance with various configurations and event handlers.
5. **$('body').on('change', '#time-zone-selector', function (e) {})**
   This jQuery code snippet sets up an event handler for the change event on the element with the ID time-zone-selector. It's used for updating the time zone in a calendar and making an AJAX request to update the time zone in the backend.
6. **$('body').on('click', '.upload_event_attachment', function (e) {})**
   This code snippet handles the process of uploading an event attachment file when an element with the class .upload_event_attachment is clicked. It utilizes AJAX to send the file data to the server, processes the server's response, and updates the UI accordingly. Additionally, it provides feedback to the user by displaying success or error messages.
7. **$('body').on('click', '.delete_event_attachment', function (e) {})**
   This jQuery code snippet sets up an event handler for clicking on elements with the class .delete_event_attachment. It's used for deleting an event attachment file via AJAX request to *'includes/delete-file.php'*.
8. **$('body').on('click', '.upload_cover_image', function (e) {})**
   This code snippet handles the process of uploading a cover image file when an element with the class .upload_cover_image is clicked. It utilizes AJAX to send the file data to the server, processes the server's response, and updates the UI accordingly.
9. **$('body').on('click', '.delete_cover_image', function (e) {})**
   This code snippet handles the process of deleting a cover image file when an element with the class .delete_cover_image is clicked. It utilizes AJAX to send a request to the server at *'includes/delete-file.php'* to delete the file and provides feedback to the user by displaying success or error messages.
10. **$('#add_category').on('click', async () => {})**
    This code sets up an event handler for clicking on an element with the ID add_category. It's used to add a new category through a modal dialog using SweetAlert and then sending the data to the server via AJAX request to *'includes/process.php'*.

Events Attendance Report

Template Name: partner-events-report.php



PHP Functions Used

1. **getAssigneeName($userid)**
   Global function from "db.class.php". This function will return all the data for user as per the User ID passed in the argument.

Third Party JS Library

1. Datatables
2. CanvasJS
3. DateRangePicker

JS File Used

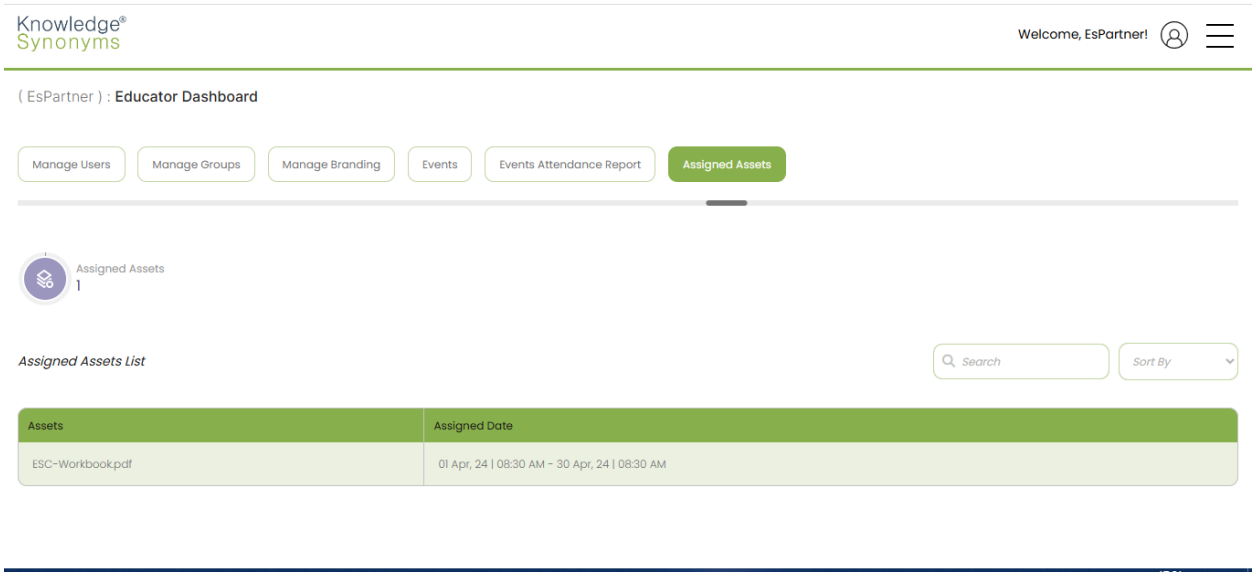
File Path: content/js/educator/event-report.js

1. **cb(start, end) {}**
   The function cb(start, end) is designed to update the content of a specific HTML element with the ID reportrange by setting the inner HTML of a span element within it to display a formatted date range.
2. **$('#group_report_range').daterangepicker({})**
   This code sets up a date range picker with various options and predefined date ranges, allowing users to easily select date ranges using a graphical interface.
3. **$('.multiselection').multiselect({})**
   This code snippet initializes a multi-select dropdown menu for elements with the class .multiselection.
4. **$('.filter_type').on('change', function() {})**
   This code controls the behavior of multiselect dropdowns based on the selected option in another dropdown, allowing users to filter data based on different criteria.

5. **resetTable = () => {}**
   The resetTable function is a JavaScript function designed to reset and clear specific DataTables on a webpage, as well as remove certain elements.
6. **$('#category').on('change', function() {})**
   This code snippet sets up an event listener for changes on a select element with the ID category. When the selected option changes, it triggers a series of actions to update another element with the ID scenario based on the selected category.
7. **$('#event').on('change', function(){})**
   The provided code is a jQuery script that attaches an event listener to an HTML element with the ID #event. This listener responds to changes in the element's value, typically a dropdown menu or input field.
8. **$("#group_performance").submit(function(e) {})**
   The provided code is a jQuery script that handles the submission of a form with the ID #group_performance. It prevents the default form submission behavior, sends an AJAX request to *'includes/educator/events.php'* to fetch data, and processes the response to populate a DataTable.
9. **$('body').on('click', '.mark_attendance', function() {})**
   This code snippet is an event listener attached to the body element that listens for clicks on elements with the class mark_attendance. It sends a POST request to the server at *'includes/ajax.php'*. It shows a success message using swal(), indicating that the attendance has been added.

Assigned Assets

Template Name: partner-assigned-assets.php



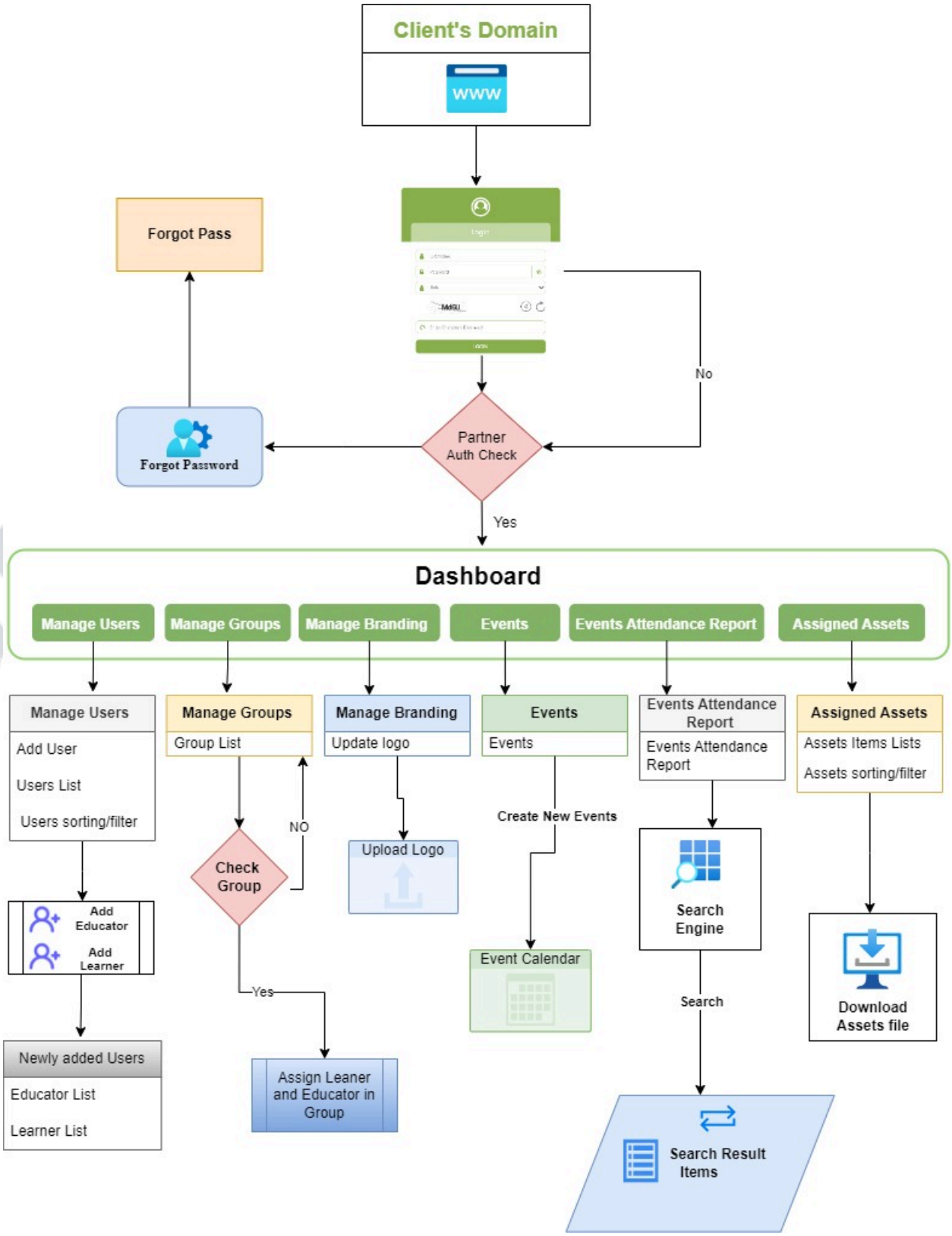
PHP Functions Used

1. **getAssigneeName($userid)**
   Global function from “db.class.php”. This function will return all the data for user as per the User ID passed in the argument.
2. **getClientPermission($client_id)**
   Global function from “db.class.php”. This function returns all the data for the permissions from table client_permissions for the give client_id.
3. **SQL Queries**
   $sqlr = "SELECT id, assets, DATE_FORMAT(start_date, '%d %b, %y | %H:%i %p') AS sdate, DATE_FORMAT(end_date, '%d %b, %y | %H:%i %p') AS edate, CASE WHEN (start_date IS NOT NULL AND end_date IS NOT NULL AND NOW() BETWEEN start_date AND end_date) OR (start_date IS NULL AND end_date IS NULL) THEN 'Enabled' ELSE 'Disabled' END AS download_status FROM assigned_assets WHERE status = 1 AND FIND_IN_SET(?, partner) $orderby";
   This SQL query selects specific columns from the assigned_assets table, formats date-time values, dynamically determines the download status based on date conditions, and filters rows based on status and partner criteria. The ordering of results is determined by the $orderby variable.

JS Functions Used

1. **getUrlVars = () => {}**
   The "getUrlVars" function extracts query parameters from the current URL and returns them as an array where each parameter is both an item and a property of the array.
2. **CircleBar = () => {}**
   The "CircleBar" function initializes circular progress bars on elements with specific classes by setting the "stroke-dashoffset" CSS property based on a data attribute.
3. **saveDownloadTrack = (id) => {}**
   This JavaScript function saveDownloadTrack sends a POST request to a server-side PHP script (*includes/process.php*) to save download tracking information.
4. **downloadFile = (filePath, fileName) => {}**
   This downloadFile function in JavaScript facilitates the downloading of a file by creating a temporary anchor element and simulating a click on it.
5. **$('.download_assets').on('click', async function(){})**
   This script sets up a click event listener for elements with the class download_assets. When clicked, it initiates a download process after displaying a confirmation dialog with terms and conditions.

Partner DFD


Client's Domain
www
Forgot Pass
No
Forgot Password
Partner Auth Check
Yes
Dashboard
Manage Users
Manage Groups
Manage Branding
Events
Events Attendance Report
Assigned Assets
Manage Users
Add User
Users List
Users sorting/filter
Manage Groups
Group List
Manage Branding
Update logo
Events
Events
Events Attendance Report
Events Attendance Report
Assigned Assets
Assets Items Lists
Assets sorting/filter
NO
Check Group
Upload Logo
Create New Events
Search Engine
Add Educator
Add Learner
Event Calendar
Yes
Search
Download Assets file
Newly added Users
Educator List
Learner List
Assign Leaner and Educator in Group
Search Result Items

Global Configuration and Settings

CSS Files & Library

Path: /content/css

a) Bootstrap.min.css

The bootstrap.min.css file provides a minified version of Bootstrap's CSS, enabling responsive design, cross-browser compatibility, and a consistent look with predefined styles for typography, forms, buttons, and components. It's essential for creating modern, mobile-friendly web pages quickly and efficiently.

b) Font-awesome.css

Font Awesome is a popular icon toolkit offering scalable vector icons for web projects. It includes a vast library of icons easily customizable with CSS. The icons are resolution-independent, ensuring they look sharp on all screen sizes, and can be styled with CSS to match the design requirements.

c) Font-awesome.min.css

The font-awesome.min.css file contains the minified version of Font Awesome's CSS, providing scalable vector icons that can be customized with CSS. It includes icons for web applications, social media, and more, allowing easy integration and styling of icons within web projects.

d) Style.css

The style.css file is a custom CSS file used to define the visual presentation of a website. It contains style rules that control the layout, colors, fonts, and overall design of web pages.

e) Responsive.css

The responsive.css file contains CSS rules specifically designed to make a website adaptable to different screen sizes and devices. It includes media queries and responsive design techniques, ensuring the layout and elements adjust seamlessly for optimal viewing on desktops, tablets, and mobile phones.

f) Daterangepicker.css

The daterangepicker.css file provides the necessary styles for the Date Range Picker component. It defines the visual appearance of the calendar interface, including the layout, colors, fonts, and interactive elements, ensuring the date picker is both visually appealing and user-friendly for selecting date ranges on web applications.

g) Owl.carousel.min.css

The owl.carousel.min.css file contains the minified CSS for Owl Carousel, a jQuery plugin. It provides styles for creating responsive, touch-enabled carousels and sliders. The file ensures smooth transitions, customizable design, and compatibility across various devices and browsers, making it easy to integrate dynamic content sliders into web projects.

h) Owl.theme.default.css

owl.theme.default.css is a stylesheet for the Owl Carousel, a jQuery plugin for creating responsive, customizable carousels and sliders. This CSS file provides default styling, ensuring the carousel elements, such as navigation buttons, pagination, and transitions, look polished and consistent across different browsers and devices.

i) Qustemplate.css

j) Questionresponsive.css

questionresponsive.css likely refers to a custom stylesheet designed to make web pages or components responsive. It contains media queries and CSS rules to adjust the layout, typography, and other styles for various screen sizes and devices, ensuring optimal user experience and readability on desktops, tablets, and mobile phones.

k) Template-updated.css

l) Jquery.fontselect.css

The jquery.fontselect.css file provides the styling for the jQuery Fontselect plugin. It defines the appearance of the font selector dropdown, ensuring a user-friendly interface for previewing and selecting fonts. The CSS handles layout, typography, and interactive states to create a seamless font selection experience in web applications.

m) Spectrum.min.css

spectrum.min.css is a minified stylesheet for Spectrum, a lightweight and customizable color picker library. This CSS file provides the necessary styles for Spectrum's interface, including the color palette, sliders, and input fields. Being minified, it ensures faster loading times and better performance for web applications.

n) Jquery.fancybox.min.css

jquery.fancybox.min.css is a minified stylesheet for Fancybox, a jQuery plugin used to create elegant, responsive lightbox pop-ups. This CSS file provides styles for displaying images, videos, and other content in an overlay that appears on top of the web page, enhancing the user experience with a sleek, modern look.

o) Jquery-ui.min.css

The jquery-ui.min.css file provides the minified CSS styles for jQuery UI, a popular user interface library. It includes predefined styles for various UI components such as buttons, dialogs, menus, and form elements. This CSS file ensures consistent design and behavior across different browsers and devices when using jQuery UI components in web applications.

p) ZoomSlider.css

The ZoomSlider.css file likely contains styles for a zoom slider component used in web interfaces. It defines the visual appearance, layout, and behavior of a slider that allows users to zoom in and out of images or content. The CSS file handles transitions, responsiveness, and other visual aspects to create a smooth and interactive zooming experience.

q) Jquery-te-1.4.0.css

jquery-te-1.4.0.css is a stylesheet for jQuery TE (Text Editor) version 1.4.0, a lightweight and customizable WYSIWYG (What You See Is What You Get) editor. This CSS file provides styles for the text editor's interface, including toolbar buttons, fonts, and formatting options, ensuring a user-friendly and visually consistent editing experience.

r) Bootstrap-responsive.css

The bootstrap-responsive.css file contains CSS rules specifically designed to make a website adaptable to different screen sizes and devices. It includes media queries and responsive design techniques, ensuring the layout and elements adjust seamlessly for optimal viewing on desktops, tablets, and mobile phones.

s) Bootstrap-editable.css

The bootstrap-editable.css file provides styles for Bootstrap Editable, a plugin that enhances form controls for inline editing. It defines the appearance of editable fields, ensuring they seamlessly integrate into Bootstrap-based web applications. The CSS file includes styles for various form elements and interactive states to facilitate user-friendly inline editing experiences.

t) Bootstrap-toggle.css

The bootstrap-toggle.css file contains styles for the Bootstrap Toggle plugin, which enhances checkboxes and radio buttons with customizable toggle switches. It defines the visual appearance, animations, and interactive behaviors of toggle switches within Bootstrap-based web applications.

u) Component.css

v) All.min.css

w) 360v.css

x) Animate.css

Animate.css is a standalone CSS library that provides a collection of premade CSS animations. It allows developers to easily add visually appealing animations to elements on web pages without needing to write complex CSS animations from scratch. The animations include fades, slides, bounces, and more, enhancing user interaction and engagement on websites.

y) Draganddrop.css

The draganddrop.css file typically contains styles for implementing drag-and-drop functionality in web applications. It defines the visual appearance of draggable elements, drop zones, and feedback during drag operations. The CSS file ensures smooth transitions, visual cues, and responsive behavior for an intuitive drag-and-drop user experience.

z) Template-updated.css

i) template-updated.css is a CSS file that likely contains updated styles for a website or web application template. It includes modifications, improvements, or customizations to the visual design, layout, colors, typography, and components of the template.

aa) App-new.css

bb) app.css

cc) Base.css

dd) Custom-timer.css

A custom-timer.css file defines the styles for a timer's appearance. It typically includes styles for the timer container, display, and control buttons. This CSS enhances visual elements, ensuring the timer is attractive and user-friendly, often with modern design elements like rounded corners, shadows, and responsive layouts.

ee) Iframe.css

ff) N-responsive.css

n-responsive.css is a custom stylesheet designed to enhance the responsiveness of a website. It contains media queries and responsive design rules to ensure the site's layout, images, and other elements adapt seamlessly to different screen sizes and devices, providing an optimal user experience across desktops, tablets, and smartphones.

gg) n-style.css

hh) Phone.css

ii) Progresscircle.css

progresscircle.css is a stylesheet for creating and styling circular progress indicators. It defines the visual appearance of circular progress bars, including their size, color, animation, and other design elements. This CSS file ensures the progress circles are displayed consistently and attractively across different web browsers and devices.

jj) Questions.css

kk) Report_page.css

ll) Select2.css

select2.css is a stylesheet for Select2, a jQuery-based replacement for select boxes. It enhances the default select elements with features like search, tagging, remote data sets, and infinite scrolling. This CSS file provides the necessary styles for the enhanced interface, ensuring a consistent and user-friendly design.

mm) Style-new.css

nn) Template-updated.css

oo) Timeline.css

pp) Yearpicker.css

yearpicker.css is a stylesheet for the YearPicker plugin, which allows users to select a year from a dropdown or calendar interface. This CSS file defines the styles for the YearPicker's appearance, including layout, fonts, and colors, ensuring it integrates seamlessly with the overall design of the web application.

qq) Zoomslider.css

zoomslider.css is a stylesheet for the ZoomSlider plugin, which creates image sliders with zooming effects. This CSS file defines styles for the slider's layout, transitions, and interactive elements, ensuring a smooth and visually appealing user experience as images zoom in and out during the sliding animation.

rr) Datatables.min.css

datatables.min.css is a minified stylesheet for DataTables, a jQuery plugin that enhances HTML tables with advanced features like pagination, search, and sorting. This CSS file provides the necessary styles for the table's appearance, including layout, fonts, and responsive design, ensuring a polished and user-friendly interface for data presentation.

Third Party JS Library

Path: /content/js/circleChart

a) circleChart.es5.min.js
circleChart.es5.min.js is the minified version of the circleChart library compatible with ES5. It provides functionality to create and display animated circular charts on web pages. The minified version ensures faster load times and improved performance, making it suitable for use in production environments.

b) circleChart.min.js
circleChart.min.js is likely a minimized JavaScript library for rendering circular charts or graphs, commonly used for visualizing data in a circular format on web pages.

Path: /content/js/daterangepicker

a) daterangepicker.js
daterangepicker.js is a JavaScript library for creating customizable, user-friendly date range pickers. It supports features like predefined ranges, single-date selection, and various display formats, enhancing date input functionality in web applications.

b) moment.min.js
moment.min.js is a compact JavaScript library that simplifies date and time manipulation in web development. It provides an easy way to parse, validate, manipulate, and format dates, as well as handle timezones effectively.

Path: /content/js/fullcalendar/fullcalendar-scheduler

a) index.global.min.js

Path: /content/js/inputmask

a) inputmask.js
inputmask.js is a JavaScript library that creates input masks for form fields, ensuring users enter data in a specific format, such as phone numbers, dates, or

credit card numbers. It enhances data validation and user experience by providing real-time feedback and preventing incorrect data entry.

b) inputmask.min.js

inputmask.min.js is the minified version of the inputmask.js library. It provides the same functionality of creating input masks for form fields to enforce specific data formats, but it's optimized for performance with a reduced file size, making it faster to load on web pages.

c) jquery.inputmask.js

jquery.inputmask.js is a jQuery plugin version of inputmask.js. It allows you to easily apply input masks to form fields using jQuery syntax, enhancing data entry by enforcing specific formats like dates or phone numbers. It simplifies the integration of input masks in jQuery-based web projects.

d) jquery.inputmask.min.js

jquery.inputmask.min.js is the minified version of the jquery.inputmask.js plugin. It offers the same functionality of applying input masks to form fields using jQuery but in a smaller, optimized file. This reduces load times and improves performance for web projects utilizing jQuery for input masking.

Path: /content/js/inputmask/blindings

a) inputmask.blinding.js

Path: /content/js/jquery-ui-1.12.1

a) jquery-ui.js

jquery-ui.js is a JavaScript library built on top of jQuery. It provides a collection of user interface interactions, effects, widgets, and themes, making it easier to build highly interactive web applications. Features include drag-and-drop, datepickers, sliders, dialogs, and more, enhancing the overall user experience.

b) jquery-ui.min.js

jquery-ui.min.js is the minified version of the jquery-ui.js library. It contains the same UI interactions, effects, widgets, and themes, but in a compressed format. This reduces file size for faster loading times and improved performance, making it ideal for production environments.

Path: /content/js/jquery-ui-1.12.1/external

a) jquery.js

jquery.js is a JavaScript library used to simplify HTML document manipulation, event handling, animation, and AJAX interactions. It simplifies JavaScript code and enhances cross-browser compatibility, widely used in web development.

Path: /content/js/model

a) bootstrap.min.js

bootstrap.min.js is a minimized JavaScript file from Bootstrap framework. It enhances web development by providing ready-to-use components like modals, carousels, tooltips, and more. It ensures responsive design and interactivity without extensive custom scripting. Used with Bootstrap CSS, it streamlines frontend development, offering a consistent UI across devices and browsers.

b) jquery.min.js

jquery.min.js is a minimized version of the jQuery library, a popular JavaScript framework. It simplifies HTML document traversing, event handling, animating, and AJAX interactions. jQuery's concise syntax and broad plugin ecosystem enable efficient DOM manipulation and cross-browser compatibility, making it integral to frontend web development for dynamic and interactive websites.

Path: /content/js/OrgChart/js

a) html2canvas.min.js

html2canvas.min.js is a minimized JavaScript library used to capture screenshots or convert HTML elements into a canvas element. It renders DOM elements, including CSS styles, into a canvas that can be saved as an image or used for further processing. This is useful for generating snapshots of web pages or creating visualizations dynamically.

b) jquery.min.js

jquery.min.js is a minimized version of the jQuery library, a popular JavaScript framework. It simplifies HTML document traversing, event handling, animating, and AJAX interactions. jQuery's concise syntax and broad plugin ecosystem enable efficient DOM manipulation and cross-browser compatibility, making it integral to frontend web development for dynamic and interactive websites.

c) jquery.mockjax.min.js

jquery.mockjax.min.js is the minified version of the jquery.mockjax library. It enables developers to mock AJAX requests in jQuery, facilitating testing and development by simulating server responses. This allows for efficient debugging and prototyping without the need for an actual backend server, enhancing development workflow.

d) jquery.orgchart.js

jquery.orgchart.js is a jQuery plugin that facilitates the creation of organizational charts. It allows developers to easily visualize hierarchical data structures, such as company staff hierarchies, with interactive features like expanding/collapsing nodes. This plugin simplifies the process of displaying complex organizational relationships in a user-friendly and visually appealing manner.

e) jspdf.min.js

jspdf.min.js is a minimized JavaScript library that enables the creation of PDF documents directly from web pages. It supports adding text, images, graphics, and other content, allowing developers to generate and customize PDF files

client-side without requiring a server. It's widely used for report generation and document export.

Path: /content/js/owlcarousel

a) owl.carousel.min.js
owl.carousel.min.js is a minified JavaScript file for Owl Carousel, a jQuery plugin enabling responsive, customizable carousels with features like touch support, autoplay, looping, and lazy loading.

Path: /content/js/Spectrum

a) spectrum.js
spectrum.js typically refers to a JavaScript library used for creating color pickers or color selectors on web pages. It allows users to choose colors from a spectrum or predefined palettes, often with options for customizing the appearance and behavior of the color picker interface.
b) spectrum.min.js
spectrum.min.js is a minimized version of the Spectrum JavaScript library.

Path: /content/js

a) articulate.js
articulate.js is a JavaScript library designed to enhance web accessibility by providing speech synthesis and recognition capabilities. It allows developers to integrate text-to-speech and voice command functionalities into web applications, improving accessibility for users with visual impairments and enabling hands-free interactions through voice commands.
b) bootstrap-datetimepicker.js
bootstrap-datetimepicker.js is a JavaScript library that extends Bootstrap to provide a customizable date and time picker. It allows users to select dates, times, or both from a calendar interface. The library supports various configuration options for format, language, and appearance, enhancing forms and improving user experience in web applications.
c) bootstrap-multiselect.js
bootstrap-multiselect.js is a JavaScript library that enhances Bootstrap's default select inputs by allowing users to select multiple options from a dropdown menu. It provides features like filtering, checkboxes, and grouping, making it easier to handle multiple selections in forms. The library is customizable and supports various configuration options to suit different needs in web applications.
d) bootstrap.3.3.7.min.js
bootstrap.3.3.7.min.js is a minimized JavaScript file from Bootstrap version 3.3.7. It includes components for responsive design and interactivity, such as modals, dropdowns, tooltips, and carousels. This file enhances web development by

providing a consistent and user-friendly interface, ensuring compatibility across various devices and browsers.

e) canvasjs.min.js

canvasjs.min.js is a minimized version of the CanvasJS library. It is used for creating interactive and responsive HTML5 charts and graphs. This JavaScript library supports various chart types, including line, bar, pie, and more, and is optimized for performance, making it suitable for data visualization in web applications.

f) chat.app.js

chat.app.js is a JavaScript file typically used to implement real-time chat functionality in web applications. It manages features like sending and receiving messages, updating the chat interface, handling user presence, and integrating with backend services or WebSocket servers to provide seamless communication between users.

g) chat.gpt.app.js

chat.gpt.app.js is a JavaScript library designed to integrate OpenAI's GPT language model into web applications. It facilitates the creation of interactive chat interfaces, enabling applications to generate human-like text responses. This library simplifies the implementation of conversational AI, enhancing user engagement and providing advanced natural language processing capabilities.

h) edu.chat.app.js

edu.chat.app.js is a JavaScript library designed for integrating chat functionalities into educational platforms. It enables real-time communication between students, teachers, and administrators, supporting features like group discussions, private messaging, and resource sharing. This enhances online learning experiences by facilitating seamless and interactive communication within educational environments.

i) go.js

go.js is a JavaScript library used for creating interactive diagrams and graphs. It provides tools for building flowcharts, org charts, mind maps, and other types of diagrams. The library supports complex layouts, custom styling, and user interactions, making it useful for visualizing structured data in web applications.

j) html2canvas.js

html2canvas.js is a JavaScript library that captures and converts HTML elements into canvas elements. It allows developers to take screenshots of web pages or specific HTML elements, rendering them as images. This is useful for generating visual content dynamically, such as snapshots, previews, or reports from web content.

k) html2canvas.js.map

html2canvas.js.map is a source map file for the html2canvas.js library. Source maps are files that map code from its compressed or compiled form back to its original source code. They enable browsers to debug and inspect minified JavaScript code, aiding developers in understanding and troubleshooting their applications more effectively.

l) html2canvas.min.js

html2canvas.min.js is a minimized JavaScript library that allows developers to capture screenshots of web pages in the browser. It renders the DOM elements of a webpage into a canvas element, enabling the creation of snapshots that can be saved as images or manipulated further. This is useful for generating previews or saving user-generated content.

m) HyperlinkText.js

n) jquery-3.3.1.min.js

jquery-3.3.1.min.js is a minimized version of jQuery library version 3.3.1. It simplifies HTML document manipulation, event handling, animation, and AJAX interactions. This library enhances cross-browser compatibility and makes it easier to create dynamic, interactive web applications with concise and readable JavaScript code.

o) jquery-sortable.js

jquery-sortable.js is a JavaScript library that extends jQuery to enable sortable functionality for HTML elements. It allows users to drag and drop items to reorder them within a list or grid. This library is useful for creating interactive and user-friendly interfaces where items need to be sorted or organized dynamically.

p) jquery.canvasjs.min.js

jquery.canvasjs.min.js is a minified JavaScript library that combines CanvasJS charting capabilities with jQuery. It allows developers to create interactive, customizable charts and graphs easily within web applications, leveraging jQuery's simplicity for DOM manipulation and event handling, while providing a wide range of chart types and features.

q) jquery.cyclotron.js

jquery.cyclotron.js is a jQuery plugin that enables a continuous, looped scrolling effect, often used for image carousels or panoramic views. It supports both mouse dragging and touch gestures, allowing users to smoothly scroll through content or images seamlessly, creating an engaging and interactive user experience.

r) jquery.fontselect.js

jquery.fontselect.js is a jQuery plugin that provides a user-friendly font selector, enabling users to browse, preview, and choose from a list of web fonts (like Google Fonts). It enhances web design by allowing easy integration and live font preview, simplifying font customization on websites.

s) jquery.simple.timer.js

jquery.simple.timer.js is a jQuery plugin that creates simple countdown timers or clocks. It is customizable and easy to integrate, allowing developers to set timers, define callback functions upon completion, and format the display of the countdown, making it useful for deadlines, events, or any time-based functionality on websites.

t) json-editor.js

json-editor.js is a JavaScript library for dynamically creating, editing, and validating JSON objects through a user-friendly web interface. It supports various

input types and provides validation, making it easier to handle JSON data in web applications.

u) jspdf.min.js

jspdf.min.js is a minified JavaScript library that allows developers to generate PDF documents directly from web pages. It supports text, images, and graphics, enabling dynamic and customizable PDF creation within client-side web applications.

v) panzoom.js

panzoom.js is a JavaScript library for panning and zooming DOM elements. It enables users to interactively zoom in and out of content or pan across large images or maps smoothly. This library simplifies the implementation of zoomable interfaces in web applications.

w) panzoom.min.js

panzoom.min.js is a compact JavaScript library that enables panning and zooming functionalities for DOM elements within web applications. It allows users to zoom in and out smoothly on content like images or maps, with support for touch gestures and customizable options for controlling zoom behavior and interaction.

x) polyfill.min.js

polyfill.min.js is a minimized JavaScript file that includes polyfills for older browsers to support modern JavaScript features and APIs that may not be natively supported. It ensures compatibility across different browsers by providing fallback implementations for functions and methods that are not available in older browser versions.

y) select2.js

select2.js is a JavaScript library that enhances HTML <select> elements with advanced features like searching, filtering, remote data fetching, and customizable styling. It provides an improved user experience for selecting options from dropdowns, making it ideal for forms and interfaces requiring rich selection capabilities.

z) Sortable.js

Sortable.js is a JavaScript library that enables drag-and-drop sorting of elements in lists or grids. It supports touch devices and works seamlessly across modern web browsers. Developers can customize sorting behavior, handle events, and create sortable interfaces effortlessly, making it essential for interactive user interfaces that involve reordering items dynamically.

aa) sweetalert.min.js

sweetalert.min.js is a JavaScript library for creating attractive and customizable modal dialogs, such as alerts, confirmations, and prompts, with a modern and user-friendly interface. It simplifies handling user interactions by providing stylish and responsive popup dialogs that can be easily integrated into web applications, enhancing user experience.

bb) sweetalert2.all.min.js

sweetalert2.all.min.js is a compact JavaScript library that extends SweetAlert2, offering enhanced features for creating customizable and visually appealing modal dialogs. It includes support for alerts, confirmations, prompts, and more, with extensive customization options like animations, themes, and callbacks. It's ideal for improving user interaction and feedback in web applications.

cc) tooltip.js

tooltip.js is a JavaScript library used to create interactive tooltips in web applications. It allows developers to easily attach tooltips to elements, providing additional information or context when users hover over them. The library typically supports customization options such as tooltip content, position, animation effects, and styling to match the application's design.

dd) validator.min.js

validator.min.js is a compact JavaScript library for client-side form validation. It provides methods to validate various input types such as email addresses, URLs, dates, and custom patterns. This library simplifies the process of ensuring data integrity and accuracy before submission, enhancing user experience by preventing incorrect data entry.

ee) ZoomSlider.js

ZoomSlider.js is a JavaScript library that enhances web applications with interactive zooming capabilities. It allows users to zoom in and out of content, typically images or maps, using a slider or similar UI element. This library simplifies the implementation of zoom functionality, providing a smooth and responsive user experience for detailed exploration of visual content.

JS Files Used

Path: /content/js/admin

a) event-report-admin.js
b) event-report.js
c) learner-report.js

Path: /content/js/educator

a) educator-events.js
b) educator-overview-dash.js
c) event-report.js

Path: /content/js/menu

a) classie.js
b) menu.js

Path: /content/js/partner

a) overview.js

Path: /content/js

a) 360.js
b) 360v-temp.js
c) 360v.js
d) admin-overview-dash.js
e) app.js
f) author-overview-dash.js
g) comment.js
h) competency-validate.js
i) custom.js
j) draganddrop.js
k) e-learning.js
l) events.js
m) heading-replace.js
n) launch-360-branching.js
o) launch-360.js
p) launch-competency.js
q) launch-sim-branching-classic-feedback.js
r) launch-sim-branching-classic.js
s) launch-sim-branching-multiple-feedback.js
t) launch-sim-branching-multiple.js
u) launch-sim-branching-video-classic-feedback.js
v) launch-sim-branching-video-multiple-feedback.js
w) launch-sim-branching-video-multiple.js
x) launch-sim-hybrid-linear-classic.js
y) launch-sim-linear-classic-360.js
z) launch-sim-linear-classic-branching-360.js
aa) launch-sim-linear-classic-branching-360v.js
bb) launch-sim-linear-classic-branching-feedback-360.js
cc) launch-sim-linear-classic-branching-feedback-360v.js
dd) launch-sim-linear-classic-feedback-360.js
ee) launch-sim-linear-classic-feedback.js
ff) launch-sim-linear-classic.js
gg) launch-sim-linear-multiple-feedback.js
hh) launch-sim-linear-multiple-video-feedback.js
ii) launch-sim-linear-multiple-video.js
jj) launch-sim-linear-multiple.js
kk) launch-sim-linear-video-classic.js
ll) launch-sim-linear-video.js
mm) launch-sim-open-response-preview.js

nn) launch-sim-open-response.js
oo) launch-sim-single-video-feedback.js
pp) launch-sim-single-video.js
qq) launch-sim.js
rr) learner-assigned-events.js
ss) learner-events-schedule.js
tt) learner-events.js
uu) learner-overview-dash.js
vv) learner-progress-report.edu.js
ww) learner-progress-report.js
xx) learner-progress-scorm-report.js
yy) loadingoverlay.min.js
zz) main.js
aaa) moreless.js
bbb) partner-events.js
ccc) proctor-api.js
ddd) progresscircle-conf.js
eee) progresscircle.js
fff) pwd-script.js
ggg) recording.js
hhh) reviewer-overview-dash.js
iii) scorm.js
jjj) script.js
kkk) sim-assignment.js
lll) sim-branding.js
mmm) sim-temp.js
nnn) template.js
ooo) timezone.js
ppp) track-assets-remediation-link.js
qqq) track-assets-sim-pages.js
rrr) track-assets.js
sss) track-feedback-assets.js
ttt) verification.js
uuu) yearpicker.js

EasySIM®