About

The Yoruba Learning App Dashboard is a central hub within the larger Yoruba Learning App, designed specifically for administrators and instructors. It provides an intuitive and efficient interface for managing user accounts, tracking learner progress, creating and organizing lessons and exercises, and accessing valuable analytics. The dashboard empowers administrators and instructors to effectively monitor and enhance the learning experience, enabling them to make data-driven decisions and ensure the successful implementation of the Yoruba Learning App's comprehensive language learning program.

✨Features

✔ Fetch API
✔ REST API
✔ SweetAlert;

✨Technologies

The following tools were used in this project:
Html
Custom css
bootstrap
JavaScript(vanilla)
SASS
Fetch API
SweetAlert
Code refactor
Icons
Font 1
Font 2

✅API Documentation

https://documenter.getpostman.com/view/16238882/UVsTq2vm#50b943d3-7346-40b4-b51a-6e692e244680

✅ Requirements

Before starting 🏁, you need to have Git and Node installed.
Knowledge of 🏁
JS Functions
Event Listeners
Conditional Statements
Event Handlers
✅ Sections
Sign Up Section

SIGN UP PAGE STEPS:

Below are the steps needed to actualize the JavaScript Code for the Sign Up section of this project.
STEP 1 – create a signup form containing name, email,password, confirm password & button
STEP 2 – use a spinner icon on your button when it is clicked
STEP 3 - Add a "onclick event" attribute to the button with an event parameter to prevent the page from refreshing when the event is called
STEP 4 - prevent the default behaviors of form buttons using .preventDefault() function.
STEP 5 - Get the inputs using getElementByID and we actually want to get the actual value of all 4 inputs in the signup page, don't forget that.
STEP 6 - At this point, call your spinner to spin when the button is clicked.
STEP 7 – get all values entered into the form
STEP 8 - Write an if statement that checks if the value of the 4 inputs are not empty strings using logical OR operator, e.g if(getName === "" || getEmail === "" ...) STEP 9 - In the if Statement block throw a sweetalert that tells the users "All fields are required"
STEP 10 - Still in the if statement set stop the spinner from spinning on the button.
STEP 11 - Write an else if that checks if the confirm password is equal to password, if it is NOT throw a sweetalert that says password does not match
STEP 12 - Still in the else if set the once again stop the spinner from spinning
STEP 13 - Now when done with all the checks above, in the last else block create a variable called signUpData and assign a form data object instance to it e.g new FormData()
STEP 14 - Now using the append() method, append the inputs to the signUpData e.g signUpData.append("name", getName);

STEP 15 - Create an object with name signReq and add a property method with value
"POST" and a property body with value signUpData
STEP 16 - Now create a variable url and assign the Signup Api to it.
STEP 17 - Now using the fetch API, do the following:
STEP 17a - Add the url and signReq to the fetch()
STEP 17b - The first .then() function should receive a response and should convert it to json() e.g (response) => response.json()
STEP 17c - The second .then() should get the result and in there we should have an if Statement that checks if the result status is "success", if it is, throw a sweetalert that says "success" showing the result message, then set a timeout of 5000 seconds that sends the users to location.href="index.html" i.e to the login page, else if all this is falsey, throw a sweet alert that shows "Registration Unsuccessful!", stop the spinner from spinning on the button.
STEP 18 - get and send the error and it's message in the .catch() method e.g (error) => console.log("error", error)

Sign In Section

Below are the steps needed to actualize the JavaScript Code for the Sign In section of this project.
STEP 1 -add an onclick event attribute to the sign in button, a spinner as well and pass an event as a parameter
STEP 2 – append a function to the sign in button and call your event.preventDefault() method to prevent the page from refreshing when the button is clicked..
STEP 3 - Add an event listener to the button with a click event and a function that has the event parameter passed into it.
STEP 4 - Prevent the default behavior of buttons using the event parameter just passed.
STEP 5 - Get the inputs and their values from your index.html or whatever you named your signin page.
STEP 6 – call your spinner when button is clicked
STEP 7 - Check if the email and password are not empty strings and throw appropriate sweet alert, also at this point call your spinner.
STEP 8a - In the else block of the if Statement above do the following:
8b - Assign a new FormData() constructor to a variable called signInData
8c - Append the value of the email input and password input to the newly created constructor e.g signInData.append("email", email)
8d - Create a new request object and add properties method:"POST" and body:signInData

8e - Create a variable called URL and add the signin API endpoint to it, Sign In API Endpoint
STEP 9 - Use the fetch API and pass in the 2 parameters needed namely URL and signReq
STEP 10 - Get the response and call the json() method on it.
STEP 11 - Get the result and create a function that console logs the result so you can know what is coming from the backend
STEP 12 - Get the object coming from the backend and store it in your local storage using the localstorage.setItem() method, it should take 2 parameters namely (1)"adminObj" and (2) JSON.stringify(result) in the second parameter we're coverting the object from the backend to json so we can store it in our local storage.
STEP 13 - Get the stored "adminObj" from the local storage and assign it to a variable called getAdminObj
STEP 14 - Write an if block that checks if the adminObj hasOwnProperty of email if so, send the user to dashboard.html e.g if (adminObj.hasOwnProperty("email")) {location.href = "dashboard.html";}
STEP 16 - Write an else statement that throws the appropriate warning Login Unsuccessful if the if Block above is falsey.
STEP 17 - Finally, catch the error that might be anywhere in this promise statements e.g .catch((error) => console.log("Error", error));

Dashboard Page

TODO 1: Re-code the function to get dashboard information that populates the cards.
TODO 2: Re-code the function that gets top three students.

HOW TO BUILD dashboard FUNCTIONALITY
STEP 1: Call the dashboard api to populate your 5 cards on the dashboard
STEP 2: add a button to the sixth card with a value of top three student
STEP 3: create a modal to show the top three student as a popup when the top three student button is clicked.
STEP 4a: display the name of the logged in admin
STEP 4b: populate the table with all students

Category Page

TODO 1: Re-code the function to get dashboard information that populates the cards.
TODO 2: Re-code the function that gets top three students.

HOW TO BUILD CATEGORY FUNCTIONALITY
STEP 1: Create a form with 2 inputs and a button, then create a function that creates categories.
STEP 2: add an onclick event attribute to the button to create a category when clicked.
STEP 3: add the spinner animation to the button to display when the button is clicked
STEP 4a: Write a conditional statement that checks if the name or image inputs are empty, if true, throw a message that says All fields required.
STEP 4b: stop your spinner from displaying.
STEP 5: Else if the name and image inputs are not empty, get the token stored on your local storage. Remember how we do that?
const authToken = localStorage.getItem("adminObj");
const tokenAcquired = JSON.parse(authToken);
const token = tokenAcquired.token;
STEP 6: Create a new Headers() and append Authorization and the Bearer ${token} to it.
STEP 7: Create a new FormData() and then append the value of the image and name inputs to it. (Check documentation for the right parameters).
STEP 8: Create a new category request object that has a method:"POST", headers and a body containing our form data.
STEP 9: Create a variable and assign the endpoint.
STEP 10: Using the fetch API, use the variable above and the category request object as parameters.
STEP 11: Get the response and use the .json() method on it.
STEP 12: Get the result and create a function that console logs the result so you can know what is coming from the backend.

Category List Dropdown

Step1: still on the category page, call the category list dropdown api
Step2: within a scrollable box, append the category name and image
Step 3: add an update and delete button to each category
Step4: wrap every category image within an anchor tag and redirect that particular image name and it's id through the url to a details page

Details category page

Step1: display the category name on the details page
Step2: create a form to create a subcategory under a category
Step3 the subcategory form will contain two inputs which are name and file image
Step4: every subcategory created under a acategory should display under the form
Step 5: append an update button on each subcategory card

End of phase 1