A. Part 1: Brand Management – CRUD operations

- Objective:
    - Build out the CREATE, RETRIEVE, UPDATE, and DELETE functionalities of *brands* on your e-commerce platform.
    - <u>Note:</u> In the previous lab, you defined **categories** that helped organise your platform's offerings under broad umbrellas (for example, "loans, savings, investments, and bill payments" for a financial platform, or "footwear, electronics, clothing" for a sales platform). Now that categories are in place, **brands** represent the more specific labels within those categories.
    - In a financial platform, if *loans* were a category, your brands might be *student loans, mortgage loans, or microloans*.
    - In a sales platform, if *footwear* were a category, your brands might be *Nike, Adidas*, and *Puma*.
    - For an aggregator platform, if *travel* were a category, your brands could be *Expedia, Skyscanner, and Kayak.*
    - For a blog-style commerce site, if *tutorials* were a category, your brands might be *Beginner tutorials, Advanced tutorials, Expert walkthroughs*.
- Deliverables:
    - Brand management functionality fully implemented.
- Tasks
    - Admin
        - brand.php
            - Using the core functions, check if the user is logged in.
            - Also check if the user is an admin.
            - If the user is not an admin, redirect to the login page.
            - RETRIEVE
                - Display brands in the system organised by their categories (only those created by the user who is logged in). Navigation and display should be intuitive and modern.
            - CREATE
                - A form that takes a brand name and allows the user to select the category it belongs to from a list of categories in the system (ID is autogenerated). All brand + category name combinations must be unique.
            - UPDATE
                - Display style is up to you, but it should collect the updated values. Only the name is editable, not the ID.
            - DELETE
                - Delete a brand.

- Actions/Functions
    - fetch_brand_action.php
        - A script that invokes the relevant function from the brand controller to fetch all the brands created by a user from the system and returns those to the caller.
    - add_brand_action.php
        - A script that receives data from the brand creation form, invokes the relevant function from the brand controller, and returns a message to the caller.
    - update_brand_action.php
        - A script that receives data from the brand update form, invokes the relevant function from the brand controller, and returns a message to the caller.
    - delete_brand_action.php
        - A script that receives an ID/name of a brand and invokes the relevant function from the brand controller to delete that brand, and returns a message to the caller.
- Classes/Models
    - brand_class.php – a class that extends database connection and contains brand methods: add brand, edit brand, delete brand, get brand, etc.
- Controllers
    - brand_controller.php – creates an instance of the brand class and runs the methods. For this lab, for example, you need an add_brand_ctr($kwargs) method to invoke the brand_class::add($args) method.
- JS
    - brand.js
        - Validate brand information, check type.
        - Asynchronously invoke the four action scripts mentioned and inform the user of the success/failure of the message using a pop-up or modal.
- index.php
    - Update your menu so it has these buttons
        - If not logged in, Register | Login
        - If logged in and an admin, Logout | Category | Brand (Brand navigates to the admin/brand.php page when clicked).
        - If logged in and not an admin, Logout

B. Part 2: Product Management – Add & Edit

- Objective:
  - Build out the CREATE and UPDATE functionalities of *products* on your e-commerce platform.
  - Note: Currently, you have both categories and brands in place. Products are the actual offerings tied to these two. The choice of whether both users and admins can add products is stylistic and platform-dependent. However, in practice, you would typically want users who sell on your platform to be designated differently in your database for ease of management, auditing, and revenue.
- Deliverables
  - Product management functionality (add and edit) fully implemented.
- Tasks
  - Admin
    - product.php
      - Using the core functions, check if the user is logged in.
      - Also check if the user is an admin (unless your design allows sellers to add products).
      - If not authorised, redirect to the login page.
      - **RETRIEVE**
        - Display products in the system organised by their categories and brands. Display should be modern, clean, and intuitive.
      - **CREATE / UPDATE**
        - Provide a form that allows adding or editing product details. You may use the same form to add and edit operations. The form should allow the user to simultaneously select the category and brand the product belongs to.
        - The following fields should be collected at a minimum:
          - Product ID is autogenerated in the database during the add. On edit, load from the database.
          - Product Category – dropdown populated from categories (value = cat_id, display = cat_name).
          - Product Brand – dropdown populated from brands (value = brand_id, display = brand_name).
          - Product Title
          - Product Price
          - Product Description

  - Product Image – upload function, store as ../images/product/image_name.png.
  - Product Keyword
- **EXTRA CREDIT:** Implement a bulk upload feature that allows uploading images in bulk.

- Actions/Functions
  - add_product_action.php
    - A script that receives data from the product creation form, invokes the relevant function from the product controller, and returns a message to the caller.
  - update_product_action.php
    - A script that receives data from the product update form, invokes the relevant function from the product controller, and returns a message to the caller.
  - upload_product_image_action.php
    - A script that constructs the new file path of product images the user uploads (instructions in the Images section of this task), stores the image in that path and stores the resolved file path in the appropriate table in the database.
    - Important: All uploads must go to the uploads/ folder that has already been created on your server by the server administrator. This folder is your only authorized upload location. You are not permitted to delete, rename, or move this folder, nor are you allowed to create new folders outside it for image storage.
    - Your script should verify that uploaded files are stored inside the uploads/ directory only, and reject any attempts to upload elsewhere.
- Classes/Models
  - product_class.php – a class that extends database connection and contains product methods: add product, edit product, get product, etc.
- Controllers
  - product_controller.php – creates an instance of the product class and runs the methods. For example, add_product_ctr($kwargs) invokes product_class::add($args).
- Images
  - uploads/
    - This is the only permitted folder for all image uploads. The folder has already been created on each server by the system administrator. Do not modify or delete it.

        - All image uploads for products, brands, and any other assets must be stored inside this directory. Any subfolders you create for organisation (for example, by user or product) must be inside the existing uploads/ folder.
        - For example, if a user with user_id = 40 is creating a product with product_id = 6, and they upload five images, your directory structure should look like:

```
uploads/u40/p6/image_1.png

uploads/u40/p6/image_2.png

uploads/u40/p6/image_3.png
```

        - This works in conjunction with the upload_product_image_action above, so think through the process carefully to ensure you do not lose track of your logic in retrieving images.
        - You may programmatically create these user or product subdirectories as needed, but only inside `uploads/`.
- JS
    - product.js
        - Validate product information (type checks, required fields).
        - Asynchronously invoke the add and update product action scripts and display success/failure messages to the user using a pop-up or modal.
- View
    - product.php
        - Display created products. Further functionality will follow in future labs.
- index.php
    - Update your menu so it has these buttons:
        - If not logged in: Register | Login
        - If logged in and an admin: Logout | Category | Brand | Add Product (direct to appropriate pages)
        - If logged in and not an admin (depending on your design): Logout