Practical 5 front-end docs

Front-end Documentation

Header Functionality

Theme Management

Theme Initialization

Trigger

- DOMContentLoaded event

Behaviour

- Checks for stored theme in order:
    1. sessionStorage.theme
    2. User object theme (from sessionStorage or localStorage)
    3. Defaults to 'light' theme
- Applies theme using setTheme()

setTheme(theme)

Parameters

- theme (String): 'light' or 'dark'

Behaviour

1. Updates document class with theme styles
2. Stores theme in:
    - sessionStorage.theme
    - User object if logged in
3. Updates UI:
    - Theme toggle checkbox
    - Theme label text
    - Logo visibility (light/dark versions)

Theme Toggle

HTML Element

- Checkbox with ID 'themeToggle'

Behaviour

- Toggles between light/dark on change

- Syncs with current theme state

Navigation

Behaviour

- Adds 'active' class to links matching current page
- Compares page names without '.php'
- Works with .nav-link and .auth-link classes

API Communication

sendRequest(body)

Request Type

- Method: POST
- Content-Type: application/json

Parameters

- body (Object): Request payload

Return

- Success: Parsed API response
- Error: Error object

Error Handling

- Returns:
  {
  status: 'error',
  message: 'Error description'
  }

Session Management

Storage Used

- sessionStorage: Temporary data
- localStorage: Persistent preferences

Stored Items

- theme: Current theme
- user: User object when logged in

Dependencies

Requirements

- Must be in PHP files requiring header.php
- Requires:
    - #themeToggle checkbox
    - #themeLabel
    - #logoLight and #logoDark
    - .nav-link and .auth-link classes

Dashboard Functionality

Initialization

Trigger

- DOMContentLoaded event

Behavior

1. Retrieves API key from sessionStorage
2. If no API key found:
    - Logs warning
    - Stops execution
3. Calls retrieveClicks() with API key

retrieveClicks(apikey)

Request Type

- Uses sendRequest() with POST method

Parameters

- apikey (String): User's API key

Request

```
{
type: 'GetStats',
apikey: 'apikey'
}
```

Return

- Success:
  ```
  {
  status: 'success',
  ```

```
data: {
clicksData: [
{
category: 'category_name',
total_clicks: number
}
],
user: {
date_registered: 'YYYY/MM/DD',
name: 'user_name'
}
}
}
```

- Error:
```
{
status: 'error',
message: 'error_message'
}
```

Behavior on Success

1. Processes click data:
    - Extracts categories into labels array
    - Extracts click counts into data_graph array
    - Stores registration date
2. Creates bar chart using Chart.js:
    - X-axis: Categories
    - Y-axis: Click counts
    - Blue-green bars
3. Updates DOM:
    - Personalized greeting
    - Registration date
    - Time since registration

timeSince(timestampStr)

Parameters

- timestampStr (String): Date in 'YYYY/MM/DD' format

Return

- Formatted time string showing duration since timestamp

Behaviour

1. Calculates time difference between now and input date
2. Returns highest two time units:
    - Years + months (if >1 year)
    - Months + days (if >1 month)
    - Days + hours (if >1 day)
    - Hours + minutes (if >1 hour)

Dependencies

Requirements

- sessionStorage must contain valid apikey
- DOM elements required:
    - #myChart (canvas for Chart.js)
    - #greeting_dash
    - #register_date
    - #registered_time
- External libraries:
    - Chart.js
    - sendRequest() function

Login Functionality

Initialization

Trigger

- DOMContentLoaded event

Behavior

1. Gets login form element by ID 'loginForm'
2. If form exists, adds submit event listener

Form Submission

Behavior

1. Prevents default form submission
2. Gets email and password values
3. Clears previous error messages

4. Validates inputs:
    - Checks both fields are filled
    - Validates email format
5. Prepares login request body:
   ```
   {
   type: 'Login',
   email: 'email',
   password: 'password'
   }
   ```

Request Handling

Request Type

- Method: POST
- Content-Type: application/json
- URL: API_Location

Response Handling

- Success (status 200):
    1. Parses JSON response
    2. On successful login:
        - Stores user data in sessionStorage
        - Stores API key in sessionStorage
        - Redirects to index.php
    3. On failed login:
        - Shows error message
- Error (status != 200):
    1. Attempts to parse error message
    2. Shows error message
- Connection Error:
  Shows "Unable to connect to server" message

Helper Functions

showError(message)

Parameters

- message (String): Error message to display

Behavior

- Displays message in element with ID 'errorMessage'

validateEmail(email)

Parameters

- email (String): Email address to validate

Return

- Boolean: true if valid email format

Behavior

- Uses regex to validate email format

Dependencies

Requirements

- DOM elements required:
    - #loginForm
    - #email
    - #password
    - #errorMessage
- Global variable:
    - API_Location (API endpoint URL)
- Server expects:
    - JSON request with Login type
    - Returns user data on success

Product Listing Functionality

Initialization

Trigger

- DOMContentLoaded event

Configuration

- PRODUCTS_PER_PAGE = 20
- currentPage = 1
- totalProducts = 0
- allProducts = []

Behavior

1. Retrieves API key from:
    - sessionStorage (apiKey/apikey)
    - localStorage (apiKey/apikey)
    - Parses from user object if available
2. Sets up DOM elements:
    - Creates pagination container
    - Adds loading indicator
3. Initializes filter elements:
    - categoryFilter
    - minPriceInput
    - maxPriceInput
    - applyFiltersBtn
    - resetFiltersBtn
    - searchInput
    - ratingFilter
    - followProductsFilter

Core Functions

fetchProducts()

Behavior

1. Shows loading state
2. Calls getAllProducts() API
3. On success:
    - Stores products in allProducts
    - Updates totalProducts count
    - Calls renderProducts()
    - Calls setupPagination()
4. On error: shows error message

fetchProductsWithFilters()

Behavior

1. Shows loading state
2. Validates filter inputs:

   - Rating between 1-5
   - Valid price numbers
   - Min price <= Max price
3. Calls getFilteredProducts() with:
   - category
   - min_price
   - max_price
   - search term
   - min_rating
   - follow flag
4. Processes response with processProductData()

renderProducts()

Behavior

1. Calculates pagination range
2. Clears product list
3. For each product:
   - Creates product card HTML
   - Formats price
   - Generates star rating
   - Adds click handler for product view
4. Handles empty results

setupPagination()

Behavior

1. Calculates total pages
2. Creates:
   - Previous button (disabled on first page)
   - Numbered page buttons
   - Next button (disabled on last page)
3. Updates current page on click

Helper Functions

makeUpdateStats(apikey, prod_id)

Request

```
{
type: 'UpdateStats',
apikey: 'api_key',
product_id: 'product_id'
}
```

Behavior

- Tracks product views via API

generateStarRating(rating)

Parameters

- rating (Number): 1-5

Return

- HTML string of star icons

showLoading()/hideLoading()

Behavior

- Toggles loading indicator visibility

showError(message)

Behavior

- Displays error message in product list

escapeHtml(unsafe)

Behavior

- Sanitizes strings for HTML output

API Functions

sendRequest(body)

Request Type

- Method: POST
- Content-Type: application/json

Behavior

- Adds API key if missing
- Handles response parsing
- Returns error object on failure

getAllProducts()

Request
{ type: 'GetAllProducts' }

getFilteredProducts()

Request Parameters

- prod_id
- brand
- category
- min_price
- max_price
- search
- store_id
- min_rating
- follow

Special Case

- When follow=1, calls filterProductsOnStoresUserFollows()

filterProductsOnStoresUserFollows()

Behavior

1. Gets followed stores via 'GetFollowing' API
2. Fetches products from each store
3. Combines results

Dependencies

Requirements

- DOM structure:
    - .product-list container
    - Filter controls (see Initialization)
- External:
    - Font Awesome for star icons
    - Chart.js (not shown but used in dashboard)
- API endpoints:
    - ../api/api.php
- Session storage for:

        - API key
        - User data

Registration Functionality

Initialization

Trigger

- DOMContentLoaded event (implied by form submission handler)

Behavior

- Attaches submit event listener to #registerForm

Form Submission

Behavior

1. Prevents default form submission
2. Gets form values:
    - name
    - email
    - password
3. Clears previous error messages
4. Validates inputs:
    - Checks all required fields are filled
5. Prepares registration request body:
   ```
   {
   type: 'Register',
   name: 'name',
   email: 'email',
   password: 'password',
   user_type: 'Customer'
   }
   ```

Request Handling

Request Type

- Method: POST
- Content-Type: application/json
- URL: http://localhost/COS221-Assignment5/api/api.php

Response Handling

- Success (status 200/201):
    1. Shows success alert
    2. Redirects to login.php
- Error (status != 200/201):
    1. Attempts to parse error message
    2. Shows error message
- Connection Error:
  Shows "Unable to connect to server" message

Helper Functions

showError(message)

Parameters

- message (String): Error message to display

Behavior

1. Displays message in #errorMessage element
2. Shows/hides error container based on message presence

Dependencies

Requirements

- DOM elements required:
    - #registerForm
    - #name
    - #email
    - #password
    - #errorMessage
- Server expects:
    - JSON request with Register type
    - Returns success/error message

Flow

1. User submits form
2. Client-side validation
3. API request sent
4. On success: redirect to login

5. On failure: show error message

Store Management Functionality

Initialization

Configuration

- API Endpoint: http://localhost/COS221-Assignment5/api/api.php
- API Key: Retrieved from sessionStorage

Behavior

1. Loads user's store and followed stores on page load
2. Sets up popup functionality for store registration

Core Functions

Store Registration

Function: registerStoreOwner()
Request:

```
{
type: 'RegisterStoreOwner',
apikey: 'api_key',
store_name: 'name',
store_url: 'url',
store_type: 'type',
registrationNo: 'number'
}
```

Behavior:

1. Validates all fields are filled
2. Submits registration request
3. On success:
    - Clears form
    - Closes popup
    - Reloads store data
4. Handles errors (existing store, etc.)

Store Display

Function: displayStores()
Behavior:

1. Creates store cards with:
    - Store info
    - Visit store button
    - Follow/unfollow button
    - Product management section
2. Sets up product CRUD operations

Product Management

Functions:

- getStoresProducts(): Fetches products for a store
- addProductToStore(): Adds new product
- editProductInStore(): Updates product
- deleteProductFromStore(): Removes product

UI Components

Popups

Functions:

- openPopup()/closePopup(): For store registration
- openProductPopup(): For product add/edit forms
- openDeletePopup(): For delete confirmation

Behavior:

- Handles form submission/cancellation
- Manages popup visibility

Follow Functionality

Function: attachFollowListeners()
Behavior:

1. Tracks followed stores in followedStoreIds array
2. Updates UI and API on follow/unfollow actions
3. Requires user to be logged in

Data Flow

1. Page load:
    - Fetch user's store

   - Fetch followed stores
2. Store registration:
   - Form submission → API call → UI update
3. Product operations:
   - Add/edit/delete → API call → Refresh product list

Dependencies

Requirements

- DOM elements:
  - #popup (registration form)
  - .store-list (store container)
  - Product popup elements
- API endpoints for all store/product operations
- sessionStorage for API key

Error Handling

- Displays alerts for API errors
- Console logs for debugging
- Validates form inputs before submission

Store Browsing Functionality

Initialization

Configuration

- API Endpoint: http://localhost/COS221-Assignment5/api/api.php
- API Key: Retrieved from sessionStorage

Behavior

1. On page load:
   - Fetches all stores
   - Fetches followed stores if logged in
2. Sets up filter event listeners

Core Functions

Store Retrieval

Function: getStores()
Request:
```
{
type: 'GetStores'
}
```

Behavior:

1. Checks for API key
2. Fetches followed stores if logged in
3. Retrieves all stores
4. Handles JSON response validation

Follow Management

Function: fetchFollowedStores()
Request:
```
{
type: 'GetFollowing',
apikey: 'api_key'
}
```

Behavior:

- Maintains followedStoreIds array
- Updates UI to reflect followed status

UI Components

Store Display

Function: displayStores()
Behavior:

1. Creates store cards with:
   - Store name and type
   - Visit store link
   - Follow/unfollow button (color-coded)
2. Shows login prompt for unauthorized follow attempts

Filtering

Function: filterStores()
Behavior:

1. Filters by:
   - Store name (text search)
   - Store type (dropdown)

        - "All" stores option

2. Real-time updates on input change

Follow Toggle

Function: attachFollowListeners()
Behavior:

1. Handles follow/unfollow actions
2. API Requests:
    - Follow: {type: 'Follow', apikey, store_id}
    - Unfollow: {type: 'Unfollow', apikey, store_id}
3. Updates:
    - Button text and color
    - followedStoreIds array

Special Features

Followed Stores Filter

Behavior:

1. Triggered by #Follow_products dropdown
2. Switches between:
    - All stores (type: 'GetStores')
    - Followed stores (type: 'GetFollowing')

Error Handling

- Console logs for API errors
- Content-type validation for responses
- Basic error alerts for follow actions

Dependencies

Requirements

- DOM elements:
    - .store-list (container)
    - #filter-input (search)
    - #filter-dropdown (type filter)
    - #Follow_products (followed filter)
- sessionStorage for API key

- Consistent API response format

Data Flow

1. Page load → Fetch stores → Display
2. Filter changes → Update display
3. Follow actions → API call → UI update
4. Followed filter → Alternate API call → Display update

User Settings Functionality

Initialization

Behavior

1. On DOMContentLoaded:
    - Loads saved theme preference
    - Applies theme to document
    - Loads price filter preferences

Core Functions

showMessage(msg, isError)

Parameters:

- msg: Message text
- isError: Boolean for error styling

Behavior:

- Displays message in #settingsMessage
- Applies red/green color based on error status

Form Submission

Trigger: #settingsForm submit

Validation

1. Requires:
    - Valid API key in sessionStorage
    - Complete email/password fields when changing credentials
    - Valid email format
    - Password length >= 8 characters

Request Handling

Request Type: POST
Content-Type: application/json
Endpoint: api/api.php

Request Body (SavePreferences):

```
{
type: 'SavePreferences',
apikey: 'api_key',
theme: 'light|dark',
min_price: number|null,
max_price: number|null,
current_email?: string,
current_password?: string,
new_email?: string,
new_password?: string
}
```

Response Handling:

- Success: Updates local storage and shows confirmation
- Error: Displays error message

Client-Side Storage

Stores in:

1. localStorage:
   - Theme preference
   - Price range preferences
2. sessionStorage:
   - Current theme
   - API key

UI Updates

1. Theme:
   - Toggles dark-theme class
   - Updates theme toggle checkbox
2. Price Filter:
   - Formats and displays saved range
3. Messages:
   - Shows success/error feedback

Error Handling

1. Client-side:

    - Form validation messages
    - API key check
2. Server-side:
    - Error message display
    - Console logging

Dependencies

Required DOM Elements:

- #settingsForm
- #theme (select)
- #price-filter
- #settingsMessage
- Email/password fields (when present)

Storage Requirements:

- sessionStorage.apikey
- localStorage.theme
- localStorage.preferences

API Expectations:

- Accepts SavePreferences requests
- Returns {status, message, data?}

Product View Functionality

Initialization

Configuration

- API Endpoint: http://localhost/COS221-Assignment5/api/api.php
- Data Source: sessionStorage.selectedProduct

Behavior

1. On page load:
    - Retrieves product data from sessionStorage
    - Populates product details (title, price, image)
    - Sets up external store link (if available)
    - Initializes rating form

Core Features

Product Display

Data Elements:

- Title
- Price (formatted with 'R' prefix)
- Thumbnail image (fallback to placeholder)
- External product link (hidden if unavailable)

Rating System

Components:

1. Rating Form:
   - Hidden by default
   - Toggle visibility with "Add Rating" button
   - Handles both new ratings and edits
2. Rating Submission:
   Request Types:

- SubmitRating (new ratings)
- EditRating (updates)

Request Body:

```
{
studentnum: "u24566170",
apikey: "api_key",
type: "SubmitRating|EditRating",
prod_id: "product_id",
rating: number,
comment: "text"
}
```

3. Rating Display:

- Calculates average rating (star visualization)
- Shows individual reviews with:
  - User name
  - Star rating
  - Comment text
- Owner-specific controls:
  - Edit button (converts form to edit mode)

    - Delete button

Rating Management

Functions:

- loadRatings(): Fetches and displays all ratings
- submitRating(): Handles form submission
- Delete Functionality:
    - Confirmation dialog
    - DeleteRating API request

API Interactions

GetRatings

Request:

```
{
type: "GetRatings",
prod_id: "product_id"
}
```

Response Handling:

- Success: Updates rating display
- Error: Logs to console

DeleteRating

Request:

```
{
type: "DeleteRating",
apikey: "api_key",
rating_id: "rating_id"
}
```

Behavior:

- Refreshes rating list after deletion

UI Elements

Required DOM Elements:

- #product-title
- #product-price
- #main-image
- #external-link
- .Add_rating (button)

- #ratingForm
- .All_Rating (container)
- #average-rating

Error Handling

- Form validation alerts
- API error logging
- User-friendly messages for:
    - Missing product data
    - Rating submission failures
    - Deletion errors

Data Flow

1. Page Load:
    - Product data → Populate UI
    - API call → Load ratings
2. Rating Actions:
    - Add → Show form
    - Submit → API call → Refresh list
    - Edit → Convert form → API call
    - Delete → Confirm → API call → Refresh

Dependencies

- sessionStorage:
    - selectedProduct (required)
    - apikey (required for mutations)
    - user (for ownership checks)
- Consistent API response format:
  { status, message, data }