How BloodConnect application works

BloodConnect: How Everything Works

Your BloodConnect application is a web-based system that uses a combination of HTML, CSS, JavaScript (frontend) and PHP, MySQL (backend) to connect blood donors and recipients.

---

1. Core Components & Technologies

* Frontend (User Interface):

  * HTML: Structures the web pages (forms, buttons, text).

  * CSS: Styles the pages, making them visually appealing and responsive.

  * JavaScript (script.js): Handles user interactions, form submissions (using fetch API), dynamic content loading, and chart rendering (using Chart.js).

* Backend (Server-side Logic & Database Interaction):

  * PHP: Processes form submissions, interacts with the MySQL database, and serves dynamic content.

  * MySQL: The database that stores all application data (user accounts, blood requests, notifications).

  * Sessions: PHP sessions (session_start(), $_SESSION) are used to maintain user login state across different pages.

---

2. User Journey & Feature Breakdown

A. Registration Process

1. index.html / login.html -> register.html (User Action)

* A new user clicks the "Register" link from the home or login page.

2. register.html (Frontend)

* Presents a form where the user enters their full name, email, phone, age, blood type, city, address, and password.

* The form's action attribute is set to backend/register.php.

3. script.js (Frontend - initializeRegisterForm, handleRegisterSubmit)

* When the user clicks "Register" on the form, JavaScript intercepts the submission (e.preventDefault()).

* It performs client-side validation (e.g., passwords match, password length).

* It uses the fetch API to send the form data (via submitForm utility) to backend/register.php using a POST request.

4. backend/register.php (Backend)

* Receives the POST data from the form.

* Performs server-side validation (e.g., checks if email already exists, validates age).

* Hashes the user's password using password_hash() for security.

* Inserts the new user's details into the users table in the MySQL database.

* Returns a JSON response ({"success": true, "message": "...", "redirect": "login.html"}) to the frontend.

5. script.js (Frontend - submitForm utility)

* Receives the JSON response.

* If success is true, it shows a success alert and redirects the user to login.html. If success is false, it shows an error alert.

B. Login Process

1. index.html / register.html -> login.html (User Action)

   * An existing user clicks the "Login" link.

2. login.html (Frontend)

   * Presents a form for email/phone and password.

   * The form's action attribute is set to backend/login.php.

3. script.js (Frontend - initializeLoginForm, handleLoginSubmit)

   * Interception form submission.

   * Sends form data via fetch API (submitForm utility) to backend/login.php.

4. backend/login.php (Backend)

   * Receives the POST data.

   * Queries the users table to find a user matching the provided email/phone.

   * Verifies the password using password_verify() against the stored hashed password.

   * If credentials are valid:

      * Sets $_SESSION['user_id'] and $_SESSION['authenticated'] to establish the user's login session.

      * Returns a JSON response ({"success": true, "message": "...", "redirect": "dashboard.php"}).

   * If credentials are invalid, returns an error JSON.

5. script.js (Frontend - submitForm utility)

   * Receives the JSON response.

   * If success is true, shows a success alert and redirects the user to dashboard.php.

C. Dashboard Page (dashboard.php)

1. Page Load (dashboard.php)

* The dashboard.php file starts with a PHP block: require_once 'backend/cleanup.php';.

* Automatic Cleanup: This immediately executes the cleanup.php script, which connects to the database and deletes any expired blood requests (critical > 24h, urgent > 3 days, normal > 7 days) based on their request_date.

* The rest of the HTML for the dashboard is then rendered.

2. script.js (Frontend - initializeDashboard, checkAuthStatus, loadUserProfile, loadBloodRequests, loadDashboardCharts)

* When dashboard.php loads, script.js runs checkAuthStatus(). This fetches from backend/check_auth.php to verify the user's session. If not authenticated, it redirects to login.html.

* loadUserProfile(): Fetches the logged-in user's profile data from backend/get_user_profile.php and displays it in the "Your Profile" section (which is now on profile.html).

* loadDashboardCharts(): Fetches blood type distribution data from backend/get_dashboard_data.php and renders the "Available Donors Chart" using Chart.js.

* loadBloodRequests(): Fetches all active blood requests from backend/get_blood_requests.php and displays them in a responsive grid layout.

3. "View/Edit Profile" Button (User Action)

* Located in the "Quick Actions" card.

* href="profile.html": Directly navigates the user to the dedicated profile page.

4. "Request Blood" Button (User Action)

* Located in the "Quick Actions" card.

* href="request.html": Directly navigates the user to the blood request form page.

5. "Find Donors" Button (User Action)

* Located in the "Quick Actions" card.

* href="search.html": Directly navigates the user to the donor search page.

6. "Logout" Button (User Action)

* Located in the navigation bar.

* script.js intercepts the click (e.preventDefault()).

* window.location.href = 'backend/logout.php';: Directly navigates the browser to the logout script.

7. backend/logout.php (Backend)

* Destroys the PHP session (session_unset(), session_destroy()).

* Redirects the browser to index.html using a header("Location: ...") command.

D. Profile Page (profile.html)

1. Page Load (profile.html)

* script.js (initializeProfilePage) is called.

* loadUserProfile(): Fetches the logged-in user's profile data from backend/get_user_profile.php and displays it.

2. "Edit Profile" Button (User Action)

* script.js (editProfile) listens for clicks.

* It fetches the current user data from backend/get_user_profile.php again to pre-fill the modal form.

* It opens the editProfileModal.

3. "Update Profile" Form Submission (User Action - within modal)

* The form's action is backend/update_profile.php.

* script.js (handleUpdateProfileSubmit) intercepts the submission and sends data via submitForm utility.

4. backend/update_profile.php (Backend)

* Receives POST data.

* Performs validation and ensures the user is logged in.

* Updates the user's information in the users table.

* Returns a JSON response.

5. script.js (Frontend - submitForm utility)

* If successful, closes the modal and calls loadUserProfile() again to refresh the displayed profile data.

E. Request Blood Page (request.html)

1. Page Load (request.html)

* script.js (initializeRequestForm) sets up the form listener.

2. Form Submission (User Action)

* The form's action is backend/request_blood.php.

* script.js (handleRequestSubmit) intercepts the submission and sends data via submitForm utility.

3. backend/request_blood.php (Backend)

* Receives POST data.

* Performs validation.

* Inserts the new blood request into the blood_requests table.

* Notification Logic: (As per previous discussion, if implemented) It then searches the users table for donors matching the blood type and city/address criteria and creates entries in the notifications table for them.

* Returns a JSON response.

4. script.js (Frontend - submitForm utility)

* If successful, shows a success alert and clears the form.

F. Search Donors Page (search.html)

1. Page Load (search.html)

* script.js (initializeSearchPage) sets up the form listener.

2. Search Form Submission (User Action)

* The form's action is backend/search_donors.php.

* script.js (handleSearchFormSubmit) intercepts the submission and sends data via submitForm utility.

3. backend/search_donors.php (Backend)

* Receives POST data (blood type, city).

* Queries the users table, filtering by blood type, and by city/address (using LIKE).

* Crucially, it also filters to only include donors whose last_donation_date is NULL or is more than 56 days ago.

* Returns a JSON response containing the list of eligible donors.

4. script.js (Frontend - handleSearchFormSubmit)

* Receives the JSON response.

* Dynamically generates HTML "donor cards" for each matching donor and displays them on the page, including their name, blood type, city, address, and phone number.

---

3. Database Tables

* users table: Stores all registered user accounts, including their personal details, blood type, address, password hash, and last donation date.

* blood_requests table: Stores all submitted blood requests, including patient details, required blood type, hospital information, contact details, urgency level, and request date.

* notifications table: (If implemented) Stores notifications for users about new blood requests that match their profile.