ChatGPT

LiqLearns Project Overview

Introduction

LiqLearns is a comprehensive **learning platform** built with React, Supabase and Stripe (soon to be removed). The platform provides role-based experiences for students, teachers, administrators and executives (CEO). Its main features include personalized dashboards, study rooms, a marketplace for learning materials, event management, community interaction, help and support, and administrative tools. Supabase backs the database, authentication and real-time infrastructure, while the UI is built with React and TailwindCSS.

The project's codebase is organized into multiple directories: `src/components` contains reusable UI elements; `src/pages` holds each page of the application (e.g., dashboard, marketplace, login); `src/contexts` provides React contexts for state management (such as `AuthContext`); `services` contains service clients for Supabase, API calls and Stripe; `supabase` holds database schema, edge functions and policies. The `public` folder holds static assets, and `.env` houses environment variables.

During testing, multiple issues were discovered: blank modals on the student dashboard, failing study room joins, missing course details in the Quest page, mismatched category counts in the marketplace, unresponsive teacher dashboard actions and 404 pages in admin areas. Additional problems included unstable real-time connections due to Content-Security-Policy (CSP) restrictions and the need to remove Stripe from the sign-up flow. This report synthesizes the instructions, code and fixes discussed with the user to offer a detailed understanding of each feature, the underlying code and how to address these issues.

---

1. Student Role: Dashboard and Learning Flow

1.1 Dashboard Structure

The **Student Dashboard** is the landing page after logging in or signing up. It is implemented in `src/pages/role-based-dashboard-hub/student/StudentDashboard.tsx`. The dashboard displays:

- **Total Lessons**, **Total Badges**, **Total XP**, **Total Certificates** and **Current Streak** metrics in cards at the top of the page. Each card is expected to open a modal showing further details when clicked. However, during testing these modals appeared blank because the data fetch failed and no fallback was implemented [1].
- **Today's Quest** section, a checklist of tasks for the day (e.g., "Complete daily quiz", "Watch a video lesson"). Each item can be marked as completed, awarding points or streaks.
- A **Progress bar** summarizing course progress. It uses React `useEffect` hooks to fetch progress data from the `student_stats` table in Supabase and update automatically via real-time subscription. A key bug was found: the original code placed a `useState` call inside a helper

function `renderProgressSection()`, violating React rules and causing unresponsive updates [2]. The fix is to move the state hook to the component root.
- **Embedded Marketplace**. During project evolution, the marketplace was integrated directly into the Student Dashboard rather than a separate page. A `MarketplaceIcon` component with 13 custom icons (interactive, template, tutorial, lecture, reference, spaced-repetition, gamified, memory, audio, video, books, flashcards, games) is used to visually tag marketplace items [3].
- **Student Stats**. A new `student_stats` table is proposed (and partially implemented) to store real-time gamification metrics: XP, gold, streaks, levels, aura points, total courses, completed courses and last active date. A Supabase **edge function** called `student-stats` can be used to fetch or create these stats and return them to the front end [4]. The front end should poll this function or use Realtime subscription to avoid blank values. Mock data must be removed [5].

1.2 Quest and Course Details

The **Quest** page lists courses grouped by learning paths (e.g., Amharic Language, Ethiopian Culture, Mathematics). Each course card should open a side panel containing:

- Course description.
- Upcoming **Live Classes** with `Set Reminder` buttons.
- **Course Materials** (PDF, audio, video, notes, assignments, homework, movies).
- **Learning Tools** categories (Audio, Video, Notes, Assignments, Homework, Movies).
- **AI Insights** with a `View Detailed Analysis` link.

During testing, selecting a course produced a “Course not found in database” error and no materials displayed [6]. This indicates that either the course slug is incorrect or the database lacks the course record. To fix it:

1. Verify the **Courses** table in Supabase has entries for each course slug used in the front end.
2. Ensure the front end fetches courses by their `id` or `slug` rather than relying on static arrays.
3. Remove any hard-coded lists and load data directly from Supabase.
4. Provide error messages when a course is missing instead of leaving the modal blank.

1.3 Study Rooms

Study rooms allow students to collaborate via video chat or chat rooms. They are listed on the **Study Rooms** page. Each room card includes a **Join Room** button. Issues discovered:

- **Failed to join room** error appears for all rooms [7]. A “Retry” button simply refreshes the list.
- **Create Room** button does nothing when clicked [8].

Possible reasons and fixes:

1. **Real-time WebSocket**: The front end uses `supabase.realtime` to join channels. If CSP restricts WebSocket or the environment variables for Realtime aren’t set correctly, connections fail. Update `vercel.json` or server headers to allow `wss://*.supabase.co` [9].
2. **Row Level Security (RLS)**: The `study_rooms` table has policies like `students_view_age_appropriate_rooms`, `students_create_study_rooms` and

`students_update_own_rooms` [10] . If the age categories or user ages aren't set, the SELECT policy denies access. Add a default age group or adjust the policy.

3. **Missing membership table**: Joining may insert into a `study_room_members` or similar table. Without proper RLS policies or triggers, insertion fails. Check Supabase for membership table and add an `INSERT` policy such as `students_join_room` with `user_id = auth.uid()`.
4. **Create Room functionality**: The UI likely calls an edge function or API to create a room. If this function isn't implemented or RLS denies insertion, the button is inert. Implement the API and add RLS.

1.4 Event Calendar

The **Events** section provides a calendar and a **Create Event** modal. The modal includes fields for title, description, start time, end time, event type and a public/private toggle. The bug observed: date/time pickers didn't open when clicking the date fields [11] . This is often due to missing date picker library CSS or script or z-index issues. To fix:

- Ensure the date picker component is imported from its library (e.g., `react-datepicker` or `@headlessui/react` date picker) and its CSS is loaded in `index.css`.
- Remove any `overflow: hidden` on modal container that might hide the picker.
- Provide proper `z-index` so the picker appears above the modal.

1.5 Marketplace Embedded in Dashboard

A major evolution was integrating the **Marketplace** into the Student Dashboard. Initially, the marketplace had its own page (`marketplace-hub`). After user feedback, the team removed the separate route and embedded a marketplace widget in the dashboard. Steps involved:

1. **Create** `EmbeddedMarketplace` **component**: The new component fetches items from the `marketplace_items` table and displays them with tag icons. It can filter by tags and category [12] .
2. **Tagging system**: The `marketplace_items` table gained a `tags` array column and triggers to populate tags based on product names and descriptions. SQL migrations were written to update existing items (e.g., assign 'interactive' tag to interactive courses, 'template' to spreadsheets and forms, 'audio' to podcasts) [13] .
3. **Marketplace Icons**: The `MarketplaceIcon` component uses a `switch` statement to return appropriate SVG icons for each tag [3] .
4. **UI integration**: The Student Dashboard now renders the marketplace grid at the bottom. Each item displays up to two tags with icons. The route `marketplace-hub` was removed from `src/Routes.tsx` [12] .
5. **Cache invalidation**: Vercel caching caused outdated versions to persist. An `APP_VERSION` constant is added to the code so that each deploy changes the version, forcing the browser to fetch fresh assets [3] . Hard reload (`Ctrl+Shift+R`) and incognito mode may be required to see updates.

1.6 Community and Help

The **Community** page offers group chats and a community wall. Creating a group doesn't open a modal, and clicking existing groups yields nothing [14]. This likely means the API to manage groups isn't implemented. To fix:

- Add a `chat_groups` table and corresponding API for creating and joining groups.
- Populate the Community page with data from Supabase and enable posting messages using a `community_posts` table.

The **Help** center includes FAQs and a ticket system. The **Submit a Ticket** modal collects title, category, priority and description. After filling in, the `Create Ticket` button becomes active [15]. Tickets appear in `My Tickets` with conversation threads [16].

1.7 Settings

The **Settings** page allows editing profile information, security, notifications, language & region, privacy and learning goals. Key features include:

- Uploading a profile picture.
- Changing password.
- Toggling notification preferences.
- Selecting primary language and timezone.
- Setting privacy levels (public, private, community only).
- Viewing GDPR/Privacy options with data export and account deletion [17].
- Setting learning goals (daily time, topics) [18].

No major issues were found here, but saving changes must trigger updates to `user_profiles` and `student_profiles` tables via Supabase.

---

2. Teacher Role: Managing Classes and Content

2.1 Teacher Dashboard

Teachers have a dedicated dashboard (e.g., `src/pages/role-based-dashboard-hub/teacher/TeacherDashboard.tsx`) that displays:

- **Total Students**, **Active Classes**, **Assignments Graded**, **Upcoming Sessions**. These metrics should be loaded from the `classes`, `student_profiles`, and `assignments` tables. However, clicking **Manage Class** or **Create Class** does nothing [19], and editing students fails [20]. This indicates missing or unimplemented handlers.

- **My Classes** section lists courses taught by the teacher, including schedules. None of the actions (Manage, Edit, Add Student) work because there are no API endpoints to support them [21].

2.2 Students & Reports

- **Student Management** lists students with progress bars and statuses. View Profile and Add Student do nothing. To fix, implement pages/modals for viewing student profiles and adding them to classes. Use Supabase RLS to ensure teachers can only see students in their courses.

- **Reports** should show performance reports (average class performance, attendance, assignments completed). An `Export Report` button is present but nonfunctional [22]. This requires generating CSV/Excel files from Supabase data and triggering file downloads.

2.3 Schedule & Store

- **Schedule** page displays a calendar of upcoming sessions. The `Add Session` button does nothing [23]. A modal should appear to set session details and store them in a `sessions` table.

- **Store** allows teachers to upload products (courses, worksheets, videos) and manage inventory. Buttons like `Add Product`, `Edit` and `Delete` are inert [24]. Implementation should include a form for product details, file uploads via Supabase Storage, and linking items to teacher IDs.

2.4 Help & Settings

The **Help** page shows FAQs but clicking questions does not expand answers [25]; fix by toggling collapsed states and adding event handlers. The **Settings** page replicates the student settings with additional fields like payment information or subscription rates [26].

---

3. Admin and CEO Roles

3.1 Admin Dashboard

Admins view overall platform metrics: total users, total teachers, total students, active users today, active courses and system alerts. They can manage users, content, finances, analytics and store. Issues found:

- **User Management** page lists users but provides no actions (edit, delete, assign roles) [27]. To fix: add actions like edit profile, reset password and deactivate account.

- **Content Management** link labeled “Open Content Management Hub” directs to a 404 page [28]. The route needs to be added to `src/Routes.tsx` and the content management page created.

- Many sections (Financial, Analytics, Store, Events, Approvals, Support) have side navigation items but no implementation. Each needs to be built and wired to the backend. For example, **Financial** can show revenue, payout schedules; **Analytics** can show user engagement; **Store** can list all marketplace items; **Approvals** can manage content submitted by teachers.

3.2 CEO Role

The CEO dashboard displays high-level metrics, including revenue, total users, students, tutors and an embed of the landing page demo video [29]. Additional sections may include:

- **Course Management** to oversee all courses, approve or reject new courses.
- **Business Analytics** to track revenue growth, user acquisition and churn.
- **Financial Overview** to see gross revenue, operating costs, profit margins.
- **Organization & Growth Metrics** for team structure and expansion.

These sections must be implemented similarly to the admin sections, with appropriate data queries and RLS policies.

---

4. Sign-Up and Authentication Flow

4.1 Existing Flow

The sign-up process is a multi-step wizard located in `src/pages/login/index.tsx`. Steps include:

1. **Role Selection**: Student, Teacher, Support, Admin. Role choice determines subsequent questions.
2. **General Questions**: Collects full name, username, email, phone and sponsor username (for referral system). It uses asynchronous functions `checkUsernameAvailability` and `checkSponsorUsername` to validate entries by calling an edge function `/functions/v1/check-username` [30].
3. **Two-Factor Verification**: Sends a code via phone or email. A verification input appears after sending the code.
4. **Policy Agreement**: Accept terms, conditions and privacy policies (e.g., COPPA for minors). The user must confirm they are above 13 years old and the content is age appropriate.
5. **Approval Confirmation**: Notifies the user that their application is submitted. For certain roles (e.g., teacher or admin), manual approval may be required [31].

Originally, there was a **Subscription/Payment step** that integrated Stripe to collect payment information. The `AuthContext.tsx` file imported `createStripeCustomer` from `../services/stripeService` and, after a successful `supabase.auth.signUp`, called `createStripeCustomer(userId, email, fullName)` [32]. The wizard also contained `PaymentStep.tsx` and `SubscriptionStep.tsx` components. These steps complicated sign-up and were removed following user feedback and to save tokens. The Stripe integration code must be fully removed.

4.2 Removing Stripe Integration

To remove Stripe from sign-up:

1. **Comment out or delete** `import { createStripeCustomer }` **in** `AuthContext.tsx`.
2. **Remove the call to** `createStripeCustomer` within the `signUp` function. This prevents Stripe customer creation after sign-up.

3. **Insert profile creation logic**: After a successful `supabase.auth.signUp`, the code should insert a new row into `user_profiles` and `student_profiles` (when role is student). For students, default values such as `subscription_plan: 'free_trial'` and `trial_end_date: NOW() + 14 days` are inserted. Use Supabase service role (via an edge function or direct insert) to bypass RLS restrictions.
4. **Remove unused components**: Delete or ignore `SubscriptionStep.tsx` and `PaymentStep.tsx` to keep the codebase clean.

Due to editing instability in Rocket, the actual modifications were challenging. The intention is clearly documented: new users should be created in the database with a free trial and no Stripe integration, and sign-up should be simplified accordingly [33].

4.3 Authentication Context

The `AuthContext` manages user sessions and exposes `signIn`, `signUp`, `signOut`, `sendTwoFactorCode`, `verifyTwoFactorCode` and `sendEmailOTP`. It uses `supabase.auth.getSession()` to restore sessions and listens for `onAuthStateChange` events. After removing Stripe, `signUp` should look like this (pseudo-code):

```
async function signUp(email: string, password: string, options?: { data: any })
{
  const { data, error } = await supabase.auth.signUp({ email, password,
options });
  if (error) return { error };
  // Create user_profile
  const userId = data.user.id;
  await supabase.from('user_profiles').insert({
    id: userId,
    email,
    full_name: options?.data?.full_name,
    username: options?.data?.username,
    role: options?.data?.role,
    phone: options?.data?.phone,
    sponsor_username: options?.data?.sponsor_username
  });
  if (options?.data?.role === 'student') {
    const trialEnd = new Date();
    trialEnd.setDate(trialEnd.getDate() + 14);
    await supabase.from('student_profiles').insert({
      id: userId,
      subscription_plan: 'free_trial',
      trial_end_date: trialEnd,
      has_active_subscription: false
    });
  }
```

```
  return { data };
}
```

This code must run in a secure context (server side or via edge function) because RLS may restrict client-side inserts. You can create a Supabase Edge Function `create-user-profile` that performs these inserts using `supabaseServiceRoleKey` and call it from the front end.

4.4 Login Flow

Login is handled by `LoginForm.tsx`, which uses `supabase.auth.signInWithPassword` and `useNavigate` to redirect users to the role-based dashboard after a successful sign-in [30]. If the credentials are invalid, an error message appears. The login form includes a **Remember Me** checkbox, a **Forgot Password?** link and a **Create Account** link to switch to the sign-up wizard.

---

5. Real-Time Connections and Environment Variables

5.1 Content Security Policy (CSP)

The site uses Vercel for deployment. To enable Supabase Real-time, the CSP must allow WebSocket connections to `wss://*.supabase.co` and API requests to `https://*.supabase.co`. Early messages indicate connection failures due to missing CSP headers. The fix is to add a `vercel.json` or `_headers` file in `public/` that includes:

```
{
  "headers": [
    {
      "source": "/(.*)",
      "headers": [
        {
          "key": "Content-Security-Policy",
          "value": "default-src 'self' https://*.supabase.co https://
static.rocket.new; connect-src 'self' https://*.supabase.co wss://*.supabase.co;
script-src 'self' 'unsafe-inline' https://*.supabase.co;"
        }
      ]
    }
  ]
}
```

This allows the front end to subscribe to real-time changes. Without it, Supabase will log errors about blocked connections [9].

5.2 Environment Variables

Different environments (preview vs. production) have separate Supabase URLs and keys. Many issues, such as missing data or failing logins, arise from the front end referencing the wrong environment variables. To fix:

- Ensure `.env` in the project and environment variables in Vercel are set correctly for `VITE_SUPABASE_URL` and `VITE_SUPABASE_ANON_KEY` [34].
- Remove fallback logic (`mockData`, `mockStats`) from code; rely solely on real data. Use environment variables to differentiate local vs. production.

5.3 Real-Time Data Handling

The Student Dashboard uses the `supabase.realtime` client to subscribe to tables like `student_stats`. When the connection is offline, the UI should show an offline indicator and fall back to HTTP polling. A sample implementation:

```
const [stats, setStats] = useState<Stats | null>(null);
const [connection, setConnection] = useState<'offline' | 'connecting' |
'online'>('connecting');
useEffect(() => {
  const bootstrap = async () => {
    const { data } = await supabase.from('student_stats').select('*').eq('id',
userId).single();
    setStats(data);
  };
  bootstrap().then(() => setConnection('online')).catch(() =>
setConnection('offline'));
  const subscription = supabase.channel('public:student_stats')
    .on('postgres_changes', { event: '*', schema: 'public', table:
'student_stats', filter: `id=eq.${userId}` }, (payload) => {
      setStats(payload.new);
    })
    .subscribe();
  return () => {
    subscription.unsubscribe();
  };
}, [userId]);
```

If `connection` is offline, the component can display a yellow WiFi icon and continue polling every 5 seconds using `setInterval` [35].

---

6. Database Structure and RLS Policies

6.1 Tables

The Supabase project for LiqLearns contains numerous tables. Important ones include:

- **user_profiles**: Stores user info (id, email, username, full_name, phone, role, sponsor_username, date_of_birth, profile_picture_url, timezone, country, etc.). Inserted when a user signs up.
- **student_profiles**: Extends `user_profiles` for students, including `subscription_plan`, `trial_end_date`, `subscription_start_date`, `subscription_end_date`, `has_active_subscription`, `parental_consent`, etc. [36].
- **teacher_profiles**: Contains teacher-specific details such as biography, skills and rating.
- **admin_profiles** and **ceo_profiles**: For admin and CEO roles, storing additional data.
- **courses**: Each course has `id`, `title`, `description`, `lesson_type` and other metadata [37].
- **lessons**: A lesson belongs to a course and contains content (video, audio, PDF) and type.
- **marketplace_items**: Each item has `id`, `title`, `description`, `price`, `point_cost`, `creator_id`, `tags` (array) and `category` (enum). The `tags` column was added to allow flexible filtering.
- **subscription_plans**: Holds subscription tiers (Basic, Standard, Pro, Elite, Premium) with monthly/yearly prices [38].
- **student_stats**: Proposed to store gamification stats such as XP, gold, streak and level 【599674900694139†screenshot】.
- **study_rooms**: Contains room `id`, `name`, `description`, `age_group`, `creator_id`. Additional tables like `study_room_members` might exist to track participants.
- **events**: Each event has `title`, `description`, `start_time`, `end_time`, `public`, `creator_id`, `type` (class, group study, etc.).
- **support_tickets**: Keeps track of help tickets with `title`, `category`, `priority`, `status`, etc.
- **notifications**: Stores notifications for real-time updates.

6.2 Policies

Supabase uses **Row Level Security (RLS)** to control access to tables. Example policies:

- `admins_view_all_student_profiles`: Allows admins to `SELECT` from `student_profiles` regardless of row content [39].
- `students_manage_own_student_profiles`: Allows students to `SELECT` or `UPDATE` only their profile where `id = auth.uid()` [39].
- `students_view_age_appropriate_rooms`: For `study_rooms`, allows students to view rooms within their age group category [10].

When features fail, it is often because RLS denies access. Always check policies to ensure the correct role/row filter is applied.

---

7. Code Structure and Key Components

The project is modular. Understanding each file helps maintain the code. Some important files:

- `src/App.tsx`: Configures React Router and imports `AuthProvider`. It defines routes for each page (e.g., `/role-based-dashboard-hub/:role`, `/login`, `/signup`, etc.). After the marketplace integration, the `marketplace-hub` route is removed.
- `src/contexts/AuthContext.tsx`: Provides authentication context. It wraps the app and exposes `signIn`, `signUp`, `signOut`, etc. It also maintains `authLoading` state to show loading spinners while checking sessions [40].
- `src/pages/login/index.tsx`: Implements the login/sign-up wizard. It toggles between `showSignup` and `showLogin` and manages the `currentStep` state for the multi-step form.
- `src/pages/role-based-dashboard-hub/student/StudentDashboard.tsx`: Renders the student dashboard and now includes the embedded marketplace widget. It fetches stats and subscription info.
- `src/pages/role-based-dashboard-hub/teacher/TeacherDashboard.tsx`: Renders the teacher dashboard (currently static and unresponsive; functions to be implemented).
- `src/pages/role-based-dashboard-hub/admin/`: Contains admin dashboards (User Management, Content Management, etc.) but many routes are missing or incomplete.
- `src/pages/role-based-dashboard-hub/ceo/`: Contains the CEO dashboard.
- `src/services`: Contains `supabaseClient.ts` (initialises Supabase with `VITE_SUPABASE_URL` and `VITE_SUPABASE_ANON_KEY`), `apiClient.ts` (axios wrapper for calling edge functions or external APIs) and `stripeService.ts` (contains `createStripeCustomer` and `handleSubscription`). After removing Stripe, this file can be deprecated.
- `src/components`: Houses reused components like forms, modals, card components, icons, etc. `MarketplaceIcon.tsx` is here, implementing the tag icons.
- `supabase/migrations`: Contains SQL migration files. Notable migrations include `20260106050000_fix_invalid_product_categories.sql` and `20260106040400_fix_remaining_invalid_categories.sql` to convert outdated category enums into tags and update constraints. Another migration adds `tags` column to `marketplace_items` and populates it.

---

8. Further Research and User Perspective

The user's perspective emphasised a messy codebase with features spread across multiple files. They expressed frustration with inconsistent behaviour (e.g., features working in preview but not production) and emphasised the importance of reading all code and instructions. They specifically requested a detailed explanation of functions and rationale behind the code. Key takeaways from the user's POV include:

1. **Project Complexity**: Many features exist but are not connected. Different roles require separate dashboards with unique functionality. The marketplace and dashboards are interwoven, creating dependencies that are hard to track.
2. **Data Misalignment**: The front end sometimes uses mock arrays or static numbers that don't match database values (e.g., marketplace category counts). This causes confusion and misrepresents available items.

3. **Unimplemented Buttons**: Across roles, many UI buttons are inert because endpoints or handlers are missing (Manage Class, Add Student, Create Room). It is crucial to implement these features or hide them until ready.
4. **Real-Time Issues**: The site experiences WebSocket connection failures. Clear documentation on environment variables, CSP and RLS is necessary to avoid confusion.
5. **Stripe Removal**: The user insisted on removing Stripe integration to simplify sign-up and avoid token exhaustion during testing. They provided instructions to comment out the import, remove the Stripe call and instead create free trials.
6. **Overlapping Code**: There are many redundant files and unused components (e.g., `SubscriptionStep.tsx`, `PaymentStep.tsx`). The user wants to streamline the code by removing them and consolidating logic.
7. **Comprehensive Report**: They requested a 20-page summary explaining every function in the codebase from their perspective, ignoring images and focusing on text and code instructions. This report aims to satisfy that requirement.

Conclusion

LiqLearns is a sophisticated learning platform with role-based dashboards, collaborative study rooms, a marketplace, event scheduling, community interaction and administrative oversight. However, the project suffers from incomplete implementations, inconsistent data and configuration issues. The Student Dashboard uses blank modals and hard-coded values; study rooms fail due to RLS and Realtime misconfiguration; marketplace counts don't reflect real items; teacher/admin dashboards are mostly static; and Stripe integration complicates sign-up.

This report synthesizes information from the project files, Supabase database, and numerous debugging instructions to provide a comprehensive overview. Key actions needed include:

1. **Complete unimplemented features**: Implement API endpoints and UI handlers for joining rooms, managing classes, uploading content, creating events and tickets.
2. **Standardize data access**: Use Supabase exclusively for data retrieval; avoid mock data. Ensure RLS policies permit necessary operations.
3. **Simplify sign-up**: Remove Stripe calls, create user and student profiles with free trial periods and default values.
4. **Fix Realtime & CSP**: Add proper headers, set environment variables correctly and provide a fallback polling mechanism.
5. **Clean up codebase**: Remove unused components and consolidate the sign-up wizard. Document functions and roles clearly.

By addressing these issues, the LiqLearns platform can deliver a stable and engaging experience across all roles, fulfilling its vision of gamified, collaborative learning.

---

1 2 3 4 5 9 12 13 34 35 Build Full Apps from Plain Text- No Coding Required. Rocket.new
https://www.rocket.new/69178932769eea00141a686d%23preview

6 7 8 11 14 15 16 17 18 19 20 21 22 23 24 25 26 27 29 LiqLearns
https://liqlearns.com/role-based-dashboard-hub

10 39 Authentication | Supabase
https://supabase.com/dashboard/project/qetfonluwxtosvhptlff/auth/policies

28 LiqLearns
https://liqlearns.com/content-management-hub

30 33 40 Build Full Apps from Plain Text- No Coding Required. Rocket.new
https://www.rocket.new/69178932769eea00141a686d%23code

31 liqlearns_admin/src/pages/login/components/ApprovalConfirmationStep.tsx at main · tensae-code/liqlearns_admin
https://github.com/tensae-code/liqlearns_admin/blob/main/src/pages/login/components/ApprovalConfirmationStep.tsx

32 liqlearns_admin/src/contexts/AuthContext.tsx at main · tensae-code/liqlearns_admin
https://github.com/tensae-code/liqlearns_admin/blob/main/src/contexts/AuthContext.tsx

36 liqlearns@outlook.com | liqlearns@outlook.com's Org | Supabase
https://supabase.com/dashboard/project/qetfonluwxtosvhptlff/editor/17547

37 liqlearns@outlook.com | liqlearns@outlook.com's Org | Supabase
https://supabase.com/dashboard/project/qetfonluwxtosvhptlff/editor/17749

38 liqlearns@outlook.com | liqlearns@outlook.com's Org | Supabase
https://supabase.com/dashboard/project/qetfonluwxtosvhptlff/editor/17615

ChatGPT

LiqLearns Project Overview

Introduction

LiqLearns is a comprehensive **learning platform** built with React, Supabase and Stripe (soon to be removed). The platform provides role-based experiences for students, teachers, administrators and executives (CEO). Its main features include personalized dashboards, study rooms, a marketplace for learning materials, event management, community interaction, help and support, and administrative tools. Supabase backs the database, authentication and real-time infrastructure, while the UI is built with React and TailwindCSS.

The project's codebase is organized into multiple directories: `src/components` contains reusable UI elements; `src/pages` holds each page of the application (e.g., dashboard, marketplace, login); `src/contexts` provides React contexts for state management (such as `AuthContext`); `services` contains service clients for Supabase, API calls and Stripe; `supabase` holds database schema, edge functions and policies. The `public` folder holds static assets, and `.env` houses environment variables.

During testing, multiple issues were discovered: blank modals on the student dashboard, failing study room joins, missing course details in the Quest page, mismatched category counts in the marketplace, unresponsive teacher dashboard actions and 404 pages in admin areas. Additional problems included unstable real-time connections due to Content-Security-Policy (CSP) restrictions and the need to remove Stripe from the sign-up flow. This report synthesizes the instructions, code and fixes discussed with the user to offer a detailed understanding of each feature, the underlying code and how to address these issues.

---

1. Student Role: Dashboard and Learning Flow

1.1 Dashboard Structure

The **Student Dashboard** is the landing page after logging in or signing up. It is implemented in `src/pages/role-based-dashboard-hub/student/StudentDashboard.tsx`. The dashboard displays:

- **Total Lessons**, **Total Badges**, **Total XP**, **Total Certificates** and **Current Streak** metrics in cards at the top of the page. Each card is expected to open a modal showing further details when clicked. However, during testing these modals appeared blank because the data fetch failed and no fallback was implemented [1].
- **Today's Quest** section, a checklist of tasks for the day (e.g., "Complete daily quiz", "Watch a video lesson"). Each item can be marked as completed, awarding points or streaks.
- A **Progress bar** summarizing course progress. It uses React `useEffect` hooks to fetch progress data from the `student_stats` table in Supabase and update automatically via real-time subscription. A key bug was found: the original code placed a `useState` call inside a helper

function `renderProgressSection()`, violating React rules and causing unresponsive updates [2]. The fix is to move the state hook to the component root.
- **Embedded Marketplace**. During project evolution, the marketplace was integrated directly into the Student Dashboard rather than a separate page. A `MarketplaceIcon` component with 13 custom icons (interactive, template, tutorial, lecture, reference, spaced-repetition, gamified, memory, audio, video, books, flashcards, games) is used to visually tag marketplace items [3].
- **Student Stats**. A new `student_stats` table is proposed (and partially implemented) to store real-time gamification metrics: XP, gold, streaks, levels, aura points, total courses, completed courses and last active date. A Supabase **edge function** called `student-stats` can be used to fetch or create these stats and return them to the front end [4]. The front end should poll this function or use Realtime subscription to avoid blank values. Mock data must be removed [5].

1.2 Quest and Course Details

The **Quest** page lists courses grouped by learning paths (e.g., Amharic Language, Ethiopian Culture, Mathematics). Each course card should open a side panel containing:

- Course description.
- Upcoming **Live Classes** with `Set Reminder` buttons.
- **Course Materials** (PDF, audio, video, notes, assignments, homework, movies).
- **Learning Tools** categories (Audio, Video, Notes, Assignments, Homework, Movies).
- **AI Insights** with a `View Detailed Analysis` link.

During testing, selecting a course produced a “Course not found in database” error and no materials displayed [6]. This indicates that either the course slug is incorrect or the database lacks the course record. To fix it:

1. Verify the **Courses** table in Supabase has entries for each course slug used in the front end.
2. Ensure the front end fetches courses by their `id` or `slug` rather than relying on static arrays.
3. Remove any hard-coded lists and load data directly from Supabase.
4. Provide error messages when a course is missing instead of leaving the modal blank.

1.3 Study Rooms

Study rooms allow students to collaborate via video chat or chat rooms. They are listed on the **Study Rooms** page. Each room card includes a **Join Room** button. Issues discovered:

- **Failed to join room** error appears for all rooms [7]. A “Retry” button simply refreshes the list.
- **Create Room** button does nothing when clicked [8].

Possible reasons and fixes:

1. **Real-time WebSocket**: The front end uses `supabase.realtime` to join channels. If CSP restricts WebSocket or the environment variables for Realtime aren’t set correctly, connections fail. Update `vercel.json` or server headers to allow `wss://*.supabase.co` [9].
2. **Row Level Security (RLS)**: The `study_rooms` table has policies like `students_view_age_appropriate_rooms`, `students_create_study_rooms` and

`students_update_own_rooms` [10] . If the age categories or user ages aren't set, the SELECT policy denies access. Add a default age group or adjust the policy.

3. **Missing membership table**: Joining may insert into a `study_room_members` or similar table. Without proper RLS policies or triggers, insertion fails. Check Supabase for membership table and add an `INSERT` policy such as `students_join_room` with `user_id = auth.uid()`.
4. **Create Room functionality**: The UI likely calls an edge function or API to create a room. If this function isn't implemented or RLS denies insertion, the button is inert. Implement the API and add RLS.

1.4 Event Calendar

The **Events** section provides a calendar and a **Create Event** modal. The modal includes fields for title, description, start time, end time, event type and a public/private toggle. The bug observed: date/time pickers didn't open when clicking the date fields [11] . This is often due to missing date picker library CSS or script or z-index issues. To fix:

- Ensure the date picker component is imported from its library (e.g., `react-datepicker` or `@headlessui/react` date picker) and its CSS is loaded in `index.css`.
- Remove any `overflow: hidden` on modal container that might hide the picker.
- Provide proper `z-index` so the picker appears above the modal.

1.5 Marketplace Embedded in Dashboard

A major evolution was integrating the **Marketplace** into the Student Dashboard. Initially, the marketplace had its own page (`marketplace-hub`). After user feedback, the team removed the separate route and embedded a marketplace widget in the dashboard. Steps involved:

1. **Create** `EmbeddedMarketplace` **component**: The new component fetches items from the `marketplace_items` table and displays them with tag icons. It can filter by tags and category [12] .
2. **Tagging system**: The `marketplace_items` table gained a `tags` array column and triggers to populate tags based on product names and descriptions. SQL migrations were written to update existing items (e.g., assign 'interactive' tag to interactive courses, 'template' to spreadsheets and forms, 'audio' to podcasts) [13] .
3. **Marketplace Icons**: The `MarketplaceIcon` component uses a `switch` statement to return appropriate SVG icons for each tag [3] .
4. **UI integration**: The Student Dashboard now renders the marketplace grid at the bottom. Each item displays up to two tags with icons. The route `marketplace-hub` was removed from `src/Routes.tsx` [12] .
5. **Cache invalidation**: Vercel caching caused outdated versions to persist. An `APP_VERSION` constant is added to the code so that each deploy changes the version, forcing the browser to fetch fresh assets [3] . Hard reload (`Ctrl+Shift+R`) and incognito mode may be required to see updates.

1.6 Community and Help

The **Community** page offers group chats and a community wall. Creating a group doesn't open a modal, and clicking existing groups yields nothing [14]. This likely means the API to manage groups isn't implemented. To fix:

- Add a `chat_groups` table and corresponding API for creating and joining groups.
- Populate the Community page with data from Supabase and enable posting messages using a `community_posts` table.

The **Help** center includes FAQs and a ticket system. The **Submit a Ticket** modal collects title, category, priority and description. After filling in, the `Create Ticket` button becomes active [15]. Tickets appear in `My Tickets` with conversation threads [16].

1.7 Settings

The **Settings** page allows editing profile information, security, notifications, language & region, privacy and learning goals. Key features include:

- Uploading a profile picture.
- Changing password.
- Toggling notification preferences.
- Selecting primary language and timezone.
- Setting privacy levels (public, private, community only).
- Viewing GDPR/Privacy options with data export and account deletion [17].
- Setting learning goals (daily time, topics) [18].

No major issues were found here, but saving changes must trigger updates to `user_profiles` and `student_profiles` tables via Supabase.

---

2. Teacher Role: Managing Classes and Content

2.1 Teacher Dashboard

Teachers have a dedicated dashboard (e.g., `src/pages/role-based-dashboard-hub/teacher/TeacherDashboard.tsx`) that displays:

- **Total Students**, **Active Classes**, **Assignments Graded**, **Upcoming Sessions**. These metrics should be loaded from the `classes`, `student_profiles`, and `assignments` tables. However, clicking **Manage Class** or **Create Class** does nothing [19], and editing students fails [20]. This indicates missing or unimplemented handlers.

- **My Classes** section lists courses taught by the teacher, including schedules. None of the actions (Manage, Edit, Add Student) work because there are no API endpoints to support them [21].

2.2 Students & Reports

- **Student Management** lists students with progress bars and statuses. View Profile and Add Student do nothing. To fix, implement pages/modals for viewing student profiles and adding them to classes. Use Supabase RLS to ensure teachers can only see students in their courses.

- **Reports** should show performance reports (average class performance, attendance, assignments completed). An `Export Report` button is present but nonfunctional [22]. This requires generating CSV/Excel files from Supabase data and triggering file downloads.

2.3 Schedule & Store

- **Schedule** page displays a calendar of upcoming sessions. The `Add Session` button does nothing [23]. A modal should appear to set session details and store them in a `sessions` table.

- **Store** allows teachers to upload products (courses, worksheets, videos) and manage inventory. Buttons like `Add Product`, `Edit` and `Delete` are inert [24]. Implementation should include a form for product details, file uploads via Supabase Storage, and linking items to teacher IDs.

2.4 Help & Settings

The **Help** page shows FAQs but clicking questions does not expand answers [25]; fix by toggling collapsed states and adding event handlers. The **Settings** page replicates the student settings with additional fields like payment information or subscription rates [26].

---

3. Admin and CEO Roles

3.1 Admin Dashboard

Admins view overall platform metrics: total users, total teachers, total students, active users today, active courses and system alerts. They can manage users, content, finances, analytics and store. Issues found:

- **User Management** page lists users but provides no actions (edit, delete, assign roles) [27]. To fix: add actions like edit profile, reset password and deactivate account.

- **Content Management** link labeled “Open Content Management Hub” directs to a 404 page [28]. The route needs to be added to `src/Routes.tsx` and the content management page created.

- Many sections (Financial, Analytics, Store, Events, Approvals, Support) have side navigation items but no implementation. Each needs to be built and wired to the backend. For example, **Financial** can show revenue, payout schedules; **Analytics** can show user engagement; **Store** can list all marketplace items; **Approvals** can manage content submitted by teachers.

3.2 CEO Role

The CEO dashboard displays high-level metrics, including revenue, total users, students, tutors and an embed of the landing page demo video [29]. Additional sections may include:

- **Course Management** to oversee all courses, approve or reject new courses.
- **Business Analytics** to track revenue growth, user acquisition and churn.
- **Financial Overview** to see gross revenue, operating costs, profit margins.
- **Organization & Growth Metrics** for team structure and expansion.

These sections must be implemented similarly to the admin sections, with appropriate data queries and RLS policies.

---

4. Sign-Up and Authentication Flow

4.1 Existing Flow

The sign-up process is a multi-step wizard located in `src/pages/login/index.tsx`. Steps include:

1. **Role Selection**: Student, Teacher, Support, Admin. Role choice determines subsequent questions.
2. **General Questions**: Collects full name, username, email, phone and sponsor username (for referral system). It uses asynchronous functions `checkUsernameAvailability` and `checkSponsorUsername` to validate entries by calling an edge function `/functions/v1/check-username` [30].
3. **Two-Factor Verification**: Sends a code via phone or email. A verification input appears after sending the code.
4. **Policy Agreement**: Accept terms, conditions and privacy policies (e.g., COPPA for minors). The user must confirm they are above 13 years old and the content is age appropriate.
5. **Approval Confirmation**: Notifies the user that their application is submitted. For certain roles (e.g., teacher or admin), manual approval may be required [31].

Originally, there was a **Subscription/Payment step** that integrated Stripe to collect payment information. The `AuthContext.tsx` file imported `createStripeCustomer` from `../services/stripeService` and, after a successful `supabase.auth.signUp`, called `createStripeCustomer(userId, email, fullName)` [32]. The wizard also contained `PaymentStep.tsx` and `SubscriptionStep.tsx` components. These steps complicated sign-up and were removed following user feedback and to save tokens. The Stripe integration code must be fully removed.

4.2 Removing Stripe Integration

To remove Stripe from sign-up:

1. **Comment out or delete** `import { createStripeCustomer }` **in** `AuthContext.tsx`.
2. **Remove the call to** `createStripeCustomer` within the `signUp` function. This prevents Stripe customer creation after sign-up.

3. **Insert profile creation logic**: After a successful `supabase.auth.signUp`, the code should insert a new row into `user_profiles` and `student_profiles` (when role is student). For students, default values such as `subscription_plan: 'free_trial'` and `trial_end_date: NOW() + 14 days` are inserted. Use Supabase service role (via an edge function or direct insert) to bypass RLS restrictions.
4. **Remove unused components**: Delete or ignore `SubscriptionStep.tsx` and `PaymentStep.tsx` to keep the codebase clean.

Due to editing instability in Rocket, the actual modifications were challenging. The intention is clearly documented: new users should be created in the database with a free trial and no Stripe integration, and sign-up should be simplified accordingly [33].

4.3 Authentication Context

The `AuthContext` manages user sessions and exposes `signIn`, `signUp`, `signOut`, `sendTwoFactorCode`, `verifyTwoFactorCode` and `sendEmailOTP`. It uses `supabase.auth.getSession()` to restore sessions and listens for `onAuthStateChange` events. After removing Stripe, `signUp` should look like this (pseudo-code):

```
async function signUp(email: string, password: string, options?: { data: any })
{
  const { data, error } = await supabase.auth.signUp({ email, password,
options });
  if (error) return { error };
  // Create user_profile
  const userId = data.user.id;
  await supabase.from('user_profiles').insert({
    id: userId,
    email,
    full_name: options?.data?.full_name,
    username: options?.data?.username,
    role: options?.data?.role,
    phone: options?.data?.phone,
    sponsor_username: options?.data?.sponsor_username
  });
  if (options?.data?.role === 'student') {
    const trialEnd = new Date();
    trialEnd.setDate(trialEnd.getDate() + 14);
    await supabase.from('student_profiles').insert({
      id: userId,
      subscription_plan: 'free_trial',
      trial_end_date: trialEnd,
      has_active_subscription: false
    });
  }
```

```
  return { data };
}
```

This code must run in a secure context (server side or via edge function) because RLS may restrict client-side inserts. You can create a Supabase Edge Function `create-user-profile` that performs these inserts using `supabaseServiceRoleKey` and call it from the front end.

4.4 Login Flow

Login is handled by `LoginForm.tsx`, which uses `supabase.auth.signInWithPassword` and `useNavigate` to redirect users to the role-based dashboard after a successful sign-in [30]. If the credentials are invalid, an error message appears. The login form includes a **Remember Me** checkbox, a **Forgot Password?** link and a **Create Account** link to switch to the sign-up wizard.

---

5. Real-Time Connections and Environment Variables

5.1 Content Security Policy (CSP)

The site uses Vercel for deployment. To enable Supabase Real-time, the CSP must allow WebSocket connections to `wss://*.supabase.co` and API requests to `https://*.supabase.co`. Early messages indicate connection failures due to missing CSP headers. The fix is to add a `vercel.json` or `_headers` file in `public/` that includes:

```
{
  "headers": [
    {
      "source": "/(.*)",
      "headers": [
        {
          "key": "Content-Security-Policy",
          "value": "default-src 'self' https://*.supabase.co https://
static.rocket.new; connect-src 'self' https://*.supabase.co wss://*.supabase.co;
script-src 'self' 'unsafe-inline' https://*.supabase.co;"
        }
      ]
    }
  ]
}
```

This allows the front end to subscribe to real-time changes. Without it, Supabase will log errors about blocked connections [9].

5.2 Environment Variables

Different environments (preview vs. production) have separate Supabase URLs and keys. Many issues, such as missing data or failing logins, arise from the front end referencing the wrong environment variables. To fix:

- Ensure `.env` in the project and environment variables in Vercel are set correctly for `VITE_SUPABASE_URL` and `VITE_SUPABASE_ANON_KEY` [34].
- Remove fallback logic (`mockData`, `mockStats`) from code; rely solely on real data. Use environment variables to differentiate local vs. production.

5.3 Real-Time Data Handling

The Student Dashboard uses the `supabase.realtime` client to subscribe to tables like `student_stats`. When the connection is offline, the UI should show an offline indicator and fall back to HTTP polling. A sample implementation:

```
const [stats, setStats] = useState<Stats | null>(null);
const [connection, setConnection] = useState<'offline' | 'connecting' |
'online'>('connecting');
useEffect(() => {
  const bootstrap = async () => {
    const { data } = await supabase.from('student_stats').select('*').eq('id',
userId).single();
    setStats(data);
  };
  bootstrap().then(() => setConnection('online')).catch(() =>
setConnection('offline'));
  const subscription = supabase.channel('public:student_stats')
    .on('postgres_changes', { event: '*', schema: 'public', table:
'student_stats', filter: `id=eq.${userId}` }, (payload) => {
      setStats(payload.new);
    })
    .subscribe();
  return () => {
    subscription.unsubscribe();
  };
}, [userId]);
```

If `connection` is offline, the component can display a yellow WiFi icon and continue polling every 5 seconds using `setInterval` [35].

---

6. Database Structure and RLS Policies

6.1 Tables

The Supabase project for LiqLearns contains numerous tables. Important ones include:

- **user_profiles**: Stores user info (id, email, username, full_name, phone, role, sponsor_username, date_of_birth, profile_picture_url, timezone, country, etc.). Inserted when a user signs up.
- **student_profiles**: Extends `user_profiles` for students, including `subscription_plan`, `trial_end_date`, `subscription_start_date`, `subscription_end_date`, `has_active_subscription`, `parental_consent`, etc. [36].
- **teacher_profiles**: Contains teacher-specific details such as biography, skills and rating.
- **admin_profiles** and **ceo_profiles**: For admin and CEO roles, storing additional data.
- **courses**: Each course has `id`, `title`, `description`, `lesson_type` and other metadata [37].
- **lessons**: A lesson belongs to a course and contains content (video, audio, PDF) and type.
- **marketplace_items**: Each item has `id`, `title`, `description`, `price`, `point_cost`, `creator_id`, `tags` (array) and `category` (enum). The `tags` column was added to allow flexible filtering.
- **subscription_plans**: Holds subscription tiers (Basic, Standard, Pro, Elite, Premium) with monthly/yearly prices [38].
- **student_stats**: Proposed to store gamification stats such as XP, gold, streak and level 【599674900694139†screenshot】.
- **study_rooms**: Contains room `id`, `name`, `description`, `age_group`, `creator_id`. Additional tables like `study_room_members` might exist to track participants.
- **events**: Each event has `title`, `description`, `start_time`, `end_time`, `public`, `creator_id`, `type` (class, group study, etc.).
- **support_tickets**: Keeps track of help tickets with `title`, `category`, `priority`, `status`, etc.
- **notifications**: Stores notifications for real-time updates.

6.2 Policies

Supabase uses **Row Level Security (RLS)** to control access to tables. Example policies:

- `admins_view_all_student_profiles`: Allows admins to `SELECT` from `student_profiles` regardless of row content [39].
- `students_manage_own_student_profiles`: Allows students to `SELECT` or `UPDATE` only their profile where `id = auth.uid()` [39].
- `students_view_age_appropriate_rooms`: For `study_rooms`, allows students to view rooms within their age group category [10].

When features fail, it is often because RLS denies access. Always check policies to ensure the correct role/row filter is applied.

---

7. Code Structure and Key Components

The project is modular. Understanding each file helps maintain the code. Some important files:

- `src/App.tsx`: Configures React Router and imports `AuthProvider`. It defines routes for each page (e.g., `/role-based-dashboard-hub/:role`, `/login`, `/signup`, etc.). After the marketplace integration, the `marketplace-hub` route is removed.
- `src/contexts/AuthContext.tsx`: Provides authentication context. It wraps the app and exposes `signIn`, `signUp`, `signOut`, etc. It also maintains `authLoading` state to show loading spinners while checking sessions [40].
- `src/pages/login/index.tsx`: Implements the login/sign-up wizard. It toggles between `showSignup` and `showLogin` and manages the `currentStep` state for the multi-step form.
- `src/pages/role-based-dashboard-hub/student/StudentDashboard.tsx`: Renders the student dashboard and now includes the embedded marketplace widget. It fetches stats and subscription info.
- `src/pages/role-based-dashboard-hub/teacher/TeacherDashboard.tsx`: Renders the teacher dashboard (currently static and unresponsive; functions to be implemented).
- `src/pages/role-based-dashboard-hub/admin/`: Contains admin dashboards (User Management, Content Management, etc.) but many routes are missing or incomplete.
- `src/pages/role-based-dashboard-hub/ceo/`: Contains the CEO dashboard.
- `src/services`: Contains `supabaseClient.ts` (initialises Supabase with `VITE_SUPABASE_URL` and `VITE_SUPABASE_ANON_KEY`), `apiClient.ts` (axios wrapper for calling edge functions or external APIs) and `stripeService.ts` (contains `createStripeCustomer` and `handleSubscription`). After removing Stripe, this file can be deprecated.
- `src/components`: Houses reused components like forms, modals, card components, icons, etc. `MarketplaceIcon.tsx` is here, implementing the tag icons.
- `supabase/migrations`: Contains SQL migration files. Notable migrations include `20260106050000_fix_invalid_product_categories.sql` and `20260106040400_fix_remaining_invalid_categories.sql` to convert outdated category enums into tags and update constraints. Another migration adds `tags` column to `marketplace_items` and populates it.

---

8. Further Research and User Perspective

The user's perspective emphasised a messy codebase with features spread across multiple files. They expressed frustration with inconsistent behaviour (e.g., features working in preview but not production) and emphasised the importance of reading all code and instructions. They specifically requested a detailed explanation of functions and rationale behind the code. Key takeaways from the user's POV include:

1. **Project Complexity**: Many features exist but are not connected. Different roles require separate dashboards with unique functionality. The marketplace and dashboards are interwoven, creating dependencies that are hard to track.
2. **Data Misalignment**: The front end sometimes uses mock arrays or static numbers that don't match database values (e.g., marketplace category counts). This causes confusion and misrepresents available items.

3. **Unimplemented Buttons**: Across roles, many UI buttons are inert because endpoints or handlers are missing (Manage Class, Add Student, Create Room). It is crucial to implement these features or hide them until ready.
4. **Real-Time Issues**: The site experiences WebSocket connection failures. Clear documentation on environment variables, CSP and RLS is necessary to avoid confusion.
5. **Stripe Removal**: The user insisted on removing Stripe integration to simplify sign-up and avoid token exhaustion during testing. They provided instructions to comment out the import, remove the Stripe call and instead create free trials.
6. **Overlapping Code**: There are many redundant files and unused components (e.g., `SubscriptionStep.tsx`, `PaymentStep.tsx`). The user wants to streamline the code by removing them and consolidating logic.
7. **Comprehensive Report**: They requested a 20-page summary explaining every function in the codebase from their perspective, ignoring images and focusing on text and code instructions. This report aims to satisfy that requirement.

Conclusion

LiqLearns is a sophisticated learning platform with role-based dashboards, collaborative study rooms, a marketplace, event scheduling, community interaction and administrative oversight. However, the project suffers from incomplete implementations, inconsistent data and configuration issues. The Student Dashboard uses blank modals and hard-coded values; study rooms fail due to RLS and Realtime misconfiguration; marketplace counts don't reflect real items; teacher/admin dashboards are mostly static; and Stripe integration complicates sign-up.

This report synthesizes information from the project files, Supabase database, and numerous debugging instructions to provide a comprehensive overview. Key actions needed include:

1. **Complete unimplemented features**: Implement API endpoints and UI handlers for joining rooms, managing classes, uploading content, creating events and tickets.
2. **Standardize data access**: Use Supabase exclusively for data retrieval; avoid mock data. Ensure RLS policies permit necessary operations.
3. **Simplify sign-up**: Remove Stripe calls, create user and student profiles with free trial periods and default values.
4. **Fix Realtime & CSP**: Add proper headers, set environment variables correctly and provide a fallback polling mechanism.
5. **Clean up codebase**: Remove unused components and consolidate the sign-up wizard. Document functions and roles clearly.

By addressing these issues, the LiqLearns platform can deliver a stable and engaging experience across all roles, fulfilling its vision of gamified, collaborative learning.

---

1 2 3 4 5 9 12 13 34 35 Build Full Apps from Plain Text- No Coding Required. Rocket.new
https://www.rocket.new/69178932769eea00141a686d%23preview

6 7 8 11 14 15 16 17 18 19 20 21 22 23 24 25 26 27 29 LiqLearns
https://liqlearns.com/role-based-dashboard-hub

10 39 Authentication | Supabase
https://supabase.com/dashboard/project/qetfonluwxtosvhptlff/auth/policies

28 LiqLearns
https://liqlearns.com/content-management-hub

30 33 40 Build Full Apps from Plain Text- No Coding Required. Rocket.new
https://www.rocket.new/69178932769eea00141a686d%23code

31 liqlearns_admin/src/pages/login/components/ApprovalConfirmationStep.tsx at main · tensae-code/liqlearns_admin
https://github.com/tensae-code/liqlearns_admin/blob/main/src/pages/login/components/ApprovalConfirmationStep.tsx

32 liqlearns_admin/src/contexts/AuthContext.tsx at main · tensae-code/liqlearns_admin
https://github.com/tensae-code/liqlearns_admin/blob/main/src/contexts/AuthContext.tsx

36 liqlearns@outlook.com | liqlearns@outlook.com's Org | Supabase
https://supabase.com/dashboard/project/qetfonluwxtosvhptlff/editor/17547

37 liqlearns@outlook.com | liqlearns@outlook.com's Org | Supabase
https://supabase.com/dashboard/project/qetfonluwxtosvhptlff/editor/17749

38 liqlearns@outlook.com | liqlearns@outlook.com's Org | Supabase
https://supabase.com/dashboard/project/qetfonluwxtosvhptlff/editor/17615

ChatGPT

Comprehensive LiqLearns Platform Report

Introduction and Context

LiqLearns is an ambitious educational platform built with a modern web stack. The front-end uses **React 18** with **TypeScript**, **Redux Toolkit** for state management, **React Router v6** for routing, **TailwindCSS** for styling, and assorted UI libraries like **Lucide** icons. The back-end leverages **Supabase** for its PostgreSQL database, authentication, row-level security (RLS) and real-time services. Originally the project included **Stripe** for paid subscriptions, but this integration is currently being removed. Throughout development the team also utilised **Supabase Edge Functions** for serverless custom logic, and the site is deployed on **Vercel**.

The project repository is organised as follows:

Folder/File	Description
`src/components/`	Reusable components like buttons, cards, modals, and icons.
`src/pages/`	Page components grouped by feature (dashboards, marketplace, events, etc.).
`src/contexts/`	React contexts (e.g. `AuthContext.tsx`) for authentication and global state.
`src/services/`	Service clients such as `supabaseClient.ts` and API helpers.
`supabase/`	SQL migrations, triggers, policies and edge functions.
`.env`	Environment variables (not committed).
`public/`	Static assets and `index.html`.

During a series of interactive debugging sessions, multiple roles were tested: **Student**, **Teacher**, **Admin**, and **CEO**. These sessions revealed functional gaps, UI inconsistencies, and architectural shortcomings. The following pages synthesise those findings and provide a holistic examination of each feature and the underlying code. To make this report self-contained, the analysis is grouped by major feature area, with code insights and suggested fixes.

---

1. Student Experience

1.1 Dashboard Overview

Upon logging in as a student, users land on the **Student Dashboard** (`src/pages/role-based-dashboard-hub/student/StudentDashboard.tsx`). The dashboard is intended to be the learner's command centre. It displays:

- **Total Lessons**, **Total Badges**, **Total XP**, **Total Certificates** and **Current Streak** in summary cards. Each card is meant to open a detailed modal when clicked. During testing these modals were blank because the data fetch returned null and there was no fallback. This indicates either a missing API endpoint or a failure to pass the current user ID to the query. A proper loading state and error message should be added instead of rendering an empty modal.

- **Today's Quest**, a checklist of daily activities (e.g., "Complete daily quiz", "Watch a video lesson"). Completing tasks should reward XP or streak points. In the current implementation the tasks list appears but there is no persistent state to track completion; marking an item as done does not update the backend. A `student_quests` table and corresponding API should record completion.

- **Study Rooms** quick entry – a button in the sidebar leads to the study rooms area (see Section 2 below).

1.2 Learning Paths and Quest Page

The **Quest** page under the student role allows learners to select a learning path (Amharic Language, Ethiopian Culture, Mathematics, Science, English, Technology & Coding). Selecting a path opens a side panel listing live classes and course materials. The panel includes:

1. **Live Classes**: Scheduled sessions with start dates and times. Each entry offers a "Set Reminder" button. Currently this button does not interact with any calendar API; it simply logs to console. To make this useful, integrate with a calendar service or send a notification via Supabase functions.

2. **Course Materials**: This section should display PDFs, videos, audio files, and quizzes. However, it currently shows an error: "Course not found in database". The root cause appears to be that the `courses` table is empty or the front-end is passing an incorrect slug. Seeding the database with real courses and adjusting the query to use dynamic IDs will resolve this error.

3. **Learning Tools**: Tabs for audio, video, notes, assignments and movies. None of these tabs load content. The UI highlights the selected tab, but there is no data fetch. This indicates the API calls (likely through `supabase.from('materials')`) were never implemented.

1.3 Marketplace Integration

Initially the marketplace existed as a separate page. Through user feedback it was decided to embed the **Marketplace** into the student dashboard to encourage discovery. The marketplace allows students to

purchase or download items such as books, videos, audio guides, games, flashcards and worksheets. In testing, several issues were noted:

- The category counts (e.g. “Books 45”) do not match the actual number of items shown (only 4 items appear). This is because the counts are hard-coded in the component rather than being computed from the database. To fix this, query `marketplace_items` with `supabase.from('marketplace_items').select('*', { count: 'exact' })` grouped by category and display the returned counts.

- Some categories like “Games” or “Flashcards” display “No items available” despite the counts suggesting otherwise. This is again due to mismatched seeds or query filters.

- A new **Tagging System** was proposed: instead of a single `category` field, each item has a `tags` array. Tags include `interactive`, `template`, `tutorial`, `lecture`, `reference`, `spaced-repetition`, `gamified`, `memory`, `audio`, `video`, `books`, `flashcards`, `games`. SQL migrations were prepared to add a `tags` column and update existing rows. A custom `MarketplaceIcon` component renders appropriate icons based on tags. This tag-driven design allows multi-dimensional filtering and should replace the rigid categories.

- Items can be added to a cart. When adding free items the cart shows “0 points” and a “Proceed to Checkout” button. The checkout integration with Stripe is being removed; until payment is re-implemented, clicking checkout should either show a modal explaining that payment is disabled or simply allow downloading free resources.

1.4 Study Rooms

Study rooms are community spaces where students can join real-time sessions. The **Study Rooms** page (`src/pages/study-rooms/index.tsx`) lists available public rooms (e.g. “Language Practice Room”). Issues observed include:

- Clicking **Join Room** triggers a dark overlay with the error “Failed to join room”. This is likely due to missing RLS policies on the `study_rooms` or `study_room_members` tables. To fix, ensure that RLS policies allow authenticated students to insert a membership record into the `room_members` table where `student_id = auth.uid()` and to select any room with `is_public = true`.

- **Create Room** button does nothing. It should open a modal or page with a form to enter room name, description and maximum capacity. After submission, the front-end should call `supabase.from('study_rooms').insert(...)`. A row-level policy should permit students to insert new rows and become the host.

1.5 Events

Students can navigate to an **Events** page where a calendar is displayed and a **Create Event** button opens a modal. The modal asks for title, description, start and end time, type and a “make event public” checkbox. The date/time pickers do not display; clicking the fields does nothing. This is due to missing imports for the date picker library (likely `@headlessui/react` or `react-date-picker`) and CSS for the picker.

Importing the component and adding the CSS will enable proper date selection. In addition, the event should be created via `supabase.from('events').insert(...)` and optionally broadcast via real-time for other attendees.

1.6 Community and Help

The **Community** section shows group chats and a wall for posts. Clicking on a group does nothing – chat rooms are not implemented. A `chat_rooms` and `messages` tables should be added, along with a WebSocket (via Supabase Realtime) to send messages. The “Create Group” button should open a modal to define group name and participants.

The **Help** page lists FAQs. A **Submit a Ticket** button opens a form requiring title, category, priority and description. The “Create Ticket” button stays disabled until all fields are filled. Submitted tickets should be inserted into a `support_tickets` table, and responses should appear in a conversation thread. A **My Tickets** tab lists previous tickets; clicking a ticket opens a conversation view. The send button should call an edge function that inserts the message into `ticket_messages` table.

1.7 Settings and Profile

Under Settings, students can modify personal information, password, language, privacy settings, GDPR preferences and learning goals. Each tab corresponds to a separate component. For example, the **Learning Goals** tab presents a checklist of skills (e.g. vocabulary, grammar, speaking). Updates should be written to `student_profiles` via Supabase. In testing, no changes were saved. Implement `onSubmit` handlers and call `supabase.from('student_profiles').update()` accordingly.

---

2. Teacher Experience

2.1 Teacher Dashboard and Classes

Logging in as a teacher takes you to **TeacherDashboard** (`src/pages/role-based-dashboard-hub/teacher/TeacherDashboard.tsx`). This page shows metrics for **Total Students**, **Active Classes**, **Assignments Graded**, and **Upcoming Sessions**. Buttons like **Manage Class** and **Create Class** appear. In practice, they do nothing because the event handlers are undefined. Fixes include:

1. **Manage Class**: On click, navigate to a dedicated class management page (`/teacher/classes/[classId]`) showing rosters, grades, and materials. The route file (`src/Routes.tsx`) must define this path.

2. **Create Class**: Show a modal with fields for name, description, schedule and capacity. Insert new row into `classes` table and assign teacher as instructor.

3. **Upcoming Sessions**: This section should fetch from an `sessions` table where `teacher_id = auth.uid()` and display date/time. Without this query, it remains static.

2.2 Student Management

The **Students** page lists all students in a teacher's classes with progress bars and statuses (active/inactive). Buttons like **View Profile** and **Add Student** do nothing. Implementation should:

- Connect the **View Profile** button to navigate to `/teacher/student/[studentId]` which displays the student's details and performance.
- Use an **Add Student** button to open a modal where the teacher can invite a student via email. The backend should insert a record in `enrollments` linking the student to the class.

2.3 Content Library and Store

Teachers can manage course content and sell supplementary materials. In the current build:

- **Content Library** displays uploaded documents but the **Edit**, **Share**, and **Upload Content** buttons are inert. Proper handlers must call Supabase Storage to upload files and update `teacher_content` table with metadata.

- **Store** allows teachers to upload products (notes, templates). The **Add Product** and **Edit/Delete** buttons have no effect. Implementation should connect these buttons to the `marketplace_items` table using Supabase calls. Teacher products should be flagged with `teacher_id` and visible only to students enrolled in their classes.

2.4 Reports, Schedule and Help

The **Reports** page summarises performance metrics (average class performance, attendance rate). Exporting to PDF or CSV should call a serverless function (e.g. using `pdf-lib`) but currently does nothing. Implement an `exportReports` handler that collects data and triggers a file download.

Teachers also have a **Schedule** calendar with the ability to add sessions. The Add Session button doesn't work because the date-picker and the `onAddSession` callback were never wired. As with events, import a date picker component and implement insertion into `sessions` table.

2.5 Teacher Settings and Help

Settings are similar to students but include additional toggles for notifications and privacy. The Help tab lists FAQs but the questions do not expand, likely due to missing `onClick` toggles or collapsed content CSS. This can be fixed by ensuring that each question component maintains its own expanded state.

---

3. Admin Experience

3.1 Overview and User Management

The **Admin** role (log in with `admin@liqlearns.com`) presents metrics for total users, students, teachers, etc. It is primarily designed to manage content and users:

- **Users** tab lists all user accounts. However, the rows are static; there is no ability to view or edit user details. To implement this, add actions to each row (e.g., "View", "Ban", "Reset Password"). Use Supabase queries to update `auth.users` and `user_profiles` accordingly.

- **Content Management** link is present in the top nav. Clicking it currently leads to a 404 page because the route is missing. A new page (`src/pages/content-management-hub/index.tsx`) should be created with tools for uploading, approving and categorising content across the platform. Then update `Routes.tsx` to map the path.

3.2 Financial and Analytics

Admin may also access financial data (subscriptions revenue, sales from marketplace) and analytics (user growth, engagement metrics). These pages either load static placeholders or are entirely missing. Implementation would require queries against `subscriptions`, `payments`, and user metrics tables and proper charts (e.g. via Recharts). A `FinancialDashboard.tsx` should display monthly revenue, churn rate, and top selling items.

3.3 Settings and Security

Admins should control platform settings: roles, permissions, content approvals, compliance. A **Security & Compliance Management** page was proposed. It should display open security tasks (like verifying Stripe webhook signatures, adding CSP headers), monitor real-time alerts, and provide toggles to enable/disable features like sign-up or event creation. This page can leverage `supabase` to fetch and update flags stored in an `admin_settings` table.

---

4. CEO Experience

The **CEO** role (login `ceo@liqlearns.com`) sees high-level metrics: total revenue, user growth, teacher count, and marketing statistics. There is a section for a **Landing Page Demo Video** with an "Edit URL" button, and **Landing Page Statistics** showing views, sign-ups, and conversion rates. Additional sections (Business Analytics, Financial Overview) are placeholders. To make this meaningful:

- Integrate charts that query aggregated data from Supabase (`payments` table for revenue, `users` table for growth). Use `supabase.rpc()` functions to precompute metrics for improved performance.

- Provide a content editor for the landing page (e.g. update hero text, CTA copy) and a video uploader. This can be built with a WYSIWYG editor and a `landing_pages` table to store content.

- Implement segmentation filters (e.g. by time period, by marketing channel). Use Supabase or external analytics tools to record UTM parameters and conversion events.

---

5. Support Experience

While not deeply tested, the **Support** role (login `support@liqlearns.com`) is meant to manage help tickets and answer user questions. The `Help` page under other roles interacts with this by creating tickets. In the admin dashboard, support agents should see a queue of open tickets, respond through a chat interface, and assign statuses (open, in progress, resolved). A `support_messages` table should record each communication. Additional features like macro responses, tagging, and analytics (average resolution time) would make the support workflow robust.

---

6. Authentication and Sign-Up Flow

6.1 Login Form

The login page lives in `src/pages/login/index.tsx`. It toggles between a **LoginForm** and a multi-step **Sign-Up Wizard**. The login form uses `supabase.auth.signInWithPassword` to authenticate. On success it stores the session in context and navigates to `/role-based-dashboard-hub`. If there's an error (invalid email/password), an error message is displayed. One improvement is to implement rate limiting on login attempts (e.g., lock out after 5 failed attempts) to prevent brute force attacks.

6.2 Sign-Up Wizard

The sign-up flow originally spanned nine steps: role selection, personal info, application form (for teachers), general questions, phone/email verification, subscription selection, policy agreements, review, and approval confirmation. Over time the payment step (via Stripe) and subscription selection were removed. The wizard still retains extraneous components such as `SubscriptionStep.tsx` and `PaymentStep.tsx`; these are no longer referenced but should be deleted to avoid confusion.

The recommended simplified flow is:

1. **Role Selection** – choose Student, Teacher, Support or Admin. The `role` value is stored in local state.
2. **Basic Info** – enter full name, username, email, password and sponsor username (optional). Use debounced validation to check if username is unique and if sponsor exists. Supabase edge functions `check-username` and `check-sponsor` were created for this purpose. They query `user_profiles` to ensure uniqueness and enforce rules (e.g., only students and teachers can sponsor; admin cannot be a sponsor). Debouncing prevents API thrashing as users type.
3. **Contact Verification** – send OTP to email or phone. Instead of blocking progression until verification, allow users to continue and verify later via a link in the welcome email.
4. **Policy Agreement** – present terms of service, privacy policy and honour code with checkboxes. Users must acknowledge to proceed.

5. **Completion** – call `supabase.auth.signUp` with the email and password. After sign up, insert a row into `user_profiles` with fields `id` (matching `auth.users.id`), `username`, `full_name`, `sponsor_username`, `role` and metadata. Next, insert a row into `student_profiles` (if role is student) with `subscription_plan = 'free_trial'` and `trial_end_date` set to 14 days from now. Use `supabase.from('student_profiles').insert(...)`. For teachers, insert into `teacher_profiles`. Finally, redirect to the relevant dashboard.

6.3 Stripe Removal

The legacy code imported `createStripeCustomer` from `services/stripeService` and called it inside the `signUp` function. The current plan is to remove all references to Stripe. To do so:

1. **Comment out or delete** the import of `createStripeCustomer` in `AuthContext.tsx`.
2. **Comment out** the call to `createStripeCustomer(data.user.id, email, fullName)` and the associated console logs.
3. **Add** insertion logic for `user_profiles` and `student_profiles` as described above.
4. Clean up environment variables – remove `VITE_STRIPE_PUBLIC_KEY` and `VITE_STRIPE_PRICE_ID` from `.env`. Remove unused components like `SubscriptionStep` and `PaymentStep`.

6.4 Security Considerations

Several security improvements were discussed:

- **Rate Limiting** – login and sign-up endpoints should be rate limited. Supabase does not provide built-in rate limiting, so a middleware using Edge Functions (e.g. checking number of attempts per IP) can be deployed. Alternatively, use a third-party like Cloudflare Turnstile.

- **Two-Factor Authentication** – the current implementation uses email or phone OTP in a basic way. Integrate a robust 2FA library or Supabase's built-in multi-factor authentication to secure admin and teacher accounts.

- **Role-Based Access Control** – currently enforced on the client. RLS policies must ensure that only appropriate roles can read or write certain tables. For example, teachers should not read `admin_settings` and students should not insert into `subscriptions`.

- **Secret Management** – environment variables should not be hard coded. Provide an `.env.example` file and ensure `JWT_SECRET`, `STRIPE_WEBHOOK_SECRET` and other sensitive keys are loaded from environment.

---

7. Supabase Database and RLS Policies

7.1 Key Tables

The following tables underpin LiqLearns:

Table	Purpose	Key Columns
`auth.users` (Supabase built-in)	Authentication table storing email and hashed password.	`id`, `email`, `encrypted_password`
`user_profiles`	Main user profile table storing username, full name, role, sponsor, and contact info.	`id` (FK to `auth.users.id`), `username`, `full_name`, `role`, `sponsor_username`, `phone`, `subscription_plan`
`student_profiles`	Extends user profile for students; stores subscription info, XP, streak, aura points.	`id` (PK, FK), `subscription_plan`, `trial_end_date`, `xp`, `gold`, `streak`, `level`, `aura_points`
`teacher_profiles`	Similar to student profile but for teachers; stores rating, earnings, etc.	`id`, `rating`, `total_earnings`
`courses`	Stores courses.	`id`, `title`, `description`, `creator_id`, `lesson_type`
`marketplace_items`	Items available for purchase or free download.	`id`, `title`, `description`, `price`, `tags` (array), `download_url`, `points_cost`
`study_rooms`	Metadata about public/ private study rooms.	`id`, `name`, `description`, `is_public`, `host_id`
`study_room_members`	Many-to-many linking users to rooms.	`room_id`, `user_id`
`events`	Student and teacher events.	`id`, `title`, `description`, `start_time`, `end_time`, `type`, `host_id`
`support_tickets`	Help centre tickets.	`id`, `creator_id`, `title`, `category`, `priority`, `status`
`ticket_messages`	Chat messages between support and users.	`id`, `ticket_id`, `sender_id`, `message`, `timestamp`

Table	Purpose	Key Columns
`quizzes`, `lessons`, `assignments`	Additional learning content.	...

7.2 RLS Policies

Supabase's Row-Level Security is key to ensuring that each role sees only what it should. The following are examples of policies and the issues discovered:

- **student_profiles**: The policy `students_manage_own_student_profiles` allows authenticated users to insert/update their own row if `id = auth.uid()`. Another policy `admins_view_all_student_profiles` allows admins to select all rows. This is generally correct.

- **study_rooms**: Policies were set for students to create rooms and update their own rooms. However, no `select` policy existed to allow reading public rooms. A correct policy would be:

```
CREATE POLICY students_view_public_rooms ON study_rooms
  FOR SELECT
  USING (is_public = true);
```

Without this, the query returns zero rows, causing the join failure.

- **study_room_members**: There was no policy enabling insertion into this join table, preventing users from joining rooms. The fix is:

```
CREATE POLICY students_join_rooms ON study_room_members
  FOR INSERT
  WITH CHECK ( user_id = auth.uid() );
```

- **user_profiles**: Initially only administrators could select other users because a misconfigured policy filtered everything. To allow everyone to view public profiles (e.g. for sponsor checks), a policy like `SELECT USING (true)` may be appropriate, or at least `role IN ('admin','student','teacher')`.

- **marketplace_items**: Items should be publicly readable (`SELECT USING (true)`), but only admins or the seller (teacher) should be able to `INSERT` or `UPDATE` (e.g. `WITH CHECK (auth.role() = 'admin' OR teacher_id = auth.uid())`).

Whenever a new feature is added (e.g. `events`, `support_tickets`), remember to create appropriate RLS policies; otherwise queries return 0 rows and the UI appears broken.

---

8. Real-Time and Performance Considerations

8.1 Real-Time Stats vs. Polling

The project attempted to display real-time stats (XP, streak, aura points) using Supabase Realtime. However, due to restrictive Content-Security-Policy settings, WebSocket connections were blocked, resulting in repeated “Realtime connection failed” logs. Debugging steps included:

1. **CSP Headers**: `vercel.json` or an `_headers` file must specify `Content-Security-Policy` that allows `wss://*.supabase.co` and `https://*.supabase.co`. Without this, browsers will block WebSocket connections. Similarly, `connect-src` should include `https://api.stripe.com` if Stripe is used.

2. **Realtime Enabled**: In the Supabase dashboard, ensure “Realtime” is turned on for the tables you want to subscribe to (e.g. `student_profiles`). Without enabling, subscription events will never fire.

3. **Environment Variables**: In production deployments on Vercel, ensure `VITE_SUPABASE_URL` and `VITE_SUPABASE_ANON_KEY` are correctly set. If these differ between preview and production, Realtime may silently fail.

Given these complexities and the limited number of stat updates, a simpler solution is to poll an Edge Function or a `student_stats` view every few seconds. An example Edge Function (`supabase/functions/student-stats`) could accept a user ID, fetch the row from `student_profiles`, return a JSON object of XP, gold, aura points and streak, and ensure a row exists by inserting default values if none. The React component can call this function using `supabase.functions.invoke('student-stats', { headers: { Authorization: Bearer ${session.access_token} } })` inside a `useEffect` with interval. This avoids WebSocket restrictions and still provides near-real-time updates.

8.2 Removing Mock Data

At times the StudentDashboard used fallback mock data such as `stats ?? mockStats` if the API call failed. This masked errors and allowed the UI to render seemingly correct values even when the backend failed. All mock/fallback references should be removed. Instead, display a loading state while fetching data and an error state if the query fails. This encourages developers to fix the actual API rather than hiding the problem.

8.3 Performance Improvements

The site audit identified several performance bottlenecks:

- **Bundle Size**: Large libraries like Recharts and Moment.js were included. Replace Moment.js with native `Date` or `date-fns` (already a dependency) and use lighter chart libraries or dynamic imports to reduce the initial bundle.
- **Image Optimisation**: Use `<img loading="lazy">` and size attributes so images don’t block initial render. Serve images through a CDN (Supabase Storage or Vercel) with compression.

- **Prefetching**: Use `link rel="prefetch"` in `index.html` to prefetch critical resources like fonts or other route bundles.
- **Caching**: Introduce a global `APP_VERSION` constant to bust caches when deploying new versions. Append `?v=${APP_VERSION}` to script and style URLs.

---

9. User Interface and Design Enhancements

9.1 Hero Section and Landing Page

Early feedback focused heavily on the **Hero Section** of the landing page. The user wanted a warmer, light orange gradient behind the heading and icons to better reflect the LiqLearns brand. After several revisions, the gradient was set to a soft orange (#fca311) fading into white, with the text left aligned and call-to-action buttons clearly visible on small screens. To ensure the hero did not overflow on mobile devices, the height class `h-screen` was replaced with `min-h-[calc(100vh-4rem)]` and `flex-col` layout to allow content to wrap.

The CTA buttons were updated to use gradient backgrounds on hover and improved contrast ratios for accessibility. A new **PricingCard** component was created to display subscription plans (although subscription selection was later removed) and uses icons to highlight plan features. The card supports a "Most Popular" ribbon and scales gracefully from mobile to desktop.

9.2 Accessibility and Colour Contrast

The site audit noted several **accessibility** violations: icon-only buttons lacked `aria-label`s, some text colours failed contrast ratios (e.g. badges with orange text on white), and focus outlines were removed globally by Tailwind preflight. Fixes implemented include:

- Adding `aria-label` attributes to all icon buttons (e.g. hearts, message icons).
- Adjusting colours using Tailwind's `text-orange-600` for better contrast or darkening grey text by 30 % to meet WCAG 2.1 AA guidelines.
- Re-enabling the focus ring by adding `focus-visible:outline` classes and removing the global `* { outline: none; }`.
- Providing closed captions for video lessons by including a `.vtt` file and setting `<track kind="captions" src="..." />` inside the video player.

9.3 Mobile Responsiveness and Overflow

Many pages used `h-screen` or fixed heights that caused content to overflow on small screens, hiding CTA buttons. Replacing these with `min-h-screen` or dynamic heights (e.g. `min-h-[calc(100vh-4rem)]`) and using `flex-col` layouts ensures that content flows vertically. For long forms (like sign-up), using a sticky progress indicator at the top helps users track their progress and prevents important buttons from being scrolled out of view.

---

10. Gamification, Questing and Leaderboards

10.1 Gamification Layer

To make learning engaging, a gamification layer was proposed. Key components include:

- **XP and Gold**: Completing lessons, quizzes and quests grants XP and gold. XP determines level; gold can be spent in the marketplace.
- **Daily Streaks**: Logging in and completing tasks daily increases your streak. A current streak counter is displayed on the dashboard. When the streak ends, an animation plays (e.g. fireworks) to celebrate the milestone.
- **Badges and Achievements**: Earn badges for reaching milestones (e.g. 100 XP, 30-day streak). A `badge_profiles` table and triggers maintain badge ownership. Badges appear in the student profile.
- **Leaderboards**: A leaderboard shows top students by XP or number of recruits (for multi-level marketing). The query uses a materialized view joining `user_profiles` with a `mlm_network` table to compute downline counts. The leaderboard can be filtered by period (daily/weekly/all-time) and by metric (XP/gold/streak).
- **Quest System**: Daily quests (e.g. complete 5 flashcards) are generated via an edge function `generate-daily-quests`. The function ensures variety by selecting different activity types and awarding random XP/gold. A `quests` table stores quest templates and a `daily_quests` table stores assignments per user per day. Users can claim rewards via an API call that updates their XP and marks the quest as complete.

Implementing these features requires additional tables (e.g. `user_activities`) and triggers to increment XP and streak upon completion of tasks. RLS policies must ensure only the rightful user can update their own stats.

10.2 Multi-Level Marketing (MLM) Features

The platform has a sponsorship model where users can specify a **sponsor** upon sign-up. A `mlm_network` table stores relationships: `sponsor_id` and `child_id`. This network allows referral bonuses and a leaderboard of top sponsors. Rules were defined via an edge function `check-sponsor` to enforce that admin/support roles cannot be sponsors. A `check-username` edge function ensures uniqueness of usernames. The sign-up form uses `apiClient` with a debounced `useEffect` to call these functions and show real-time validation messages.

---

11. Security, Privacy and Compliance

11.1 Environment Management and Secrets

One of the earliest tasks was creating a `.env.example` file. This file lists environment variables (e.g. `VITE_SUPABASE_URL`, `VITE_SUPABASE_ANON_KEY`, `JWT_SECRET`, `STRIPE_WEBHOOK_SECRET`) but contains placeholder values. Team members can copy this to `.env` and insert real keys. Hard-coded secrets in the codebase must be removed. In CI/CD, the environment is loaded from Vercel's Secrets.

11.2 Content Security Policy (CSP)

The application lacked proper security headers. Adding a `vercel.json` or `_headers` file with a strict `Content-Security-Policy` mitigates XSS and clickjacking. A recommended CSP includes:

```
default-src 'self';
script-src 'self' https://cdn.jsdelivr.net;
style-src 'self' 'unsafe-inline';
img-src 'self' data: https://avatars.githubusercontent.com https://
*.supabase.co;
connect-src 'self' https://*.supabase.co wss://*.supabase.co;
frame-ancestors 'none';
base-uri 'none';
```

This allows Supabase API and WebSocket connections while blocking other domains. Additional domains like Stripe or analytics providers can be appended as needed.

11.3 GDPR, Cookie Consent and Account Deletion

The platform must comply with GDPR and other privacy laws. An early audit recommended:

- Displaying a **cookie consent banner** on first visit, explaining which cookies are used and allowing opt-in/out. Accepting stores the preference in local storage.
- Creating a **data export** tool where users can download all data associated with their account (from `user_profiles`, `student_profiles`, `user_activities`, etc.). A Supabase edge function can gather data and return a JSON file.
- Implementing an **account deletion** flow: from the settings page, users can request deletion. This triggers a confirmation email. On confirmation, an edge function deletes the user from `auth.users` and cascades to related tables (using `ON DELETE CASCADE`).

11.4 Security & Compliance Management Centre

For administrators, a dedicated page summarises security tasks: verifying Stripe webhook signatures, ensuring JWT secrets rotate periodically, auditing RLS policies, and monitoring rate limiting. Each task appears as a checklist item with status (open, in progress, resolved). Implementing this reduces the risk of leaving critical vulnerabilities unresolved.

---

12. Developer Experience and Testing

12.1 Code Organisation and Conventions

The codebase mixes components in `pages` and `components`. A proposed organisation is:

- `src/components/` for generic UI elements (buttons, form fields, cards, modal wrappers).
- `src/features/` for domain-specific logic (e.g. `auth`, `marketplace`, `gamification`).

- `src/pages/` for route-level components only.
- `src/contexts/` for global state providers (auth, theme, settings).

Naming conventions should be consistent (e.g. `StudentDashboard`, `TeacherDashboard`, `AdminDashboard` rather than mixing PascalCase and camelCase). Logs should use a unified logger instead of scattered `console.log` and `console.error`. Prettier and ESLint should be configured to enforce code style.

12.2 Testing and CI

Currently there is no automated test suite. Adding **unit tests** with Jest and **integration tests** with React Testing Library will catch UI regressions. For critical flows (sign-up, login, marketplace purchase), consider end-to-end tests using Cypress or Playwright. A **CI pipeline** (GitHub Actions) should run tests on every pull request, lint the code, and optionally run **Lighthouse CI** to track performance metrics (FCP, TTI). Add a `coverage` badge to the README to encourage comprehensive tests.

12.3 Supabase Migrations and Edge Functions

The `supabase` directory contains SQL migrations (e.g. `20260105154700_add_user_activities_table.sql`) and TypeScript edge functions. Use `supabase db push` to apply migrations locally and `supabase functions deploy` to deploy edge functions. Keep migrations atomic and idempotent – each file should modify one aspect (adding a table, adding RLS, seeding data). Use semantic versioning in filenames (YYYYMMDDHHMMSS). Document functions and triggers in a `SUPABASE_SCHEMA.md` so new team members can understand the database.

Conclusion

LiqLearns is a feature-rich platform with a broad vision: real-time study rooms, an integrated marketplace, gamified learning journeys, multi-level marketing referrals, administrative dashboards and support tools. Through the debugging sessions, we uncovered numerous issues – from simple UI bugs (blank modals, unresponsive buttons) to deeper architectural flaws (broken RLS policies, missing API endpoints) and security concerns (hard-coded secrets, missing CSP). The report summarises every feature, the associated files, and the recommended fixes.

Key recommendations moving forward:

1. **Simplify sign-up** – finalise the removal of Stripe and ensure proper insertion of profile rows with a free trial. Validate username and sponsor via edge functions.
2. **Seed the database** – populate `courses`, `marketplace_items`, and `quests` with real content. Ensure RLS policies allow read access.
3. **Implement missing handlers** – connect UI buttons to Supabase calls for study rooms, events, teacher content, and admin management. Add error handling and loading states throughout.
4. **Fix real-time** – adjust CSP, enable Realtime on needed tables, or use polling functions for stats. Remove mock fallbacks.
5. **Complete security tasks** – supply `.env.example`, add CSP and HSTS headers, rate-limit auth flows, implement GDPR and account deletion.

6. **Invest in developer tooling** – set up automated tests, CI, and consistent code formatting. Document the Supabase schema and edge functions.

Once these improvements are in place, LiqLearns will provide a robust, secure and engaging learning experience for all user roles.
