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