PODSLICE: step-by-step implementation of a comprehensive user account settings page,

Summary

This technical documentation outlines the **step-by-step implementation of a comprehensive user account settings page** designed to manage subscriptions, personal profiles, notifications, and security. The primary goal is to provide users with a **centralized hub** where they can control their membership, moving beyond the previously inadequate redirection to an "about" page for plan changes.

The initial phases focused on building the foundational **tabbed interface** and robust **subscription management**, including API endpoints for upgrades, downgrades, cancellations, and displaying billing history. Subsequent phases meticulously developed the **profile management** section, allowing users to edit their name, email, and avatar. This was followed by the **notification preferences** tab for controlling email and in-app alerts.

The final stage, detailed in the last output, completes the account settings with a **security tab**, providing functionalities for password changes, two-factor authentication toggles, session management, and a critical account deletion process. Throughout the development, a strong emphasis was placed on **mock data and test controls** to ensure thorough testing in a development environment before production, ensuring a secure, user-friendly, and fully functional account management system.User Auth, Roles and Permissions

User Authentication Layer

User sign-in is primarily handled by Clerk, which acts as the **authentication layer** for the application.

Here's a breakdown of how it functions:

- **Clerk's Core Role**: Clerk is responsible for managing user sign-up and sign-in processes, providing **unique user IDs** for each authenticated user.
- **Integration with Prisma**: While Clerk handles authentication, **user data and subscription entitlements are stored in the Prisma database**, linked to the user's unique ID obtained from Clerk.
- **Clerk Limitations on Free Plan**: It's important to note that Clerk's free plan **does not support custom roles** (such as 'Casual Listener' or 'Curate & Control') or custom permissions for production use. Therefore, feature access is determined by querying the Prisma database, not by Clerk roles. Clerk on the free plan only provides default roles like `org:admin` and `org:member`.
- **Workflow**: When a user signs up or logs in, **Clerk manages their session and provides a unique user ID**. This ID is then used by the application's backend logic to query the Prisma database for the user's subscription tier and other preferences.

`User`

User data is primarily stored within the **Prisma database**, which acts as the **definitive source of truth** for various aspects of a user's account, including their profile, preferences, and entitlements.

Specifically, the `User` **model** in your Prisma schema contains several fields that directly store user data:

- `user_id`: A unique identifier for the user.
- `name`: Stores the user's full name.
- `email`: Stores the user's unique email address.
- `password`: Stores the user's password.
- `image`: Stores the URL for the user's profile picture (avatar).
- `email_verified`: A timestamp indicating when the user's email address was verified.
- `is_admin`: A boolean flag that indicates if the user has administrator privileges.
- `email_notifications`: A boolean flag that determines if the user receives email notifications.
- `in_app_notifications`: A boolean flag that determines if the user receives in-app notifications.
- `created_at`: The timestamp when the user's account was created.
- `updated_at`: The timestamp of the last update made to the user's record.
- `paystack_customer_id`: Stores the unique customer ID from Paystack, used for managing billing.

Beyond these direct fields, the `User` model also defines **relations to other tables** which indirectly store or link to more user-specific data and entitlements. Notably, the `subscriptions` field links to the `Subscription` model, which is crucial for managing **user entitlements**. The `Subscription` model contains fields such as:

- `paystack_plan_code`: This code identifies the specific Paystack plan the user is subscribed to (e.g., for 'Casual Listener' or 'Curate & Control' tiers).
- `status`: Indicates the current status of the subscription (e.g., "active", "canceled", "trialing").

> Important `User` Note:
> Clerk's free plan does not support custom roles, **subscription tiers and membership variants (like 'FreeSlice', 'Casual Listener', and 'Curate & Control') are managed and stored directly in your Prisma database**, rather than being tied to Clerk roles. When a user attempts to access a feature tied to a specific membership variant, the application's backend queries the Prisma database to check their `subscriptionTier` and grant or deny access accordingly.

User entitlements

User entitlements are primarily stored in the **Prisma database**, which serves as the **definitive source of truth** for a user's subscription status and tier.

Here's a breakdown of why and how this is managed:

- **Clerk's Limitations**: While Clerk handles user authentication and provides unique user IDs, its free plan **does not support custom roles** (such as 'Casual Listener' or 'Curate & Control') or custom permissions for production use. This limitation means that Clerk cannot be used for application-level authorisation based on subscription tiers [conversation history].
- **Prisma's Role**: Due to Clerk's limitations, the specific membership variants – **FreeSlice**, **Casual Listener** (£5/month), and **Curate & Control** (£10/month) – are **stored directly in the Prisma database**. This data is linked to the user's Clerk ID.
- **Workflow for Entitlement Updates**:
  - When a user successfully makes a payment or a subscription is updated via Paystack (the payment gateway), **Paystack notifies the backend via webhooks**.
  - The backend then **updates the user's** `subscriptionTier` **in the Prisma database**.
  - This ensures that the Prisma database always holds the most current and definitive information about a user's entitlement.
- **Access Control Implementation**: When a user attempts to access a feature that is tied to a specific membership variant (e.g., "Advanced AI curation" for Curate & Control users), the application's **backend logic queries the Prisma database** for the user's `subscriptionTier` to determine whether to grant or deny access, rather than relying on Clerk roles. This is managed by a **custom access control system**.
- **Account Settings Integration**: The comprehensive Account Settings page displays the user's **current plan directly from Prisma**. User preferences like `email_notifications` and `in_app_notifications` are also stored in the Prisma `User` model.

Subscriptions

Subscriptions are managed through a robust system primarily leveraging **Paystack** for payments and a **Prisma database** as the definitive source of truth for user entitlements, with **Clerk** handling user authentication. This architecture was designed to accommodate specific membership variants: **FreeSlice**, **Casual Listener** (£5/month), and **Curate & Control** (£10/month).

Member Subscription Tiers Management

Subscription tiers are **primarily managed in the Prisma database**, which serves as the **definitive source of truth** for a user's subscription status and tier.

Here's a detailed breakdown of how they are managed:

- **Prisma Database as the Source of Truth**: The specific membership variants, namely **FreeSlice**, **Casual Listener** (£5/month), and **Curate & Control** (£10/month), are **stored directly in your Prisma database**. This data is linked to the user's unique ID obtained from Clerk. The `User` model in the Prisma schema includes fields relevant to a user's subscription, and the `Subscription` model holds details like `paystack_plan_code` and `status`.
- **Clerk's Limitations**: Clerk, while handling user sign-in and providing unique user IDs, **does not support custom roles** like 'Casual Listener' or 'Curate & Control' on its free plan for production use. This means Clerk's default roles (`org:admin`, `org:member`) cannot be used to determine feature access based on subscription tiers. Consequently, feature access is **determined by querying the Prisma database**, not Clerk roles.
- **Paystack's Role in Updating Tiers**: **Paystack** is the designated payment gateway that manages **recurring payments and subscriptions**. When a user successfully makes a payment or a subscription is updated (e.g., upgrade or downgrade) via Paystack, Paystack **notifies the backend via webhooks**. Your application's backend then **updates the user's `subscriptionTier` in the Prisma database** based on these notifications.
- **Workflow for Managing Membership Variants**:
  a. **User Authentication (Clerk)**: A user signs up or logs in using Clerk, which provides a unique user ID and manages their session.
  b. Private (https://app.clickup.com/t/8699ywt1h)
  c. **Feature Authorisation (Application Logic & Prisma)**: When a user attempts to access a feature tied to a specific membership variant (e.g., "Advanced AI curation" for 'Curate & Control' users), your application's **backend logic queries the Prisma database** using the user's ID (from Clerk) to check their `subscriptionTier`. Based on this retrieved tier, your application grants or denies access to the specific features.
- **Account Settings Integration**: The comprehensive Account Settings page allows users to view their **current plan information directly from Prisma**. This page now includes features for **upgrading, downgrading, and cancelling subscriptions** through integration with the Paystack API and displaying billing history. Specific API endpoints have been created for these actions, such as `/api/account/subscription/upgrade`, `/api/account/subscription/downgrade`, `/api/account/subscription/cancel`, and `/api/account/subscription/billing-history`.
- **Frontend State Management**: A dedicated `SubscriptionStore` (using Zustand) handles the state for subscription preferences on the frontend, managing loading states and errors, and includes actions like `upgradeSubscription`, `downgradeSubscription`, `cancelSubscription`, and `loadBillingHistory`.

User Account Settings Page

There are **four** main settings tabs that exist within the account settings page.

These tabs are:

- **Subscription Management**: This tab handles all aspects of user subscriptions, including displaying the current plan, enabling upgrades/downgrades, showing billing history, and managing cancellations.
- **Profile Settings**: This section allows users to manage their personal information, such as their name and email, and includes functionality for uploading or removing a profile picture.
- **Notification Preferences**: Users can manage how they receive notifications, including toggling email and in-app notifications, and seeing "always on" notification types like subscription or system updates.
- **Security Settings**: This tab provides options for account security, such as changing passwords, enabling two-factor authentication, managing active sessions, and initiating account deletion.

Here's a detailed breakdown of how subscriptions are managed:

Core Components and Their Roles:

- **Clerk**: Serves as the **authentication layer**, managing user sign-up, sign-in, and providing unique user IDs. Critically, on its free plan, Clerk only provides default roles (`org:admin` and `org:member`) and **does not support custom roles** like 'Casual Listener' or 'Curate & Control' for production use. This means Clerk is *not* used for application-level authorisation based on subscription tiers.
- **Paystack**: This is the designated **payment gateway**, handling all financial aspects of subscriptions, including recurring payments and tokenization. Paystack provides an API for programmatically managing subscriptions (updating, pausing, cancelling, viewing), but it **does not offer application-level authorisation** or user roles within your platform's features.
- **Prisma Database**: This is the **central source of truth** for a user's subscription status and tier. Due to Clerk's limitations, the specific membership variants (e.g., 'FreeSlice', 'Casual Listener', 'Curate & Control') are stored directly in the Prisma database, linked to the user's Clerk ID.

Workflow for Membership Variant Management:

- **User Authentication**: A user signs up or logs in via Clerk, obtaining a unique user ID and session.
- **Subscription Purchase**:
  - For a **FreeSlice** user, a record reflecting this tier exists in the Prisma database.
  - For **Casual Listener** or **Curate & Control**, the user initiates a payment flow through Paystack within the application.
- **Backend Update (via Paystack Webhooks & Prisma)**: Upon successful payment or subscription updates, Paystack notifies the backend via webhooks. The backend then updates the user's `subscriptionTier` in the Prisma database.
- **Feature Authorisation**: When a user attempts to access a feature tied to a specific membership variant (e.g., "Advanced AI curation" for Curate & Control users), the application's backend logic

queries the Prisma database for the user's `subscriptionTier` to determine access, rather than relying on Clerk roles.

Account Settings for Subscription Management:

- A **dedicated `/account` page** was created with a modern, tabbed interface to solve the problem of users being incorrectly redirected to `/about` when trying to change their plan.
- The **"Subscription Management" tab** within this page provides comprehensive controls for users, addressing previous missing functionalities. Key features include:
  - **Current Plan Display**: Shows the user's current subscription status directly from Prisma.
  - **Upgrade/Downgrade Flow**: Users can change their plans using Paystack API endpoints (`/api/account/subscription/upgrade` and `/api/account/subscription/downgrade`). This includes smart options showing only relevant choices based on the current plan.
  - **Billing History**: Users can view past payments and transactions via the `/api/account/subscription/billing-history` endpoint.
  - **Cancellation Flow**: Users can cancel their subscriptions via the `/api/account/subscription/cancel` endpoint, with confirmation dialogs to ensure safe cancellation.
  - **Real-time Status Updates**: The system displays the current subscription status and billing dates.
  - **Plan Comparison**: Available plans are displayed for easy comparison.
  - **Error Handling and Loading States**: Comprehensive error messages and loading indicators are implemented for a better user experience.

Technical Implementation Details:

- **API Endpoints**: Dedicated backend routes are created to handle specific subscription actions (upgrade, downgrade, cancel, billing history).
- `useSubscriptionStore`: A Zustand store manages the subscription state, including the current plan, available tiers, billing history, loading states, and error handling, along with actions to interact with the subscription APIs.
- **Mock Data and Test Controls**: For development, mock subscription data and test controls are provided, allowing easy switching between different subscription states (FreeSlice, Trial, Casual, Premium, Canceled) to facilitate testing without real Paystack transactions.

This comprehensive approach ensures that users have full control over their subscriptions within a centralized and user-friendly account management experience.

Recurring Payment Management

Paystack.

- **Paystack as the Payment Gateway**: **Paystack** is explicitly designated as the **payment gateway** for the application. Its primary role is to handle all financial aspects of subscriptions.
- **"Subscriptions" Feature**: Paystack's "Subscriptions" feature allows the system to **schedule payments at regular intervals**. This is crucial for managing the monthly subscription tiers, such as 'Casual Listener' (£5/month) and 'Curate & Control' (£10/month).
- **Tokenisation**: Paystack supports accepting recurring credit card payments through **tokenisation**, which is a key part of its subscription feature.
- **API for Management**: While Paystack focuses on the financial aspects, it provides an API (Application Programming Interface) that allows your application to **programmatically update, pause, cancel, and view subscription information**.
- **No Application-Level Authorisation**: It's important to note that Paystack's functionalities are centred on financial transactions and business management within its own dashboard, and it **does not provide features for managing end-user roles or permissions within your application's features**. The application's own logic and Prisma database are used for that.

Handling of credit card details

User payments and the handling of credit card details are managed by Paystack, which serves as the designated payment gateway for the application.
Here's how this process works:

Managed by Paystack

Paystack's Core Role:

Paystack is explicitly designated as the payment gateway and is responsible for managing all financial aspects of subscriptions, including recurring payments. It handles the technical processing of these financial transactions.

Where Credit Card Details are Added:

When a user wishes to subscribe to a paid tier (such as 'Casual Listener' or 'Curate & Control'), they initiate a payment flow directly through Paystack within the application. Paystack's "Subscriptions" feature supports accepting recurring credit card payments through tokenisation, meaning the sensitive card details are securely handled by Paystack, not directly by your application.

Payment Flow

1. When a user opts to subscribe to a paid tier, such as 'Casual Listener' or 'Curate & Control,' they initiate the payment process within your application.
2. This process involves interacting with Paystack to initiate a transaction. For instance, when a user upgrades or downgrades their plan, the application calls an API endpoint, such as `/api/account/subscription/upgrade` or `/api/account/subscription/downgrade`.
3. The endpoint communicates with the Paystack API to initiate the transaction, typically providing an `authorization_url` or `checkoutUrl`.
4. The user is then redirected to Paystack's secure checkout environment via the `authorization_url` or `checkoutUrl`, where they enter their credit card details or other payment information.
5. Upon successful payment, Paystack processes the transaction and notifies your backend through webhooks.
6. Your backend updates the user's subscription tier in the Prisma database, which serves as the definitive source of truth for their entitlements.
    a. **API for Management:** Paystack provides an API that allows your application to programmatically update, pause, cancel, and view subscription information, ensuring effective subscription management. Additionally, Paystack offers functionalities within its dashboard for business management.
    b. **No Application-Level Authorization:** It's important to note that Paystack's functionalities focus on the financial aspects of subscriptions and do not extend to managing end-user roles or permissions within your application's features.

In summary, Paystack is the dedicated service that handles the technical and financial processing of recurring payments for subscription tiers.

Email Notifications

Email notifications are managed through a dedicated section within the comprehensive **Account Settings** page, leveraging the **Prisma database** as the core data store and a specific **API endpoint** for updates.

Here's a breakdown of how email notifications are managed:

Data Storage:

- The user's preference for email notifications is stored directly in the **Prisma database** within the `User` model, specifically using the `email_notifications` field (which is a Boolean type).

User Interface (UI):

- A **"Notification Preferences" tab** is part of the `/account` settings page. This tab is where users interact with their notification settings.

- Within this tab, there is an **"Email Notifications" toggle** that allows users to enable or disable receiving notifications via email.
- The interface also includes **"Quick Actions"** buttons like "Enable All" and "Disable All" for bulk management of notification preferences.
- Certain notification types, such as "Subscription Updates," "New Episodes," and "System Updates," are explicitly displayed as **"Always On"**, indicating they cannot be toggled by the user.
- A **custom** `Switch` **UI component** was created for the toggles to avoid adding new dependencies to the codebase.

Frontend State Management: A dedicated `NotificationStore` (implemented using Zustand) handles the state for notification preferences on the frontend, managing loading states and errors. This store includes actions such as `loadPreferences`, `updatePreferences`, and `toggleEmailNotifications`.

Backend API:

- A specific `/api/account/notifications` API endpoint is used to interact with these preferences.
- A `GET` request to this endpoint retrieves the current `email_notifications` and `in_app_notifications` settings from the Prisma database.
- A `PATCH` request to this endpoint allows for **updating the user's notification preferences**, directly modifying the `email_notifications` and `in_app_notifications` fields in the Prisma `User` model.

Testing:

For development and testing, **mock notification preferences data (** `MOCK_PREFERENCES` **)** is provided, and API calls are simulated with delays to mimic real backend behavior. This allows for comprehensive testing of the email notification management functionality without requiring a live database or API.

Mock Data and Test Controls:

For development, mock subscription data and test controls are provided, allowing easy switching between different subscription states (FreeSlice, Trial, Casual, Premium, Canceled) to facilitate testing without real Paystack transactions.