Overview of the Frontend Codebase

The codebase is built using **React Native** with **TypeScript** and leverages several libraries and frameworks:

- **Expo Router**: For navigation and routing between screens.
- **NativeWind**: For styling components using Tailwind CSS classes in React Native.
- **Axios**: For making HTTP requests to the backend API.
- **i18next**: For internationalization and localization support.
- **Day.js**: For date manipulation.
- **Lucide Icons**: For vector icons.
- **React Native Modal**: For creating modal dialogs.

The application consists of multiple screens/components:

1. **Login Screen**: Allows users to log in to the application.
2. **Register Screen**: Enables new users to sign up.
3. **Home Screen**: Displays user-specific information and navigation options.
4. **Courses Screen**: Shows a list of courses the user is enrolled in or teaches.
5. **Profile Screen**: Displays user profile information and allows editing.
6. **Admin Dashboard**: Provides administrative functionalities.

Each screen is implemented as a React functional component, and navigation between screens is handled via the Expo Router.

Detailed Explanation of Each Component/Page

1. Login Screen (`Login.tsx`)

Purpose: Allows users to authenticate using their email and password.

Key Features:

- **Form Validation**: Validates email and password fields using utility functions from `utils/validation`.
- **API Interaction**: Sends a POST request to `/api/auth/login` to authenticate the user.
- **Loading Overlay**: Displays a loading indicator while the authentication request is in progress.
- **Error Handling**: Shows error messages for invalid credentials or server errors.
- **Third-Party Logins**: Includes buttons for Google and WhatsApp login (functionality to be implemented).
- **Navigation**:

    - Redirects to the home page upon successful login using the `router.replace` method.
    - Provides a link to the registration page for new users.
    - Includes a "Forgot Password" link.

Styling:

- Uses `nativewind` for styling with Tailwind CSS classes.
- Adjusts styles based on the current color scheme (light or dark mode).

Localization:

- Text content is localized using the `useTranslation` hook from `i18next`.

2. Register Screen (`Register.tsx`)

Purpose: Enables new users to create an account.

Key Features:

- **Multi-Step Form**: Divided into multiple stages:
    - Email Stage
    - Personal Details Stage
    - Address Details Stage
    - Password Stage
- **Form Validation**: Validates inputs at each stage before proceeding to the next.
- **Animated Transitions**: Uses the `Animated` API for smooth transitions between stages.
- **Password Strength Indicators**: Checks for alphanumeric characters, special characters, and minimum length.
- **API Interaction**: Sends a POST request to `/api/auth/register` to create a new user.
- **Error Handling**: Displays error messages for invalid inputs or registration failures.
- **Navigation**:
    - Includes "Next" and "Previous" buttons to navigate between form stages.
    - Provides a link back to the login page.

Styling and Localization:

- Similar to the Login Screen, it uses `nativewind` for styling and `i18next` for localization.

3. Home Screen (`HomePage.tsx`)

Purpose: Serves as the main landing page after login, displaying user information and navigation options.

Key Features:

- **User Profile**:
  - Fetches and displays user details from `/api/users/{id}/all`.
  - Shows the user's initials in a profile image placeholder.
- **Notifications**:
  - Includes a `NotificationsList` component to display user notifications.
  - Registers the device for push notifications.
- **Navigation Items**: Provides quick access to various sections:
  - Messages
  - Calendar
  - Learning (Courses)
  - Gallery
- **Header**:
  - Displays the user's full name.
  - Includes a button to view/edit the user's profile.
- **Styling and Theme**:
  - Adjusts styles based on the color scheme.
  - Uses an `ImageBackground` for the profile section.

4. Courses Screen (`CoursesPage.tsx`)

Purpose: Displays a list of courses the user is enrolled in (for students) or teaches (for teachers).

Key Features:

- **Course Fetching**:
  - Retrieves courses from `/api/users/{id}/courses`.
  - Maps course categories to specific icons.
- **Conditional Rendering**:
  - Shows an "Add Course" button for teachers, allowing them to create new courses.
- **Course Navigation**:
  - On selecting a course, navigates to the course details page using the router.
- **Add Course Modal**:
  - Uses the `AddCourseModal` component to handle course creation.
  - Refreshes the course list upon successful creation.
- **Error Handling**:
  - Displays alerts for errors during course fetching or creation.
- **Styling**:
  - Adjusts styles for dark and light modes..

5. Admin Dashboard (`AdminDashboard.tsx`)

Purpose: Provides administrative functionalities for managing the application.

Key Features:

- **Dashboard Buttons**:
  - Notifications
  - Analytics
  - Users
  - Manage Tasks
  - Manage App Settings
  - Manage Educational Content
- **Navigation**:
  - Each dashboard button navigates to a specific admin page.
- **Styling**:
  - Uses custom dashboard button components with icons and colors.
  - Arranged in a grid layout for accessibility.

6. Profile Screen (`Profile.tsx`)

Purpose: Displays detailed user profile information and allows the user to edit certain aspects like bio and profile picture.

Key Features:

- **User Information Display**:
  - Fetches and displays user details from `/api/users/{id}/all`.
  - Shows profile image, name, contact information, and address.
- **Role-Specific Sections**:
  - **Students**: Displays a list of assigned mentors.
  - **Teachers**: Displays a list of assigned students and allows editing of bio.
- **Bio Editing**:
  - Provides a modal to edit and update the user's bio.
  - Sends a PUT request to `/api/users/{id}/bio` to update the bio.
- **Profile Image Upload**:
  - Includes an `UploadComponent` for uploading certifications or profile images (implementation pending).
- **Settings and Logout**:
  - Provides buttons to access settings and logout.
  - **Settings Modal**: Uses a `Settings` component to adjust app settings like theme.
- **Navigation**:
  - Back button navigates to the previous screen.
  - Tapping on a student's name navigates to their profile (for teachers).
- **Error Handling**:
  - Displays alerts for errors during data fetching or updating.

- **Styling**:
    - Uses `nativewind` for responsive and theme-aware styling.
    - Adjusts styles based on the current color scheme.
- **Localization**:
    - Text content is localized using the `useTranslation` hook from `i18next`.

Common Utilities and Components

Utility Functions (`utils/validation.ts`)

- Provides validation functions for:
    - Email and phone number validation.
    - Password strength checking.
    - Error message retrieval based on error types.

Components

- **PageLayout**: A common layout component used across multiple screens for consistent styling and structure.
- **InputField**: A custom input component with built-in error handling and styling.
- **ActionButton**: A customizable button component used for actions like submit, next, and previous.
- **DatePickerField**: A component for selecting dates, used in the registration form.
- **NotificationsList**: Displays a list of notifications for the user.
- **DashboardButton**: Used in the Admin Dashboard for navigation to different admin functionalities.
- **AddCourseModal**: A modal component that allows teachers to add new courses.
- **UploadComponent**: Handles file uploads, such as certifications or profile images.
- **Settings**: A component for adjusting application settings like theme.

Navigation and Routing

- **Expo Router**: Used for navigating between screens.
- Dynamic routing is used for screens that require parameters, such as user IDs or roles (e.g., `/courses/[id]/[role]/index.tsx`).
- **Navigation Functions**:
    - `router.push`: Navigates to a new screen.
    - `router.replace`: Replaces the current screen with a new one.
    - `router.back`: Navigates back to the previous screen.
- **Local Search Params**:

    - `useLocalSearchParams`: Retrieves dynamic parameters from the route (e.g., user ID and role).

State Management and Data Flow

- **Local State**: Managed using the `useState` hook for component-specific state.
- **Effects**:
    - `useEffect` is used for side effects such as data fetching and registering for push notifications.
- **Forms**:
    - Form inputs are controlled components with state variables tracking their values.
    - Validation states are maintained using separate state variables for errors.
- **Modals**:
    - `react-native-modal` is used for displaying modals, such as the bio edit modal and settings modal.

Themes and Styling

- **NativeWind**: Enables the use of Tailwind CSS classes in React Native components.
- **Color Scheme Detection**:
    - `useColorScheme`: Detects whether the app is in dark mode or light mode.
    - Styles and colors adjust based on the color scheme.
- **Tailwind Configuration**:
    - Custom colors and themes are defined in `tailwind.config`.
    - Components use classes like `bg-gray-800`, `text-white`, etc., which adjust according to the theme.
- **Dynamic Styling**:
    - Components adjust styles dynamically based on the current theme.
    - Icons and text colors change to maintain readability.

API Interactions

- **Axios**: Used for making HTTP requests to the backend API.
- **Endpoints**:
    - **Authentication**:
        - `POST /api/auth/login`: User login.
        - `POST /api/auth/register`: User registration.
    - **User Data**:

        - `GET /api/users/{id}/all`: Fetches all user details.
        - `PUT /api/users/{id}/bio`: Updates the user's bio.
        - `GET /api/student/{id}/mentors`: Retrieves mentors assigned to a student.
        - `GET /api/mentor/{id}/students`: Retrieves students assigned to a mentor.
    - **Courses**:
        - `GET /api/users/{id}/courses`: Retrieves courses for a user.
        - `POST /api/courses`: Creates a new course (for teachers).
    - **Files**:
        - Endpoints for file uploads (implementation pending).
- **Error Handling**:
    - Try-catch blocks are used to handle errors during API calls.
    - Errors are logged to the console, and user-friendly alerts are displayed.

Localization

- **i18next**: Provides internationalization support.
    - New languages can be added easily by creating a localization JSON file under the locales folder in the frontend directory.
        1. Create a JSON file
        2. Populate the JSON file with key-value pairs. Each key should represent a unique identifier for the text, and the value should be the translated string in the target language.
        3. Add the new language in `languages.ts`. Create a Language object as defined in the file and add it to the language array.
        4. Lastly, in `frontend/app/_layout.tsx` import your new localization JSON and add it to the resources array under RootLayout.

```
import en from '../locales/en.json';
import fr from '../locales/fr.json';
import hi from '../locales/hi.json';
import gu from '../locales/gu.json';
```

```
const resources = {
  en: { translation: en },
  fr: { translation: fr },
  hi: { translation: hi },
  gu: { translation: gu },
};
```

- **Usage in frontend components**:
    - The `useTranslation` hook is used to access localized strings.
    - Keys like `t('loginButton')` are used to fetch the appropriate translation as defined in your JSON language files.
- **Placeholders**:
    - The `replacePlaceholders` utility function replaces dynamic values in localized strings.

Testing

We primarily focused on testing any utility functions or components reused many times throughout the app. We used Jest for testing the React Native components as well as the typescript utility modules. The tests render each component and ensure they have the correct properties and behaviors by simulating a user's actions and interactions with it. Below is a list of the modules and components tested.

- **Utils**
  - validation.ts
  - formatters.ts
- **Components**
  - ActionButton.tsx
  - StyledText.tsx
  - AdminDashboardButton.tsx
  - AddCourseModal.tsx