# Klariti Frontend Documentation

## What is Klariti LMS?

**Klariti** is a modern Learning Management System (LMS) designed to facilitate seamless education and training for students, teachers, and administrators. The platform provides a user-friendly interface for managing courses, tracking progress, and fostering collaboration among users. Built with cutting-edge technologies like **Next.js**, **React**, and **TypeScript**, Klariti ensures a scalable and secure environment for online learning.

### Purpose of the Website

The Klariti website serves as a centralized platform for:

- Students: Access learning materials, view peer profiles, and track academic progress.
- Teachers: Manage student lists, assign tasks, and review progress.
- Admins: Assign roles, manage users, and oversee platform operations.

### Simple User Flow

Here's how users interact with Klariti:

- Sign Up/Login: Users register with their email or phone number and verify their identity via OTP.
- Role Assignment: Admins assign roles (Student, Teacher, Admin) and relevant subjects or teachers.
- Dashboard Access: Users are redirected to role-specific dashboards (e.g., Student: My Learnings, Teacher: Student List).
- Interaction: Students access courses, teachers manage students, and admins oversee operations.
- Logout: Users can securely log out, clearing their session data.

## Overview

This document provides an overview of the frontend architecture and components for the Klariti application, a learning platform built with **Next.js**, **React**, and **TypeScript**. The frontend is structured to handle user authentication, role-based access, and various user interactions, leveraging modern UI components and state management.

## Project Structure

The frontend codebase is organized as follows:

```
/app
  /(auth)
    /login
    /logout
    /signup
    /verify-otp
  /(portal)
    /admin
      /assign-role
    /teacher
      /student-list
      /student/[studentId]
      /report-card/[studentId]
    /student
      /peers/[studentId]
      /teacher
/components
  /ui
    /button.tsx
    /card.tsx
    /input.tsx
    /country-dropdown.tsx
    /otp-input.tsx
/lib
  /api.ts
  /auth.tsx
  /user-context.tsx
  /utils.ts
/types
  /index.ts
```

### Key Directories and Files

- **/app**: Contains the main application routes, organized using Next.js App Router.
  - **/(auth)**: Routes for authentication-related pages ( `/login` , `/logout` , `/signup` , `/verify-otp` ).
  - **/(portal)**: Role-specific routes ( `/admin` , `/teacher` , `/student` ).
- **/components/ui**: Reusable UI components (e.g., `button.tsx` , `Card` , `Input` , `CountryDropdown` , `OTPInput` ).
- **/lib**: Utility and core files for API integration, authentication, and user management.
  - `api.ts` : Axios instance for API requests with interceptors for token and device ID handling.
  - `auth.tsx` : Authentication context for managing user state and logout functionality.
  - `user-context.tsx` : Context for managing detailed user information.
  - `utils.ts` : Utility functions for class merging and masking sensitive data (email, phone).
- **/types**: TypeScript interfaces and types for type safety across the application.

## Key Features and Components

### Authentication

The authentication system supports email and phone-based login/signup with OTP verification. Key components include:

**Login ( `/app/(auth)/login` )**

- Allows users to log in via email or phone number.
- Validates input using `validateIdentifier` (email or 10-digit phone).
- Sends OTP to the provided identifier and redirects to `/verify-otp` .
- Uses `CountryDropdown` for phone number country code selection.
- Stores `identifier` and `identifierType` in `localStorage` .

**Signup ( `/app/(auth)/signup` )**

- Collects user details (name, email, phone, gender).
- Validates inputs (name: letters/spaces, email: valid format, phone: 10 digits, gender: required).
- Sends OTPs to both email and phone, redirecting to `/verify-otp` .
- Generates a unique `deviceId` using `uuidv4` .

**Verify OTP ( `/app/(auth)/verify-otp` )**

- Handles OTP verification for login and signup.
- Supports resending OTPs with a 60-second cooldown and rate-limiting (15-minute lockout after excessive requests).
- Uses `OTPInput` component for 6-digit OTP entry.
- Updates `localStorage` with tokens and user data upon successful verification.

**Logout ( `/app/(auth)/logout` )**

- Clears `localStorage` and redirects to `/login` .
- Ensures user is logged in before displaying the logout confirmation.

### Role-Based Access

The application supports multiple user roles ( `Student` , `Teacher` , `Admin` , `Super Admin` ), with role-specific routes and functionality.

**Admin: Assign Role ( `/app/(portal)/admin/assign-role` )**

- Allows Admins/Super Admins to assign roles and subjects to users.
- Fetches users without roles via `/admin/users` API.
- Supports role selection ( `Student` , `Teacher` , `Admin` , `Super Admin` for Super Admins).
- For `Student` role, allows selection of one subject and a teacher.
- Uses `Dropdown` for role and teacher selection, with subject checkboxes.

**Teacher: Student List ( `/app/(portal)/teacher/student-list` )**

- Displays a list of students assigned to the logged-in teacher.
- Each student card shows name, subjects, and a student ID badge.
- Clicking a card navigates to the student's profile ( `/teacher/student/[studentId]` ).
- Includes a "Report Card" button to view student progress ( `/teacher/report-card/[studentId]` ).

**Student/Teacher Profiles ( `/app/(portal)/student/peers/[studentId]` , `/app/(portal)/student/teacher` )**

- Displays detailed user information in tabs ( `Personal Info` , `Academic/Work Info` , `Additional Info` ).
- Includes profile image, name, email, gender, subjects, join date, and profile details (bio, hobbies, skills, etc.).
- Uses `Tabs` component for navigation between info categories.

### Video Playback

**VideoPlayerPopup ( `/components/VideoPlayerPopup.tsx` )**

- A modal popup for playing videos using `ReactPlayer` .
- Features play/pause, volume control, playback speed (0.5x, 1x, 1.5x, 2x), and full-screen toggle.
- Includes a seekable progress slider and time display.
- Prevents video downloading via `controlsList="nodownload"` .

### UI Components

- **Button, Card, Input**: Reusable components from `/components/ui` , styled with Tailwind CSS.
- **CountryDropdown**: Allows country code selection for phone numbers, with a search feature.
- **OTPInput**: A 6-digit input field for OTP entry, with validation and disabled states.
- **Tabs**: Used in profile pages for organizing information into categories.

## Authentication Flow

**Login/Signup:**

- User enters email/phone (login) or name/email/phone/gender (signup).
- Input is validated, and an OTP is sent via `/auth/login` or `/auth/signup` API.
- `deviceId` is included in headers for device tracking.

**OTP Verification:**

- User enters OTP(s) in `/verify-otp` .
- For signups, both email and phone OTPs are required; for login, only one is needed.
- On success, a token is stored in `localStorage` , and the user is redirected based on role and first-login status.

**Session Management:**

- `AuthContext` ( `/lib/auth.tsx` ) fetches user data on mount via `/auth/me` .
- Token and `deviceId` are included in all API requests via `api.ts` interceptors.
- On 401 errors, `localStorage` is cleared, and the user is redirected to `/login` .

**Logout:**

- Clears `localStorage` and calls `/auth/logout` API.
- Generates a new `deviceId` and redirects to `/login` .

## Role-Based Routing

- **Unauthenticated Users**: Redirected to `/login` .
- **Students**:
  - Redirected to `/myclassroom` if no role or to `/timezone-setup` if first login or timezone not set.
  - Can view peer profiles ( `/student/peers/[studentId]` ) and teacher profile ( `/student/teacher` ).
- **Teachers**:
  - Redirected to `/teacher` or `/timezone-setup` if first login or timezone not set.
  - Can view student list ( `/teacher/student-list` ) and individual student profiles ( `/teacher/student/[studentId]` ).
- **Admins/Super Admins**:
  - Redirected to `/admin` or `/super-admin` .
  - Can assign roles ( `/admin/assign-role` ).
- **First-Login/Timezone Setup**:
  - Students and Teachers are redirected to `/timezone-setup` if `isFirstLogin` or `!isTimezoneSet` .

## API Integration

**Axios Instance ( `/lib/api.ts` )**

- Base URL: `process.env.NEXT_PUBLIC_API_URL` .
- Automatically adds `Authorization` (Bearer token) and `Device-Id` headers.
- Handles 401 errors by clearing `localStorage` and rejecting the promise.

**Key Endpoints:**

- `/auth/login` : Sends OTP for login.
- `/auth/signup` : Sends OTPs for signup.
- `/auth/verify-otp` : Verifies OTPs for signup.
- `/auth/verify-login-otp` : Verifies OTP for login.
- `/auth/me` : Fetches current user data.
- `/admin/users` : Fetches users for role assignment.
- `/admin/assign-role` : Assigns role, subjects, and teacher.
- `/users/teacher/[teacherId]/students` : Fetches students for a teacher.
- `/users/student/[studentId]` : Fetches student profile.
- `/users/student/[studentId]/peers` : Fetches teacher profile for a student.

## State Management

**React Context:**

- `AuthContext` ( `/lib/auth.tsx` ): Manages user authentication state ( `user` , `loading` , `deviceId` , `logout` ).
- `UserContext` ( `/lib/user-context.tsx` ): Manages detailed user information ( `userDetails` , `loading` ).

**Local State:**

- Components use `useState` for local state (e.g., form inputs, loading states, errors).
- `useEffect` is used for side effects like fetching data or handling redirects.

**Local Storage:**

- Stores `token` , `deviceId` , `isLoggedIn` , `isVerified` , `userIdentifier` , `identifierType` , `userEmail` , `userPhone` .

## UI/UX Features

- **Responsive Design**: Uses Tailwind CSS for responsive layouts, with grid and flexbox for card-based UIs.
- **Animations**: Leverages `framer-motion` for smooth transitions and hover effects (e.g., card scaling, fade-ins).
- **Error Handling**: Displays user-friendly error messages via `react-hot-toast` for API failures and validation errors.
- **Interactive Elements**:
  - Cards have 3D hover effects using CSS transforms and perspective.
  - Proposers and dropdowns for role/teacher selection in `assign-role` .
  - Tabs for organizing profile information.
- **Accessibility**:
  - Uses ARIA attributes ( `aria-label` , `aria-checked` ) for interactive elements.
  - Ensures keyboard navigation support (e.g., gender selection in `signup` ).

## Dependencies

- **Next.js**: App Router for routing and server-side rendering.
- **React**: Component-based UI with hooks ( `localStorage` , `useEffect` , `useContext` ).
- **TypeScript**: Type safety with interfaces defined in `/types` .
- **Axios**: API requests with interceptors.
- **Tailwind CSS**: Utility-first CSS for styling.
- **Framer Motion**: Animations and transitions.
- **React Player**: Video playback in `VideoPlayerPopup` .
- **React Hot Toast**: Toast notifications for user feedback.
- **UUID**: Generates unique `deviceId` .
- **Lucide React**: Icons for UI elements.
- **ShadcnUI**: Reusable UI components ( `Button` , `Card` , `Input` , etc.).

## Setup Instructions

**Clone the Repository:**

```
git clone
cd klariti
```

**Install Dependencies:**

```
npm install
```

**Environment Variables:**

- Create a `.env.local` file in the root directory.
- Add the API base URL:

```
NEXT_PUBLIC_API_URL=https://api.klariti.com
```

**Run the Development Server:**

```
npm run dev
```

- Access the app at `http://localhost:3000` .

**Build for Production:**

```
npm run build
npm run start
```

## Future Improvements

- **Enhanced Accessibility**: Add more ARIA attributes and improve screen reader support.
- **Performance Optimization**: Implement lazy loading for images and components.
- **Testing**: Add unit and integration tests using Jest and React Testing Library.
- **Localization**: Support multiple languages for global accessibility.
- **Real-Time Features**: Integrate WebSocket for notifications or live updates.
- **Improved Error Handling**: Centralize error messages and add retry mechanisms for API failures.

## Conclusion

The Klariti frontend provides a robust, user-friendly interface for managing authentication, role-based access, and user interactions. Built with modern technologies and best practices, it ensures scalability, maintainability, and a seamless user experience. For further details, refer to the codebase or contact the development team.