

**School of Computer Science & Engineering**

**Department of Computer Science and Applications**

**SY BSc CS**

**Div –A**

**Year - 2025-26**

**Project Based Learning - Project Report**

**On**

**“KalaSetu”**

**Submitted By**

**Samyak Doshi-1272240315**

**Under Guidance of**

**Prof. Punam Chaudhari**



**School of Computer Science & Engineering**

**Department of Computer Science and Applications**

Certificate

This is to certify that, **Samyak Doshi** has successfully completed his/her Project Based Learning - project / Mini Project work entitled **“KalaSetu”** in partial fulfillment of SYBScCS Sem – III, for the year 2025-26. He has worked under our guidance and direction.

Date:

Place:

|  |  |  |
| --- | --- | --- |
| **Punam Choudhari**  **Project Guide** | **Dr. Ashish Kulkarni**  **Program Head** | **Dr. Anuradha Kanade**  **Program Director** |

A list of information on a white background

AI-generated content may be incorrect.

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **Contents** | **PAGE NO.** |
| **1** | **Introduction** |  |
| 1.1 | Existing System and Need for System |  |
| 1.2 | Scope of System |  |
| 1.3 | Operating Environment – Hardware and Software |  |
| 1.4 | Brief Description of Technology Used |  |
| **2** | **Proposed System** |  |
| 2.1 | Feasibility Study |  |
| 2.2 | Objectives of Proposed System |  |
| 2.3 | Users of System |  |
| **3** | **Analysis and Design** |  |
| 3.1 | System Requirements (Functional and Non-Functional requirements) |  |
| 3.2 | Module Hierarchy Diagram |  |
| 3.3 | Sample Input and Output Screens |  |
| 3.3 | Database |  |
| **4** | **Coding** |  |
| 4.1 | Code snippets |  |
| **5** | **Limitations of Proposed System** |  |
| **6** | **Proposed Enhancements** |  |
| **7** | **Conclusion** |  |
| **8** | **Bibliography** |  |

**1.Introduction :**

**Core Concept**

KalaSetu is a full-stack web application designed as an artisan marketplace. Its primary goal is to connect traditional artisans with users. The application is built using the MERN stack: React for the frontend, with Node.js and Express.js for the backend, and MongoDB as the database.

**Key Features**

The platform is divided into functionalities for artisans, users, and the overall system:

**For Artisans:**

* **Authentication:** Secure login and registration using email, phone, and Firebase OTP.
* **Profile Management:** Artisans can manage their professional profiles.
* **Portfolio Uploads:** A feature to upload portfolio images, which are handled using Cloudinary.
* **Public URLs:** Artisans get custom public URLs for their profiles.
* **Security:** The system tracks login attempts to secure artisan accounts.

**For Users:**

* **Authentication:** Users can register and log in using their email.
* **Discovery:** Users can browse and search for artisans.
* **Viewing:** Users have access to view artisan profiles and their portfolios.

**Security & Performance Features:**

* **Security Headers:** Uses Helmet.js to set secure HTTP headers.
* **Authentication:** Implements JWT (JSON Web Tokens) with HTTP-only cookies for authentication.
* **Input Validation:** Uses Zod for validating user inputs.
* **Rate Limiting:** In-built rate limiting to prevent abuse.
* **Environment Validation:** Ensures all necessary environment variables are present and valid.

**Technology Stack**

The project uses a modern and extensive set of technologies:

| **Component** | **Technology** |
| --- | --- |
| **Frontend** | React 19, React Router v7, Tailwind CSS, Axios, Vite, Algolia InstantSearch, Google Maps API. |
| **Backend** | Node.js (v18-22), Express.js, MongoDB with Mongoose, JWT, Zod, Cloudinary, Resend/Nodemailer. |
| **Real-time** | Stream Chat (for chat), Daily.co (for video calls), OneSignal (for push notifications). |
| **Other Services** | Razorpay (payments), QStash (background jobs), Redis (caching), PostHog, Sentry, LogRocket (analytics). |

**Documentation and Project Structure**

The project is well-documented within the docs/ directory, providing guides for setup, integrations, API reference, and CI/CD.

The project is structured into two main folders:

* kalasetu-backend/: Contains the Express.js application, including controllers, models, routes, and middleware.
* kalasetu-frontend/: Contains the React application, structured with components, pages, context, and hooks.

**Deployment and Testing**

The README.md provides deployment guides for both the backend (on Render.com) and the frontend (on Vercel).

For testing and demonstration purposes, the project includes:

* **Demo Accounts:** Ready-made accounts for an artisan, a user, and an admin are provided for quick testing.
* **Database Scripts:** It includes scripts to seed the database with demo artisans and users (npm run seed:all) and to clean up test data (npm run cleanup)

**1.1 Existing systems and need for system :**

**1.Existing System :**

Based on your project's README.md file, here is an explanation for "Existing systems and need for system":

The README.md describes KalaSetu as a "full-stack web application connecting traditional artisans with USERs". This description implies that the primary *need* for this system stems from the lack of a specialized, dedicated platform where traditional artisans can effectively showcase their work and connect with potential customers.

While the document doesn't explicitly name existing systems (such as generic e-commerce giants or social media platforms), the features of KalaSetu are designed to address the gaps left by those systems. The need is for a tailored solution that provides artisans with tools to build a professional online identity, which KalaSetu addresses through features like:

* **Professional Profile Management:** Artisans can create and manage their own professional profiles.
* **Portfolio Uploads:** The system allows artisans to upload portfolio images (managed via Cloudinary) to showcase their work.
* **Discoverability:** KalaSetu provides custom public profile URLs for artisans and allows users to browse and search specifically for artisans, making them more discoverable than on a general-purpose platform.
* **Security:** The system focuses on secure authentication (including email, phone, and OTP) and account security with login attempt tracking, which may be more robustly tailored to this specific user base than generic solutions.

**1.2 Scope of the System :**

The scope of **KalaSetu** is to function as a comprehensive, full-stack marketplace platform. Its primary boundary is defined by its two main user groups: **traditional artisans** and the **end-users** who want to connect with them.

The system's scope covers the complete interaction lifecycle, from discovery and communication to payment and security.

**1. Functional Scope: Artisan Features**

The scope for artisans is focused on providing them with a complete digital storefront and professional identity.

* **Authentication:** Secure registration and login using multiple methods, including email, phone, and Firebase OTP.
* **Profile Management:** Artisans can create, manage, and customize their professional profiles.
* **Portfolio Showcase:** A core feature is the ability to upload portfolio images (handled by Cloudinary) to display their work.
* **Discoverability:** The system provides custom public URLs for artisan profiles to make them easily shareable and accessible.
* **Security:** The scope includes account security features like login attempt tracking.

**2. Functional Scope: User Features**

The scope for users is centered on discovery, browsing, and interaction.

* **Authentication:** Users have a separate, email-based authentication system.
* **Discovery:** Users can browse listings and use an advanced search (powered by Algolia InstantSearch) to find specific artisans.
* **Interaction:** Users can view artisan profiles and their portfolios.

**3. Technical & Interaction Scope**

The project's technical scope is extensive, incorporating a wide range of modern services to create a rich, interactive platform.

* **Communication:** The system is designed for direct artisan-user communication via **real-time chat** (Stream Chat) and **video calls** (Daily.co).
* **Transactions:** The scope includes financial transactions, managed through **Razorpay** for payments.
* **Notifications:** It includes user engagement via **push notifications** (OneSignal) and **email** (Resend/Nodemailer).
* **System Integrity:** The platform's scope includes robust security measures (Helmet, JWT, HTTP-only cookies), input validation (Zod), and performance features (rate limiting, Redis caching).
* **Analytics & Jobs:** The system is built to be monitored (PostHog, Sentry, LogRocket) and to handle asynchronous tasks (QStash for background jobs).

**4. Boundaries & Out of Scope**

The README.md also clarifies the project's boundaries:

* **Future Features:** **Reviews and ratings** are explicitly listed as a "Future" item, placing them outside the current scope of the application.
* **Logistics:** The documentation does not mention features for inventory management, shipping, or physical product fulfillment. This suggests the scope is focused on **facilitating connections, bookings, and services** rather than acting as a traditional e-commerce platform for physical goods.

**1.3 Operating Environment Hardware and Software :**

**Software Environment**

The README.md specifies the following software prerequisites and technologies required to run the KalaSetu project:

* Runtime: Node.js (v18 or higher, with v20+ recommended).
* Database: MongoDB (either a local instance or a cloud-hosted service like MongoDB Atlas).
* Version Control: Git.
* Frontend Stack: The frontend is built with React 19 and Vite, using React Router v7 for routing and Tailwind CSS for styling.
* Backend Stack: The backend runs on Node.js (v18-22) and the Express.js framework.
* Key Services & Integrations: The system also relies on several external services that are part of its operating environment, including:
  + Algolia: For search functionality.
  + Cloudinary: For portfolio image uploads.
  + Stream Chat & Daily.co: For real-time chat and video calls.
  + Razorpay: For processing payments.
  + QStash: For managing background jobs.
  + Firebase, Resend/Nodemailer, OneSignal: For authentication, email, and push notifications.

**Hardware Environment**

The README.md file does not specify explicit hardware requirements (such as CPU, RAM, or disk space).

* For Development: The environment is a standard development machine capable of running Node.js, MongoDB (or connecting to a remote instance), and a modern web browser.
* For Production: The project is designed for a cloud-based operating environment. The deployment guides explicitly mention Render.com for the backend and Vercel for the frontend. This indicates the hardware is managed by these Platform-as-a-Service (PaaS) providers**.**

**1.4 Brief Description of Technology Used :**

**1. Frontend Technologies**

**React.js**  
React is a JavaScript library developed by Meta for building dynamic, component-based user interfaces. It enables efficient rendering through its virtual DOM mechanism and supports reusable UI components, which reduces development complexity. In KalaSetu, React is used to create interactive pages such as artisan profiles, dashboards, and search interfaces. Its declarative nature improves maintainability and enhances the overall user experience.

**Vite**  
Vite is a next-generation frontend build tool that offers fast hot module replacement and optimized production builds. It replaces older bundlers like Webpack, significantly reducing build and refresh times during development. KalaSetu uses Vite to ensure faster development and smooth integration with modern JavaScript libraries.

**Tailwind CSS**  
Tailwind is a utility-first CSS framework that allows rapid UI development using predefined classes. It ensures consistent design without writing extensive custom CSS. In this project, Tailwind provides a clean, responsive, and modern interface for both user and artisan dashboards, ensuring adaptability across devices.

**Axios**  
Axios is used for asynchronous HTTP communication between the frontend and backend. It simplifies data transfer for actions like user login, artisan registration, and portfolio uploads. Its promise-based API provides efficient error handling and request management.

**2. Backend Technologies**

**Node.js**  
Node.js is a server-side JavaScript runtime built on Chrome’s V8 engine. It allows asynchronous, event-driven operations, making it ideal for scalable web applications. In KalaSetu, Node.js powers the backend logic, handling authentication, API requests, and server-side data processing.

**Express.js**  
Express is a lightweight and flexible web application framework for Node.js. It provides routing, middleware management, and simplified API creation. KalaSetu uses Express to build RESTful APIs that connect the frontend with the database, enabling operations like user registration, artisan search, and bookings.

**MongoDB**  
MongoDB is a NoSQL document-oriented database that stores data in flexible JSON-like documents. It allows dynamic schemas, which are suitable for evolving project requirements. KalaSetu uses MongoDB to manage user and artisan data, portfolios, chat messages, and session tokens efficiently.

**Mongoose**  
Mongoose acts as an Object Data Modeling (ODM) library for MongoDB. It enforces schema validation and provides an easy interface for database queries. It ensures data integrity and simplifies database operations like CRUD actions and relationship handling.

**3. Authentication and Security**

**Firebase Authentication / OTP Verification**

Firebase Authentication simplifies user verification through secure phone-based OTPs and email logins. KalaSetu integrates Firebase to enable both users and artisans to log in safely without complex password management.

**JWT (JSON Web Tokens)**

JWTs are used for stateless authentication between client and server. Tokens are stored in HTTP-only cookies for enhanced security. This approach prevents cross-site scripting (XSS) attacks and ensures authenticated sessions remain tamper-proof.

**4. Cloud and Media Management**

**Cloudinary**  
Cloudinary is a cloud-based media management service that handles image and video uploads, storage, and optimization. KalaSetu uses Cloudinary for artisans to upload portfolio images securely and efficiently, ensuring responsive image delivery across devices.

**5. Additional Integrations (Optional/Future Scope)**

* **Socket.io** – For real-time chat and notifications between artisans and users.
* **WebRTC / PeerJS** – Enables direct video calls for personalized client–artisan interaction.
* **Google Maps API** – Helps users locate nearby artisans using geolocation features.
* **Zod Validation** – Provides runtime data validation for API requests to maintain consistency and prevent invalid data entries.
* **Render / Vercel** – Used for backend and frontend deployment respectively, ensuring continuous integration and easy scalability

**2.Proposed System :**

The Before You Go system is designed as an intelligent, personalized travel-planning platform that overcomes the limitations of existing travel recommendation tools. The proposed system focuses on delivering tailored travel experiences by analyzing user preferences, personality traits, and travel habits. It provides an end-to-end solution—from discovering destinations to supporting travelers throughout their journey.

The proposed system introduces a personality-based travel profiling model known as Travel DNA, which acts as the core of all recommendations. Based on a short quiz, the system identifies the user’s travel style—such as explorer, comfort-seeker, nature lover, or adventure enthusiast—and provides curated travel suggestions accordingly. This personalized approach makes the planning experience unique and user-centric.

In addition, the system integrates real-time itinerary adjustments, group travel matching, cloud-based data management, and an AI-powered Travel Buddy for assistance. All features are accessible within a single, responsive web platform, eliminating the need for multiple apps or manual planning.

**2.1 Feasibility study**

Before developing the Before You Go travel planning system, a detailed feasibility study was conducted to evaluate the practicality and viability of the project. The study focuses on three major aspects: technical feasibility, operational feasibility, and economic feasibility. These factors help determine whether the system can be successfully developed, implemented, and maintained.

**1. Technical Feasibility**

Technical feasibility examines whether the required technology and tools are available, reliable, and capable of supporting the system’s functionalities.

Findings:

* The project uses widely adopted technologies such as React.js, JavaScript, HTML5, CSS3, and Firebase, all of which are well-supported and easy to integrate.
* Firebase provides built-in features like Authentication, Firestore Database, and Hosting, reducing the need for complex backend servers.
* The application can be deployed using Vercel, which simplifies CI/CD and ensures fast global performance.
* All development tools (VS Code, npm, GitHub) are easily accessible and require no special hardware.

**2. Operational Feasibility**

Operational feasibility checks whether the system will function effectively for users and whether the workflow aligns with real-world needs.

Findings:

* The system provides a simple, quiz-based interface to generate a personalized Travel DNA, which is easy for users of all ages.
* Saving profiles, accessing itineraries, and interacting with the AI Travel Buddy enhances user convenience.
* Group personality merging supports real-world scenarios like family trips, college tours, or office outings.
* Cloud-based access allows users to use the system anytime, from any device, without installation.

**3. Economic Feasibility**

Economic feasibility analyzes the cost-effectiveness of the system, ensuring that benefits outweigh development and maintenance costs.

Findings:

* The system uses free or low-cost services such as Firebase (free tier), GitHub, Node.js, and Vercel (free deployment).
* No physical hardware, servers, or paid tools are required for development.
* Maintenance costs are minimal due to the serverless architecture.
* The system has potential for future monetization (ads, premium AI features, affiliate partnerships like hotel/flight booking).

**2.2 Objective of purposed system**

1. **Digital Empowerment of Artisans:**  
   To provide artisans—especially those from rural or semi-urban regions—a digital space to display their portfolios and reach a broader audience without intermediaries.
2. **Direct Customer Interaction:**  
   To allow customers to discover artisans, view their work, and contact them directly for orders or commissions through chat or video call facilities.
3. **Streamlined Management:**  
   To simplify profile creation, order management, and communication through a single integrated dashboard for both artisans and users.
4. **Secure and Scalable Architecture:**  
   To ensure reliable, scalable performance with secure authentication, encrypted communications, and efficient data management via cloud services.
5. **Future Integrations:**  
   To support future extensions such as AI-based artisan recommendations, analytics dashboards for artisans, and integrated payment gateways.

**2.3 Users of System**

The KalaSetu platform caters to multiple user categories, each interacting with the system differently:

1. Artisans:  
Individuals or groups creating handcrafted items who can register, manage their profile, and display their portfolio to attract customers.

2. Customers/Users:  
Buyers who can browse artisan profiles, view product details, communicate directly, and place custom or direct orders.

3. System Administrator:  
Responsible for maintaining user records, verifying artisans, monitoring system activity, and ensuring smooth operation.

4. Future Integrations (Optional):  
Institutions, NGOs, or government bodies promoting traditional arts may use the platform to organize virtual exhibitions or artisan events.

**3.Analysis and design**

The Analysis and Design phase focuses on understanding system requirements, user expectations, and functional behavior, followed by creating a logical model of the system architecture. This phase defines how the KalaSetu platform operates, what modules it contains, and how data flows between the frontend, backend, and database.

The purpose of this section is to ensure that the system’s design supports scalability, maintainability, and usability while meeting all project objectives.

**3.1 System Requirements**

System requirements describe what the system should achieve (Functional Requirements) and how well it should perform (Non-Functional Requirements).

**Functional Requirements**

1. **User Management**
   * The system must allow both artisans and customers to register and log in securely.
   * Firebase OTP or email-based authentication must be used for verification.
   * Role-based access must differentiate artisan and customer dashboards.
2. **Artisan Profile Management**
   * Artisans can create and update profiles with personal details, skills, and portfolio images.
   * Uploaded media should be securely stored in Cloudinary and linked to artisan accounts.
3. **User Search and Discovery**
   * Customers should be able to browse or search artisans by name, craft type, or location.
   * Search results should display artisan profiles, ratings (future enhancement), and availability.
4. **Portfolio and Media Uploads**
   * The system must allow artisans to upload product images, descriptions, and contact information.
   * Each upload must be validated and optimized for faster display.
5. **Communication and Interaction**
   * Users and artisans must be able to send messages through an integrated chat module (Socket.io).
   * Optional future feature: video calls via WebRTC for real-time communication.
6. **Authentication and Security**
   * JWT tokens and HTTP-only cookies must be used for session security.
   * Sensitive data like passwords or tokens should never be exposed in frontend code.
7. **Database Management**
   * MongoDB must store all data related to users, artisans, portfolios, and chat messages.
   * The database should support CRUD operations efficiently**.**
8. **Responsive Web Design**
   * The user interface must adapt to desktop, tablet, and mobile devices without losing functionality.
9. **Deployment and Hosting**
   * The frontend should be deployed on Vercel, backend on Render, and media on Cloudinary.

**Non-Functional Requirements**

1. **Performance**
   * All pages should load within 2–3 seconds under normal network conditions.
   * API response time must remain under 1 second for standard requests.
2. **Security**
   * Data transmission must be encrypted using HTTPS.
   * Authentication must use secure JWT and Firebase verification.
3. **Usability**
   * The interface should be intuitive, consistent, and visually appealing using Tailwind CSS.
4. **Reliability**
   * System uptime should be at least 99% during deployment.
   * The database should auto-reconnect if a connection failure occurs.
5. **Scalability**
   * The architecture should support adding new modules like analytics, reviews, and payments.
6. **Maintainability**
   * The system codebase should follow modular structure (MERN stack separation) for easy updates.
7. **Compatibility**
   * Should function across modern browsers (Chrome, Firefox, Edge) and OS platforms.
8. **Portability**
   * Should run on any internet-enabled device without installation.

**3.2 Module Hierarchy Diagram**

**A diagram of a system

AI-generated content may be incorrect.**

**3.3 Sample Input and output screens**

**1.User sample input output:**

**Login page User:**

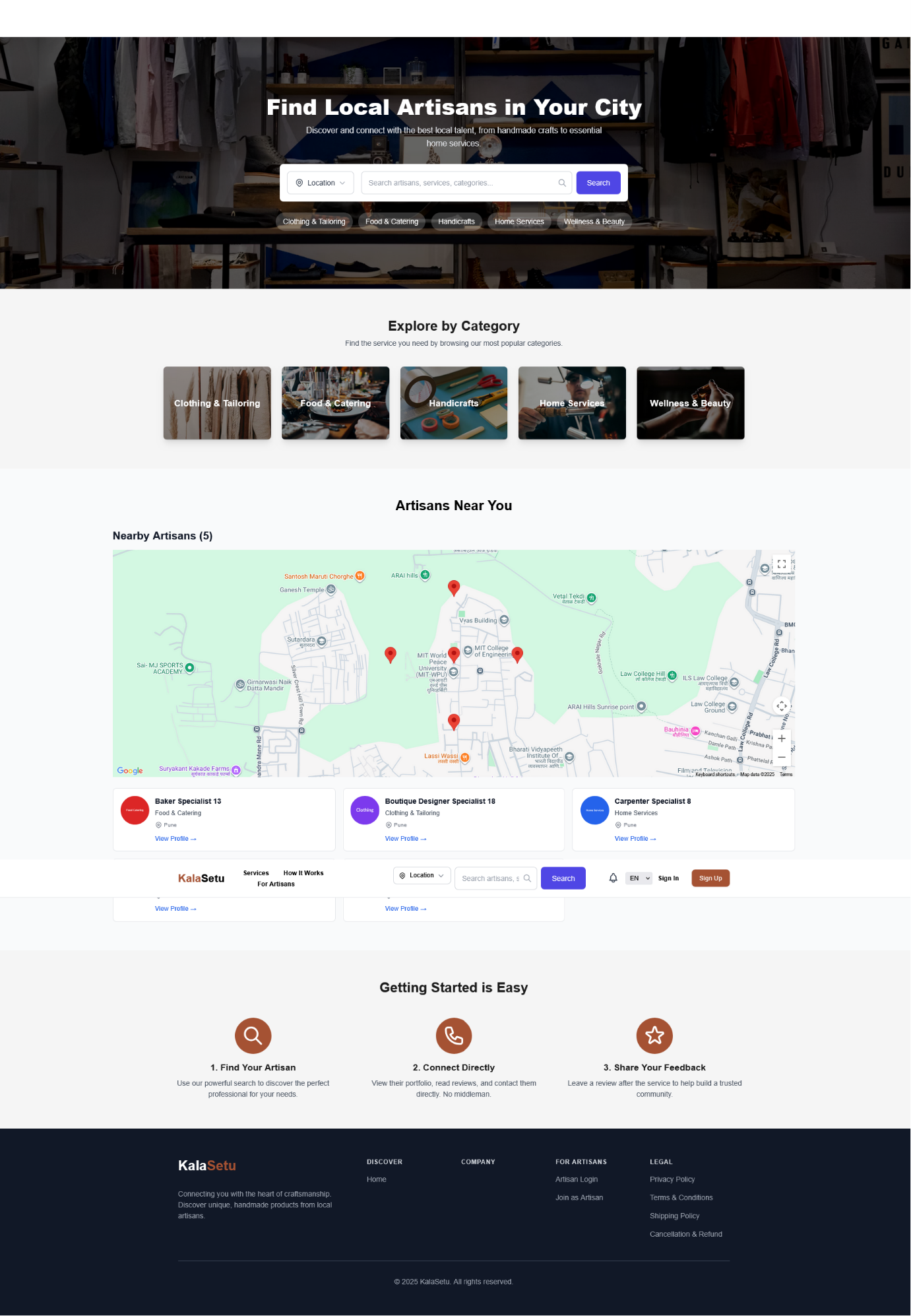
**A screenshot of a login page

AI-generated content may be incorrect.**

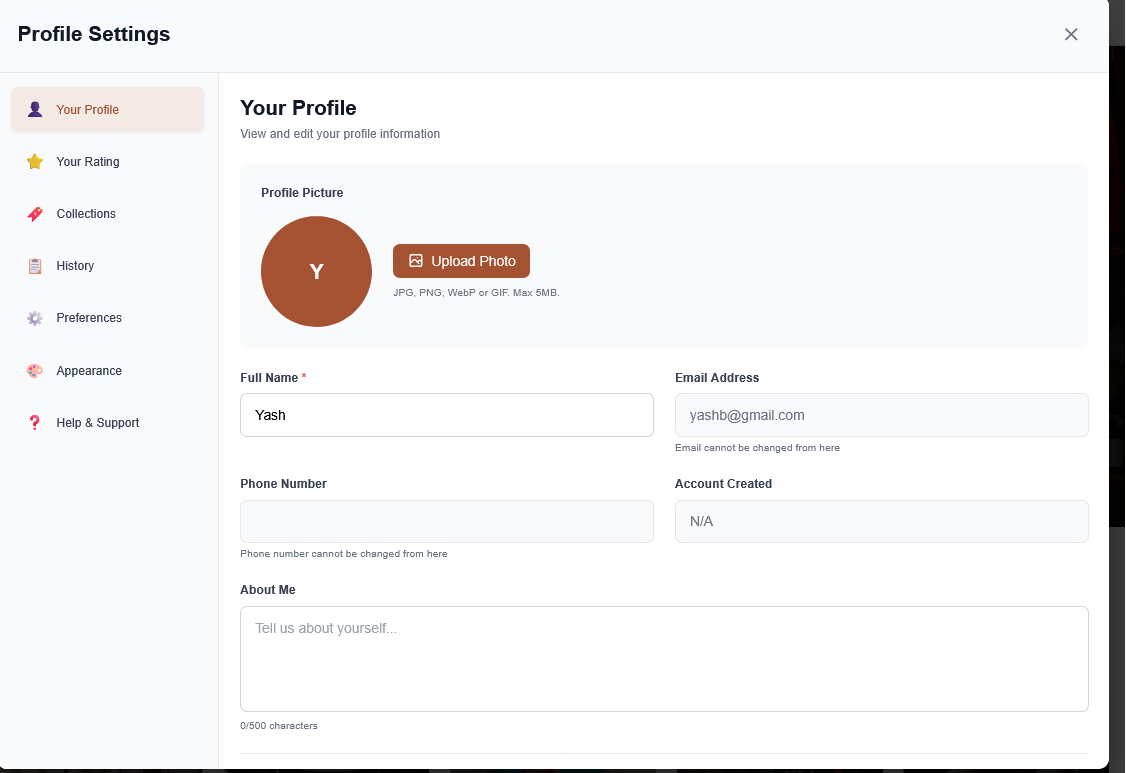
**User Login**

* Page: KalaSetu Sign-In Page
* Steps:
  1. Open the KalaSetu main website.
  2. Enter your email or phone number.
  3. Enter your password.
  4. Click Sign In to access your account.
  5. If you don’t have an account, click “Create an account” to register.
  6. Use “Forgot Password?” if you need to reset your password.

**User Interference (Dashboard)**

****

**User profile page**

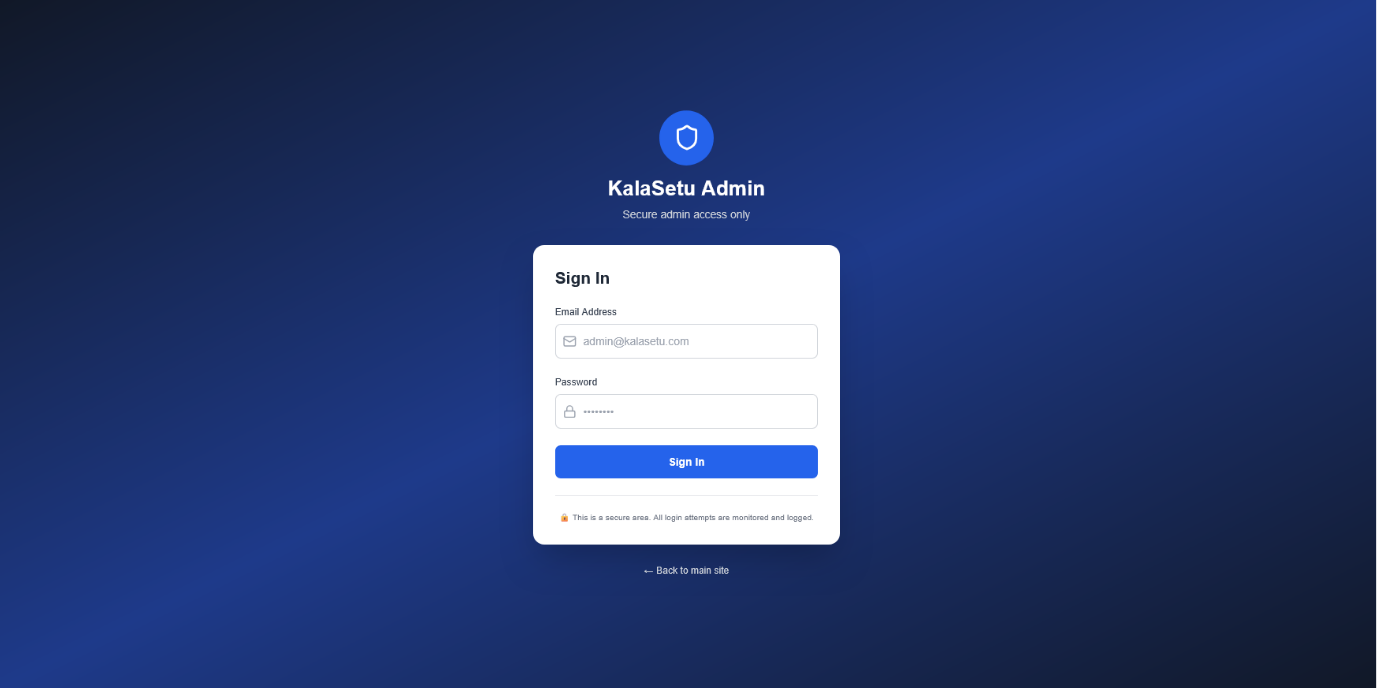


**Profile Management**

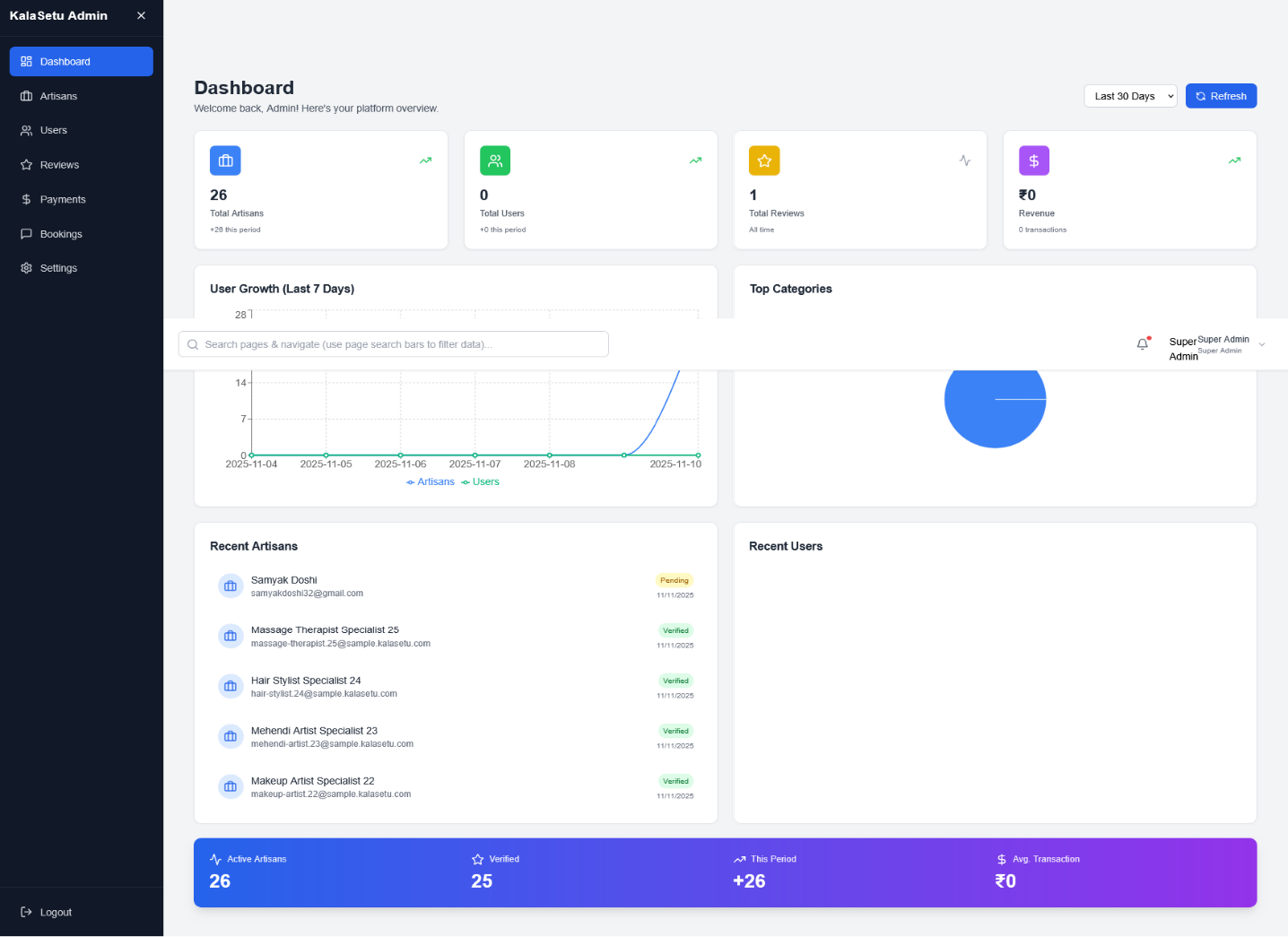
* Page: Profile Settings
* Access: After signing in, go to your profile icon → Settings.
* Features:
  + Upload Profile Picture: Supports JPG, PNG, WebP, GIF (max 5MB).
  + Edit Personal Info:
    - Update Full Name and About Me section.
    - View your Email Address and Account Creation Date (non-editable).
    - Phone Number cannot be changed here.
  + Navigation Panel:
    - Your Rating: View your user/artisan rating.
    - Collections: Manage your saved artisans or services.
    - History: View your interaction or booking history.

**2.Admin Interference**

**Admin sign in page :**

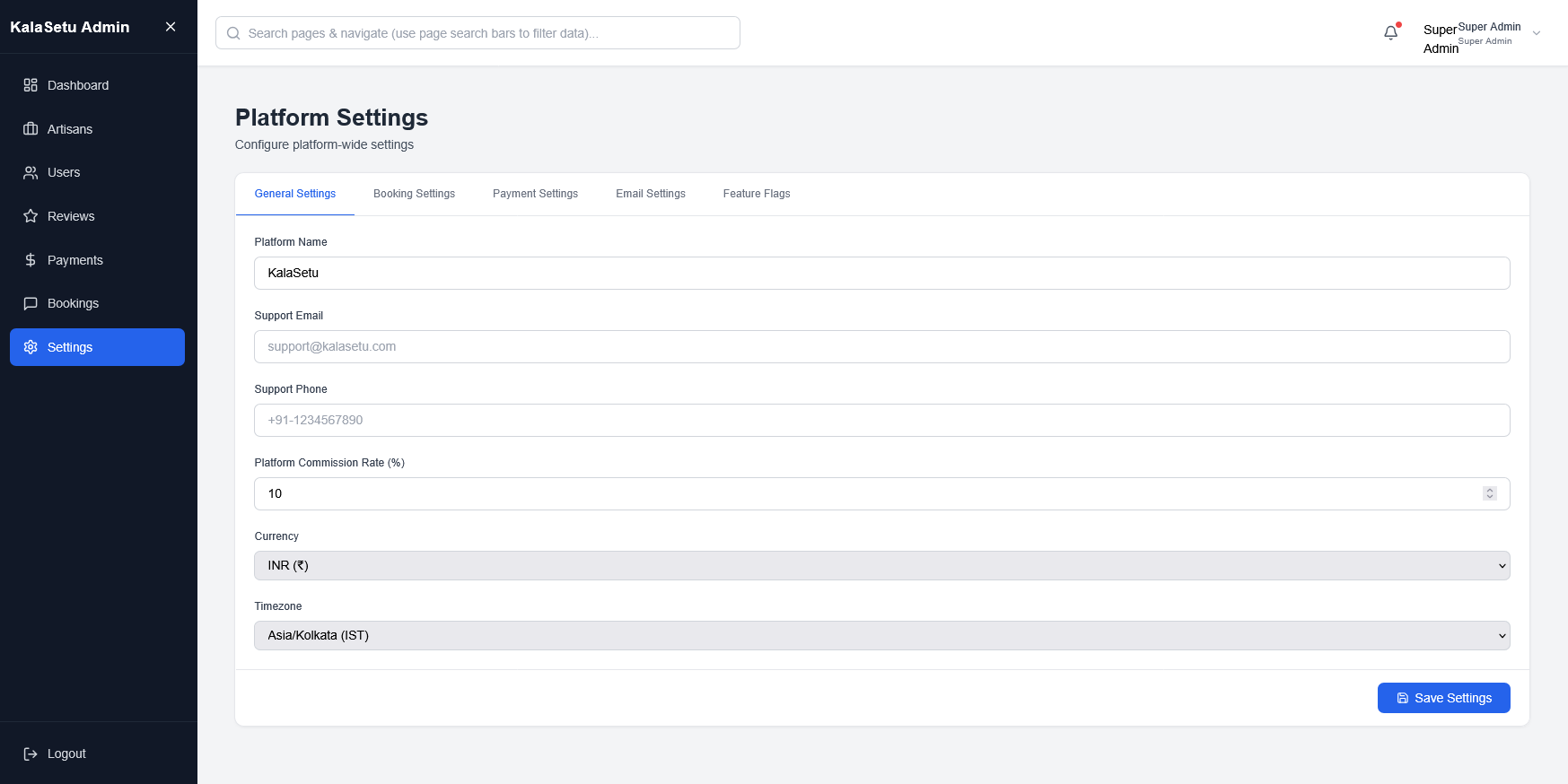
****

This is the main homepage screen for the Kalasetu project. It shows the "Kalasetu" logo and a search bar where users can find artisans and services.

**Admin Dashboard:**

This image displays the "How It Works" section, explaining the three-step process for users: 1. Discover, 2. Connect, and 3. Hire.

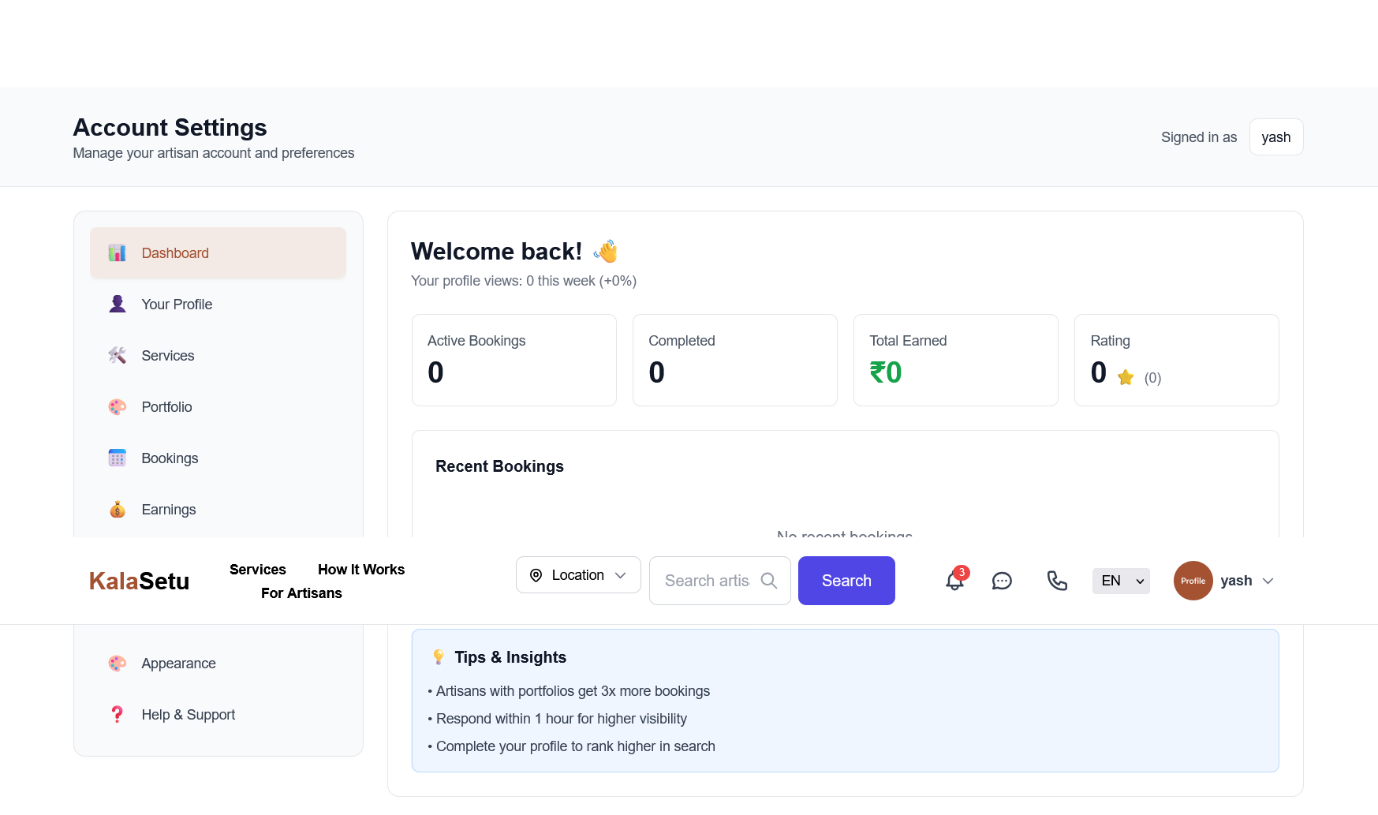
**Admin Setting:**

****

This shows the "Nearby Artisans" map feature from the homepage. It displays a map with pins indicating the locations of artisans near the user.

**3.Artisan Interference**

**Dashboard:**

****

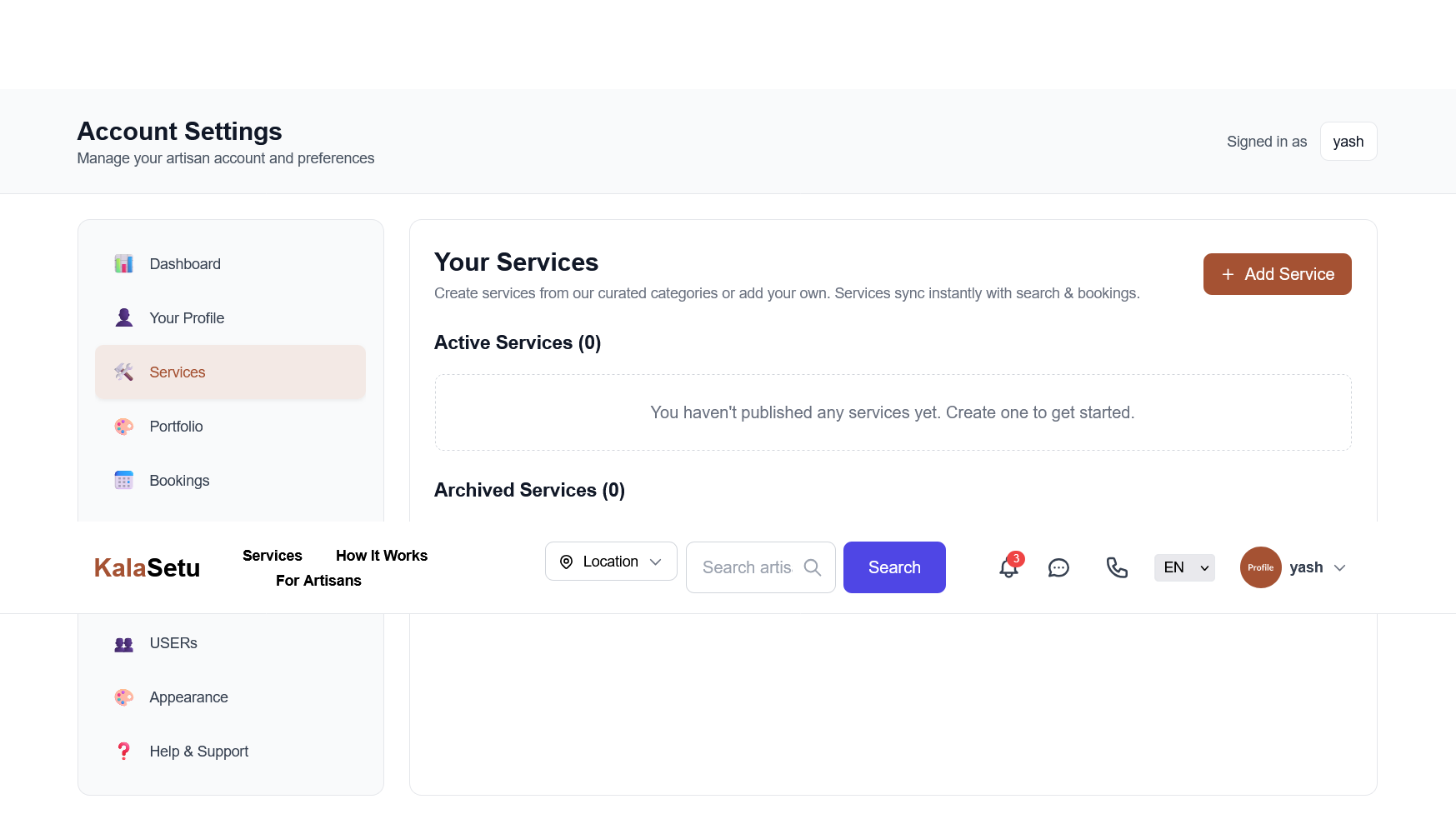
1. **Find an Artisan Start on the homepage.** You can find artisans by:

* Typing what you're looking for (like "Pottery" or "Plumber") into the main search bar.
* Browsing the "Nearby Artisans" map to see who is close to you.

**2. Learn "How It Works"** The process is three simple steps:

* **Discover**: Find the perfect artisan for your needs.
* **Connect**: Contact the artisan, chat with them, or have a video call.
* **Hire**: Book their services directly through the platform.

**Service registration :**

****

3. View the Artisan's Profile When you select an artisan, you can see their full profile. Here you can:

* See their "Services," "Portfolio," and "Reviews".
* Choose to "Book Now" or "Contact" the artisan.

4. Manage Your Account After you log in, you can click on your profile to manage your account. The dashboard lets you:

* View your "Order History".
* See your saved "Bookmarks".
* Change your "Preferences".

**3.4 Database Design**

The database design of the KalaSetu Platform follows a document-oriented approach using MongoDB. This allows flexible data storage and scalability for future enhancements. The database contains multiple collections that store data related to users, artisans, portfolios, and communication records.

Each collection is designed to maintain relational consistency through unique identifiers (ObjectIDs) while ensuring fast access, minimal redundancy, and optimized query performance.

**1. Collections Overview**

| Collection Name | Description |
| --- | --- |
| users | Stores login and basic account details of both artisans and customers. |
| artisans | Contains artisan-specific information such as craft type, bio, and location. |
| portfolios | Stores artisan product images and descriptions, linked to artisan accounts. |
| messages | Stores real-time chat messages exchanged between artisans and users. |
| admins | Holds admin login credentials and moderation activity logs. |

**2. Collection Structures**

**(a) users**

| Field | Type | Description |
| --- | --- | --- |
| \_id | ObjectId | Unique identifier for each user. |
| name | String | Full name of the user. |
| email | String | Registered email address. |
| passwordHash | String | Encrypted password using bcrypt. |
| role | String | Specifies user type – “artisan” or “customer”. |
| createdAt | Timestamp | Date and time of registration. |

**(b) artisans**

| Field | Type | Description |
| --- | --- | --- |
| \_id | ObjectId | Unique artisan ID. |
| userId | ObjectId | References the corresponding user entry. |
| craftType | String | Type of craft or product (e.g., pottery, textiles). |
| description | String | Short bio or about section. |
| portfolio | Array | List of uploaded product URLs. |
| location | String | Artisan’s city or area. |

**(c) portfolios**

| Field | Type | Description |
| --- | --- | --- |
| \_id | ObjectId | Unique portfolio ID. |
| artisanId | ObjectId | Linked to the artisan. |
| imageUrl | String | URL of uploaded image (Cloudinary). |
| caption | String | Description or product name. |
| createdAt | Timestamp | Upload date. |

**(d) messages**

| Field | Type | Description |
| --- | --- | --- |
| \_id | ObjectId | Unique message ID. |
| senderId | ObjectId | User who sent the message. |
| receiverId | ObjectId | Recipient of the message. |
| content | String | Text of the message. |
| timestamp | Timestamp | Message time. |

**(e) admins**

| Field | Type | Description |
| --- | --- | --- |
| \_id | ObjectId | Unique admin ID. |
| email | String | Admin email ID. |
| actions | Array | Records of verification or moderation activities. |

**3. Relationships Between Collections**

* Each artisan references a single entry in the users collection (One-to-One relationship).
* Each artisan can have multiple portfolios (One-to-Many relationship).
* Each user can exchange multiple messages with artisans (Many-to-Many relationship).
* Admins oversee artisans and have permissions to verify or remove them from the system.

**4. Advantages of the Database Design**

* Scalable: Can handle large datasets with minimal performance degradation.
* Flexible: Allows easy addition of new fields and collections.
* Efficient Queries: Optimized for search and filtering using MongoDB indexes.
* Secure: Sensitive user data is encrypted before storage

**Data base using JSON**

| **Field Name** | **Data Type** | **Required** | **Description** |
| --- | --- | --- | --- |
| fullName | String | ✓ | Full name of the user |
| email | String | - | Email address (unique, sparse) |
| phoneNumber | String | - | Contact phone number (unique, sparse) |
| password | String | ✓ | Encrypted password |
| lockUntil | Date | - | Timestamp until account is locked |
| loginAttempts | Number | - | Count of failed login attempts (defaults to 0) |
| resetPasswordToken | String | - | Token for password reset |
| resetPasswordExpires | Date | - | Expiry date for password reset token |
| emailVerificationToken | String | - | Token for email verification |
| emailVerificationExpires | Date | - | Expiry date for email verification token |
| emailVerified | Boolean | - | Flag if email is verified (defaults to false) |
| otpCode | String | - | One-time password for phone verification |
| otpExpires | Date | - | Expiry date for OTP |
| otpAttempts | Number | - | Count of failed OTP attempts (defaults to 0) |
| bookmarks | [ObjectId] | - | Array of references to bookmarked Artisans |

**4.Coding**

**4.1 Code Snippets**

**1. Backend – Server Setup (Node.js + Express.js)**

// server.js

import express from "express";

import mongoose from "mongoose";

import cors from "cors";

import dotenv from "dotenv";

dotenv.config();

const app = express();

app.use(cors());

app.use(express.json());

mongoose.connect(process.env.MONGO\_URI)

  .then(() => console.log("MongoDB Connected"))

  .catch((err) => console.error(err));

app.get("/", (req, res) => {

  res.send("KalaSetu Server Running...");

});

const PORT = process.env.PORT || 5000;

app.listen(PORT, () => console.log(`Server running on port ${PORT}`));

**2.** **User Model – MongoDB Schema**

// models/User.js

import mongoose from "mongoose";

const userSchema = new mongoose.Schema({

  name: { type: String, required: true },

  email: { type: String, required: true, unique: true },

  passwordHash: { type: String, required: true },

  role: { type: String, enum: ["artisan", "customer"], required: true },

  createdAt: { type: Date, default: Date.now },

});

export default mongoose.model("User", userSchema);

**(c) Authentication Route (JWT + Bcrypt)**

// routes/auth.js

import express from "express";

import bcrypt from "bcryptjs";

import jwt from "jsonwebtoken";

import User from "../models/User.js";

const router = express.Router();

router.post("/login", async (req, res) => {

  const { email, password } = req.body;

  const user = await User.findOne({ email });

  if (!user) return res.status(404).json({ msg: "User not found" });

  const isMatch = await bcrypt.compare(password, user.passwordHash);

  if (!isMatch) return res.status(401).json({ msg: "Invalid credentials" });

  const token = jwt.sign({ id: user.\_id, role: user.role }, process.env.JWT\_SECRET, { expiresIn: "1h" });

  res.json({ token, role: user.role });

});

export default router;

(d)Login page

import { useState } from 'react';

import { Link, useNavigate } from 'react-router-dom';

import { useAuth } from '../context/AuthContext';

const LoginPage = () => {

    const [loginIdentifier, setLoginIdentifier] = useState('');

    const [password, setPassword] = useState('');

    const [error, setError] = useState(null);

    const [loading, setLoading] = useState(false);

    const [showPassword, setShowPassword] = useState(false);

    // Use artisanLogin for artisan portal authentication

    const { artisanLogin } = useAuth();

    const navigate = useNavigate();

    const handleSubmit = async (e) => {

        e.preventDefault();

        setLoading(true);

        setError(null);

        try {

            const user = await artisanLogin({ loginIdentifier, password });

            // After successful login, take artisans to their account page

            navigate('/artisan/dashboard/account');

        } catch (err) {

            setError(err.response?.data?.message || err.message || 'An unexpected error occurred.');

        } finally {

            setLoading(false);

        }

    };

    return (

        <div className="min-h-screen bg-gradient-to-br from-gray-50 to-gray-100 flex items-center justify-center py-12 px-4 sm:px-6 lg:px-8">

            <div className="max-w-md w-full space-y-8">

                {/\* Header \*/}

                <div className="text-center">

                    <Link to="/" className="text-3xl font-bold text-[#A55233]">Kala<span className="text-gray-800">Setu</span></Link>

                    <h2 className="mt-6 text-3xl font-extrabold text-gray-900">

                        Welcome back

                    </h2>

                    <p className="mt-2 text-sm text-gray-600">

                        Sign in to your artisan account

                    </p>

                </div>

**5.Limitations of Proposed System**

Although the **KalaSetu – Artisan Marketplace Platform** provides a strong digital foundation for connecting artisans and customers, it still has certain limitations that can be improved in future versions. These limitations arise from technical, resource, and implementation constraints during the current development phase.

1. **Limited Payment Integration**

The present system does not include a built-in online payment gateway. Transactions between artisans and users are conducted externally, which may affect convenience and trust. Future versions should integrate secure payment APIs such as Razorpay or Stripe.

1. **Basic Rating and Review System**

Currently, users cannot provide reviews or ratings for artisans. Feedback mechanisms are essential for building credibility and helping customers make informed decisions.

1. **No AI-Based Recommendations Yet**

Although the architecture supports future AI integration, personalized artisan recommendations and product suggestions are not yet implemented in this version.

1. **Limited Portfolio Analytics**

Artisans cannot currently track engagement metrics such as profile views, portfolio performance, or inquiry statistics. Analytical dashboards can enhance their understanding of user interest.

1. **Chat System Without Media Sharing**

The existing chat module supports only text messages. It does not currently allow sharing of images, documents, or voice notes, which can limit communication efficiency between users and artisans.

1. **Dependence on Stable Internet Connection**

Since KalaSetu is a cloud-based platform, a continuous internet connection is necessary. Users in low-connectivity areas may face difficulties accessing the system.

1. **Manual Artisan Verification**

The artisan verification process is partially manual, requiring admin involvement. Automated verification through government databases or ID APIs could streamline this process.

1. **No Offline Access or Progressive Web App (PWA)**

The application currently functions only in an online environment. Implementing offline caching or a PWA version could allow basic functionality without active internet access.

1. **Limited Localization Support**

The interface supports only English at present. For wider reach, multilingual support (e.g., Hindi, Marathi, Tamil) should be added in future versions.

10**.Basic Admin Dashboard**

The administrator currently manages artisans and users using minimal features.

A more comprehensive dashboard could include analytics, approval workflows, and

reporting tools.

11.**Scalability Constraints in Free Hosting Services**

Since the project uses free-tier services like Render and Vercel, high traffic may cause temporary slowdowns or service interruptions. Upgrading to paid hosting would improve performance stability.

12.**Absence of Mobile Application**

The system is web-only and lacks a dedicated mobile application. A native app would improve accessibility and usability for artisans with limited technical experience.

**6. Proposed Enhancements**

While the current version of the KalaSetu – Artisan Marketplace Platform provides a solid foundation for connecting artisans and customers, several enhancements can be implemented in future releases to make the system more advanced, user-friendly, and feature-rich.  
These proposed improvements focus on scalability, automation, security, and user engagement.

**1. Online Payment Integration**

Integrate secure and reliable payment gateways such as Razorpay, Stripe, or PayPal to enable direct online transactions between artisans and customers.  
This will reduce dependency on third-party communication and increase trust in the platform.

**2. Advanced Rating and Review System**

Add a review module where customers can rate artisans and leave feedback.  
This feature will build authenticity, help artisans improve, and guide new customers toward quality creators.

**3. AI-Powered Recommendation Engine**

Implement an AI recommendation system that suggests artisans or products based on user behavior, location, and past searches.Machine learning models can enhance personalization, improving customer engagement and satisfaction.

**4. Portfolio Analytics Dashboard**

Develop an analytics dashboard for artisans to monitor profile visits, product engagement, and customer inquiries.Visual charts and metrics will help artisans understand trends and optimize their offerings**.**

**5. Multimedia Chat Feature**

Upgrade the existing chat system to support image, file, and voice note sharing.  
This will allow artisans to share product samples, and users to send references or custom order details directly within the chat interface.

**6. Mobile Application (Android & iOS)**

Design native mobile apps or a Progressive Web App (PWA) version of KalaSetu for artisans with limited access to laptops.This will improve accessibility, especially for rural artisans using smartphones.

**7. Automated Artisan Verification**

Integrate government ID verification APIs (like DigiLocker) to automate artisan identity checks.  
This will reduce admin workload and increase trust and authenticity on the platform.

**8. Multilingual Support**

Add multi-language options (e.g., Hindi, Marathi, Tamil, Bengali) to make the platform inclusive and easier for regional artisans and users to navigate.

**9. Smart Notification System**

Introduce a real-time notification system to alert artisans and customers about new messages, orders, or profile updates.Push notifications or email alerts will keep users engaged even when offline.

**10. Integration of E-Commerce Features**

Allow artisans to list products for direct purchase through a cart and checkout system.  
This will expand KalaSetu from a service-based to a product-based marketplace, similar to Etsy or Amazon Handmade.

**11. Social Media and Community Features**

Enable social sharing of artisan profiles and portfolios to platforms like Instagram, Facebook, and LinkedIn.Additionally, a community forum can help artisans exchange tips, ideas, and collaboration opportunities.

**12. Enhanced Admin Dashboard**

Develop a feature-rich Admin Panel with role-based access, detailed analytics, and automated monitoring tools.This will allow smoother management of users, portfolios, and system operations.

**13. AI-Driven Fraud Detection**

Use AI algorithms to identify suspicious activities, fake accounts, or fraudulent transactions.  
This will enhance system security and protect both artisans and customers from scams.

**14. Cloud Scalability and CDN Optimization**

Upgrade hosting infrastructure by using AWS, Google Cloud, or Cloudflare CDN for faster global delivery, lower latency, and improved scalability as the user base grows.

**15. Sustainability and Cultural Promotion Initiatives**

Collaborate with NGOs and government bodies to organize digital exhibitions, artisan fairs, and workshops directly through KalaSetu, thus promoting Indian craftsmanship globally

**7. Conclusion:**

The KalaSetu – Artisan Marketplace Platform successfully demonstrates how technology can empower traditional artisans by providing them with a dedicated digital space to showcase their talent, connect with customers, and manage their business efficiently.By bridging the gap between artisans and the modern marketplace, KalaSetu promotes cultural heritage while contributing to economic inclusivity and sustainable growth.

The project achieves its core objectives — creating a secure, scalable, and user-friendly system for artisans and customers alike. Through features such as artisan registration, portfolio management, user search, and real-time communication, the platform offers a smooth and interactive experience for all users.

Technologies like React.js, Node.js, Express.js, MongoDB, Firebase, and Cloudinary ensure reliable performance, seamless integration, and a strong backend structure. The platform’s modular architecture also enables future enhancements such as online payments, analytics, AI-driven recommendations, and multilingual support.

Although the current version has certain limitations, it lays a strong technical and conceptual foundation for future development.With proposed enhancements like payment integration, mobile applications, and smart verification systems, KalaSetu can evolve into a complete digital ecosystem for artisans — connecting tradition with technology on a global scale.

Ultimately, KalaSetu reflects a vision of empowerment — where local craftsmanship meets digital innovation, helping artisans gain visibility, financial independence, and long-term sustainability in the modern economy.

**8.Bibliography :**

1. **React.js Documentation**  
   <https://react.dev/>  
   Official documentation for React library used for building the front-end components and user interface of KalaSetu.
2. **Node.js Documentation**  
   <https://nodejs.org/en/docs>  
   Used for developing the backend server and handling API requests efficiently.
3. **Express.js Documentation**  
   <https://expressjs.com/>  
   Backend framework for creating RESTful APIs and managing routing, middleware, and HTTP requests.
4. **MongoDB Documentation**  
   <https://www.mongodb.com/docs/>  
   NoSQL database used for storing user, artisan, and portfolio data in a document-oriented structure.
5. **Mongoose ODM**  
   <https://mongoosejs.com/>  
   Library for modeling and validating MongoDB data with schemas in Node.js applications.
6. **Firebase Authentication**  
   <https://firebase.google.com/docs/auth>  
   Used for secure user authentication through OTP and email verification methods.
7. **Cloudinary Documentation**  
   <https://cloudinary.com/documentation>  
   Cloud-based media management service used for storing and optimizing artisan portfolio images.
8. **Socket.io Documentation**  
   https://socket.io/docs/  
   Real-time communication library used for chat and live messaging between users and artisans.
9. **Tailwind CSS Documentation**  
   <https://tailwindcss.com/docs>  
   CSS framework used for creating responsive and modern UI components.
10. **Vercel Documentation**  
    <https://vercel.com/docs>  
    Deployment platform used to host the frontend of the KalaSetu web application.
11. **Render Documentation**  
    <https://render.com/docs>  
    Platform used for hosting the backend APIs and server modules of KalaSetu.
12. **JavaScript Official Reference (MDN Web Docs)**  
    <https://developer.mozilla.org/en-US/docs/Web/JavaScript>  
    Reference guide for JavaScript language features and APIs used in frontend and backend development.