Frontend Project - Clean & Organized Structure

🚀 Overview

This is a clean, frontend-only digital marketplace application built with **Next.js 14**, **TypeScript**, **Tailwind CSS**, and **React**. The project features a comprehensive **Escrow System** for secure transactions between buyers and sellers.

📁 Project Structure

```
frontend/
├── src/
│   ├── app/                        # Next.js App Router pages
│   │   ├── checkout/
│   │   │   ├── [projectId]/        # Dynamic checkout route
│   │   │   └── escrow/             # Escrow checkout page
│   │   ├── transactions/           # Transaction dashboard
│   │   │   ├── [id]/               # Individual transaction
details
│   │   │   └── page.tsx            # Transactions list
│   │   ├── profile/                # User profiles
│   │   ├── projects/               # Project listings
│   │   └── ...                     # Other pages
│   │
│   ├── components/                 # React components
│   │   ├── escrow/                 # Escrow system components
│   │   │   ├── EscrowCheckoutPage.tsx
│   │   │   ├── EscrowPaymentForm.tsx
│   │   │   ├── PaymentMethodSelector.tsx
│   │   │   ├── TransactionCard.tsx
│   │   │   ├── TransactionDashboard.tsx
│   │   │   ├── TransactionFilters.tsx
│   │   │   ├── TransactionStatusBadge.tsx
│   │   │   ├── TransactionTimeline.tsx
│   │   │   ├── ReviewPeriodCountdown.tsx
│   │   │   ├── InstallmentPayment.tsx
│   │   │   └── InstallmentTracker.tsx
│   │   │
│   │   ├── notifications/          # Notification components
│   │   │   ├── NotificationCenter.tsx
│   │   │   ├── TransactionNotificationCard.tsx
│   │   │   └── TransactionNotificationCenter.tsx
│   │   │
│   │   ├── auth/                   # Authentication components
(reserved)
```

```
│   │   ├── ui/                       # General UI components
(reserved)
│   │   ├── Navbar.tsx                # Main navigation
│   │   ├── ProjectCard.tsx           # Project display card
│   │   └── PurchaseRequestModal.tsx  # Purchase request modal
│   │
│   ├── contexts/                     # React Context providers
│   │   └── NotificationContext.tsx   # Global notification state
│   │
│   ├── types/                        # TypeScript type definitions
│   │   └── index.ts                  # Consolidated type
definitions
│   │
│   ├── utils/                        # Utility functions
│   │   ├── helpers.ts                # General helper functions
│   │   └── notificationTemplates.ts  # Notification message
templates
│   │
│   ├── constants/                    # Application constants
│   │   └── index.ts                  # App configuration and
constants
│   │
│   └── data/                         # Mock data for development
│       └── projects.ts               # Sample project data
│
├── public/                           # Static assets
├── package.json                      # Dependencies and scripts
├── tailwind.config.js               # Tailwind CSS configuration
├── tsconfig.json                    # TypeScript configuration
└── next.config.js                   # Next.js configuration
```

🎯 Key Features

✅ Escrow System

- **Secure Payment Flow**: Buyer payments are held in escrow until delivery confirmation
- **Multi-Payment Methods**: Credit/Debit cards, PayPal, Bank Transfer, Cryptocurrency support
- **Transaction Dashboard**: Real-time status tracking for all transactions
- **Review Period**: Built-in countdown timer for buyer review period
- **Installment Payments**: Optional split payment system for large transactions
- **Dispute Resolution**: UI for handling transaction disputes

✅ Smart Notification System

- **Real-time Alerts**: In-app notifications for all transaction stages
- **Categorized Notifications**: Organized by type (payments, deliveries, disputes, etc.)
- **Action-based Notifications**: Direct links to relevant pages and actions
- **Notification Center**: Centralized notification management

✅ Clean Architecture

- **Component Organization**: Logical separation by feature (escrow, notifications, auth, ui)
- **Consolidated Types**: Single source of truth for TypeScript definitions
- **Modular Structure**: Easily maintainable and extensible codebase
- **No Backend Dependencies**: Pure frontend implementation with mock data

🛠️ Technical Stack

- **Framework**: Next.js 14 (App Router)
- **Language**: TypeScript
- **Styling**: Tailwind CSS
- **Icons**: Lucide React
- **State Management**: React Context API
- **Development**: ESLint, Hot Reloading

🚀 Getting Started

Installation

```
cd frontend
npm install
```

Development

```
npm run dev
```

Build

```
npm run build
npm start
```

📋 Type Definitions

The project uses a consolidated type system in `src/types/index.ts` with the following main interfaces:

User Types

- `User`, `Seller`, `Buyer` - User account interfaces
- `Purchase`, `Subscription` - User activity types

Transaction Types

- `EscrowTransaction` - Main transaction interface
- `TransactionStatus` - Transaction state enum
- `PaymentMethod` - Payment method configuration
- `TransactionInstallment` - Installment payment details

Notification Types

- `Notification` - Notification interface
- `NotificationType` - All notification categories
- `PurchaseRequest` - Purchase request interface

🔧 Configuration

Constants (`src/constants/index.ts`)

- `NOTIFICATION_TYPES` - All notification type constants
- `SUBSCRIPTION_PLANS` - Available subscription plans
- `STORAGE_KEYS` - Local storage key definitions
- `VALIDATION_RULES` - Form validation configurations

Mock Data

- Located in `src/data/` for development and testing
- Easily replaceable with real API calls

🎨 Component Architecture

Escrow Components

- **Modular Design**: Each component handles a specific aspect of the escrow flow
- **Reusable Components**: TransactionCard, StatusBadge, etc. can be used across the app
- **State Management**: Uses React Context for global state (notifications, transactions)

Notification Components

- **Smart Notifications**: Context-aware notifications based on transaction events
- **Filtering & Categorization**: Advanced notification management
- **Real-time Updates**: Reactive notification system

🔒 Security Considerations

- **Client-side Only**: No sensitive backend logic or database connections
- **Mock Data**: All transaction data is simulated for demonstration
- **Type Safety**: Full TypeScript coverage for improved code reliability
- **Input Validation**: Form validation rules defined in constants

📱 Responsive Design

- **Mobile-first**: Responsive design using Tailwind CSS

- **Accessible**: Following web accessibility guidelines
- **Cross-browser**: Compatible with modern browsers

🔄 State Management

Notification Context

- Global state for all notifications
- Mock transaction management for demonstration
- Easy integration points for real backend services

Local Storage

- Defined storage keys in constants
- Consistent data persistence patterns
- Easy migration to external state management if needed

Note: This is a frontend-only implementation with mock data. All transaction flows, payments, and notifications are simulated for demonstration purposes. To connect to a real backend, replace the mock data and context logic with API calls to your chosen backend service.