CampusLoop Frontend Technical Documentation

Project Overview

CampusLoop is a React Native mobile application developed using the Expo framework. It functions as a comprehensive campus marketplace and community platform, facilitating interactions within a university environment. The application enables users to:

- Buy, sell, and rent items within the campus community.
- Communicate via real-time chat functionality.
- Discover and participate in campus-specific events.
- Manage personal listings and rental history.

Application Configuration

```
{
 "name": "CampusLoop",
 "version": "1.0.0",
 "platform": ["iOS", "Android", "Web"],
 "package": "com.shifat74.CampusLoop"
}
```

Technology Stack

Core Framework

- **React Native** (v0.81.5): Primary mobile application framework.
- **Expo** (SDK 54): Development platform and build toolchain.
- **React** (v19.1.0): JavaScript library for building user interfaces.

Navigation

- **Expo Router** (v6.0.14): File-system based routing solution.
- **React Navigation**: Handles Bottom Tabs and Native Stack navigation primitives.

Styling

- **NativeWind** (v4.1.23): Tailwind CSS implementation for React Native.
- **TailwindCSS** (v3.4.17): Utility-first CSS framework.

State Management

- **React Context API**: Manages global application state, specifically user authentication.
- **AsyncStorage**: Handles persistent local storage for authentication tokens.

Backend Integration

- **Axios** (v1.11.0): Promise-based HTTP client for REST API interaction.
- **Base URL**: https://campusloop-theta.vercel.app/api/v1

Firebase Services

- **Firebase SDK** (v12.5.0): Backend-as-a-Service implementation.
- **Cloud Firestore**: NoSQL cloud database used for real-time chat messaging.

Project Structure

The project follows a standard Expo Router directory structure.

```
g:\frontend\
├── app/                     # Main application source code
│   ├── (auth)/              # Authentication route group
│   │   ├── Sign.jsx         # Login interface
│   │   ├── SignUp.jsx       # Registration interface
│   │   └── _layout.jsx      # Authentication group layout
│   ├── (tabs)/              # Main tab navigation group
│   │   ├── Home.jsx         # Main feed dashboard
│   │   ├── Explore.jsx      # Search and discovery interface
│   │   ├── Post.jsx         # Listing creation interface
│   │   ├── Events.jsx       # Event listing interface
│   │   ├── Profile.jsx      # User profile management
│   │   └── _layout.jsx      # Tab navigator configuration
│   ├── products/            # Dynamic product routes
│   │   ├── [productId].jsx  # Dynamic product detail view
│   │   └── index.jsx        # Product listing wrapper
│   ├── screens/             # Standalone utility screens
│   │   ├── ChatScreen.jsx   # Real-time chat interface
│   │   └── EventDetailScreen.jsx # Detailed event view
│   ├── Components/          # Reusable UI components
│   │   ├── RentForm.jsx     # Rental request modal
│   │   ├── Slider.jsx       # Image carousel component
│   │   ├── Splash.jsx       # Custom splash screen
│   │   ├── MyListing.jsx    # User inventory list
│   │   ├── MyRental.jsx     # User rental history list
│   │   └── Home/            # Home-specific sub-components
│   ├── _layout.jsx          # Root application layout
│   ├── index.js             # Application entry point
│   └── global.css           # Global stylesheet
├── Context/                 # React Context providers
│   └── UserContext.jsx      # Authentication state provider
```

```
├── lib/                    # Utility libraries and configurations
│   └── api.js              # Axios instance configuration
├── firebase/               # Firebase service configuration
│   ├── firebase.config.js  # App initialization
│   └── chatService.js      # Firestore interaction methods
├── assets/                 # Static resources
├── .env                    # Environment variables
├── tailwind.config.js      # Tailwind configuration
└── app.json                # Expo project configuration
```

Architecture & Design Patterns

File-Based Routing (Expo Router)

CampusLoop utilizes Expo Router's file-system based routing:

- **Route Groups:** Directories wrapped in parentheses, such as (auth) and (tabs), allow for logical grouping of routes without affecting the URL structure.
- **Dynamic Routes:** Files wrapped in brackets, such as [productId].jsx, handle dynamic path parameters.
- **Layouts:** _layout.jsx files define persistent UI elements (headers, tab bars) that wrap child routes.

Component Architecture

The application follows a hierarchical component structure rooted in the UserProvider.

```
Root Layout (_layout.jsx)
└── UserProvider (Global Auth State)
    ├── (auth) Group
    │   ├── Sign
    │   └── SignUp
    ├── (tabs) Group
    │   ├── Home
    │   ├── Explore
    │   ├── Post
    │   ├── Events
    │   └── Profile
    └── Screens
        ├── ChatScreen
        └── EventDetailScreen
```

Design Patterns Implementation

1. **Provider Pattern:** The UserContext wraps the entire application stack, ensuring authentication state is accessible to all components without prop drilling.
2. **Custom Hooks:** Encapsulation of logic, such as useUser, to simplify context consumption.
3. **Interceptor Pattern:** Axios interceptors are utilized to automatically inject authentication headers into private API requests and handle 401 Unauthorized responses globally.

Navigation & Routing

The navigation graph consists of a conditional entry point (Splash), leading to a main Tab Navigator, with nested Stack Navigators for specific flows.

Route Definitions

Route	**File Path**	**Description**
/	index.js	Initial splash screen with auto-redirect.
/Home	(tabs)/Home.jsx	Primary dashboard.
/Explore	(tabs)/Explore.jsx	Search and discovery.
/Post	(tabs)/Post.jsx	Form for creating listings.
/Events	(tabs)/Events.jsx	Event calendar and lists.
/Profile	(tabs)/Profile.jsx	User account management.
/products/[productId]	products/[productId].jsx	Dynamic product detail view.
/screens/ChatScreen	screens/ChatScreen.jsx	Messaging interface.
/Sign	(auth)/Sign.jsx	Authentication entry.

State Management

UserContext

Location: Context/UserContext.jsx

This context manages the user's authentication session. It provides the following interface:

```
[
  "const value = {",
  "    user,",
  "    loading,",
  "    error,",
  "    fetchUser,",
  "    updateUser,",
  "    clearUser,",
  "    isAuthenticated: !!user,",
  "  };"
]
```

Operational Logic:

- **fetchUser:** Attempts to retrieve the user profile using the stored token.
- **clearUser:** Removes tokens from AsyncStorage and resets the state to null (Logout).

API Integration

Configuration

The application uses two distinct Axios instances defined in lib/api.js.

1. Public API

- **Instance Name:** publicAPI
- **Usage:** Authentication endpoints (Login, Register), Public item fetching.
- **Headers:** Standard JSON content type.

2. Private API

- **Instance Name:** privateAPI
- **Usage:** Protected routes (Profile, Create Item, Rent Item).
- **Request Interceptor:** Retrieves accessToken from AsyncStorage and appends it to the Authorization header.

- **Response Interceptor:** Monitors for 401 Unauthorized status codes to trigger automatic token cleanup and user logout.

Firebase Integration

Firestore Chat Service

Location: firebase/chatService.js

The application uses Firestore for real-time messaging. The data model is structured to optimize query performance for chat histories.

```
chats/
└── {chatId}/           # ID format: "userA_userB" (alphabetically sorted)
    └── messages/
        └── {messageId}
            ├── senderId: string
            ├── receiverId: string
            ├── text: string
            └── timestamp: ServerTimestamp
```

Styling & Theming

The application utilizes **NativeWind**, allowing Tailwind CSS classes to be applied directly to React Native components.

Color Palette (Campus Theme)

Color Name	Hex Code	Usage
Forest	#2D473E	Primary branding, headers, active states.
Sage	#8EA77B	Secondary accents, success states.
Mint	#D7E4C2	Background highlights, tags.
Slate	#788881	Body text, inactive icons.
Pearl	#F6F2EE	App background, cards.

Component Styling Example

```
<View className="bg-campus-forest px-6 py-4 rounded-campus shadow-campus">
  <Text className="text-headline-sm font-bold text-white">
    CampusLoop
  </Text>
</View>
```

Key Features

1. **Marketplace:** Facilitates the listing of items for rent or sale. Includes category filtering and search capabilities.
2. **Rental System:** A dedicated workflow for requesting item rentals, including date selection via date pickers and automatic cost calculation based on duration.
3. **Real-Time Chat:** Integrated messaging system allowing buyers and sellers to communicate instantly, powered by Firestore.
4. **Event Discovery:** A directory of campus events, allowing students to browse details and schedules.
5. **Profile Management:** Centralized dashboard for users to view their listings, track active rentals, and manage account details.

Components

RentForm

A modal component located in Components/RentForm.jsx.

- **Function:** Handles the rental request logic.
- **Inputs:** Start Date, End Date, Notes.
- **Logic:** Calculates total price based on the daily rate and selected date range. Submits data to the /rent endpoint.

Slider

A carousel component located in Components/Slider.jsx.

- **Function:** Displays featured images or promotions.
- **Behavior:** Auto-scrolls every 4 seconds. Includes pagination indicators.

Chat Interface

Built using react-native-gifted-chat.

- **Function:** Renders the chat UI (bubbles, input bar).
- **Integration:** Maps Firestore data to the format required by GiftedChat.

Authentication Flow

The authentication process ensures secure access to protected resources.

1. **Initialization:** On app launch, UserContext checks AsyncStorage for an accessToken.
2. **Validation:**
   - **If Token Exists:** The app attempts to fetch user details via /user/me.
     - *Success:* User state is populated; access is granted.
     - *Failure (401):* Token is invalid; storage is cleared; user is redirected to Login.
   - **If Token Missing:** User is redirected to the Sign-in screen.
3. **Login Action:**
   - User submits credentials to /auth/login.
   - Server responds with accessToken.
   - Token is stored in AsyncStorage.
   - App fetches user profile and updates context.
4. **Logout Action:**
   - clearUser is invoked.
   - Tokens are removed from storage.
   - User is redirected to the authentication route.

Build & Deployment

Development Environment

To initialize the development server:

```
# Start Metro Bundler
npm start

# Clear cache if necessary
npm start --clear
```

```
Environment Variables:
Ensure the .env file contains the correct API endpoint before building:
API_BASE_URL=https://campusloop-theta.vercel.app/api/v1
```