Phase 1: Infrastructure Setup & Backend API Development

Objective: Establish the foundation for Localite by setting up the core infrastructure, backend API, blockchain framework, AI-driven matchmaking, and cross-platform compatibility. This phase is crucial for ensuring a scalable, secure, and high-performance system.

Task 1 - Development Environment Setup

Step-by-Step Implementation with Roo Code Prompts

We'll begin by setting up the development environment for Localite. Since we use React Native for the app and Rust for the backend with blockchain integration, we need to configure our workspace accordingly.

Step 1: Install Required Tools & Dependencies

Objective:

Set up VS Code with necessary extensions and install required tools for React Native, Rust, and blockchain development.

Action Plan:

1. **Install VS Code** if not already installed.
2. **Install Roo Code AI Agent** extension in VS Code.
3. **Install Node.js & npm (for React Native development).**
4. **Install Rust & Cargo (for backend development).**
5. **Install React Native CLI and Expo CLI.**
6. **Set up Android Studio and Xcode for emulator support.**

Roo Code Prompts for Step 1

Use these prompts in Roo Code AI agent in VS Code to install and verify dependencies.

1️⃣ **To check if Node.js & npm are installed:**

```
Check if Node.js and npm are installed. If not, guide me through the
installation process for my OS.
```

2️⃣ **To install React Native CLI & Expo CLI:**

```
Generate the command to install React Native CLI and Expo CLI globally
using npm.
```

3️⃣ **To install Rust & Cargo:**

```
How do I install Rust and Cargo on my system? Provide the installation
command.
```

4️⃣ **To verify Rust installation:**

```
Check if Rust and Cargo are installed on my system and provide the
version details.
```

5️⃣ **To set up Android Studio for React Native: Skip for later**

```
Generate step-by-step instructions to configure Android Studio for
React Native development, including setting up an emulator.
```

6️⃣ **To set up Xcode for iOS development: Skip for later**

```
Guide me through the steps to install and configure Xcode for React
Native development on macOS.
```

Step 2: Initialize the Project Repository

Objective:

Set up the project structure for Localite, including the GitHub repository and codebase organization.

Action Plan:

1. **Create a new GitHub repository** for Localite.
2. **Initialize the React Native project** in the repo.
3. **Initialize the Rust backend project** in the repo.

4. **Set up Git version control & commit initial files.**

Roo Code Prompts for Step 2

1️⃣**To create a new React Native project:**

```
Generate the command to create a new React Native project using Expo
with TypeScript support.
```

2️⃣**To initialize a Rust backend project:**

```
Generate the command to create a new Rust project for the backend with
Actix-Web framework.
```

3️⃣**To set up Git and push the initial commit:**

```
Generate the commands to initialize Git, add a .gitignore file for
React Native and Rust, commit changes, and push to GitHub.
```

Step 3: Install Essential Packages & Dependencies

Objective:

Install required libraries and frameworks for smooth development.

Action Plan:

1. **Install React Navigation** for screen management.
2. **Install Axios** for API requests.
3. **Install SQLite for local data storage.**
4. **Install Rust dependencies (Actix-Web, Diesel for DB, Serde for serialization).**

Roo Code Prompts for Step 3

1️⃣**To install React Navigation:**

```
Generate the npm command to install React Navigation and its
dependencies for a React Native project.
```

2️⃣**To install Axios for API requests:**

```
Generate the npm command to install Axios for handling API requests in
React Native.
```

3️⃣ **To install SQLite for local storage:**

```
Generate the npm command to install SQLite for local storage in a
React Native app.
```

4️⃣ **To install Actix-Web in Rust:**

```
Generate the Cargo command to add Actix-Web, Serde, and Diesel to a
Rust project.
```

Step 4: Set Up Environment Variables & Configurations

Objective:

Ensure a secure and flexible configuration setup for API keys, database credentials, and other sensitive data.

Action Plan:

1. **Create an `.env` file** for managing environment variables.
2. **Set up database configuration for SQLite (React Native) and PostgreSQL (Rust backend).**
3. **Configure API keys and security tokens.**

Roo Code Prompts for Step 4

1️⃣ **To set up an `.env` file for React Native:**

```
How do I create an .env file in React Native, and how can I access
variables inside the app?
```

2️⃣ **To configure Diesel ORM for PostgreSQL in Rust:**

```
Generate the steps to set up Diesel ORM with PostgreSQL for a Rust
backend.
```

Step 5: Verify Setup & Run Initial Tests

Objective:

Ensure that the development environment is working correctly before moving forward.

Action Plan:

1. **Run the React Native app on an emulator.**
2. **Start the Rust backend server and check if it runs without errors.**
3. **Verify database connections for both frontend and backend.**

Roo Code Prompts for Step 5

1️⃣ To run the React Native app:

```
Generate the command to start the React Native app on an Android/iOS
emulator.
```

2️⃣ To start the Rust backend server:

```
Generate the command to run the Actix-Web server in Rust and listen
for incoming requests.
```

3️⃣ To check database connections:

```
How can I test if my SQLite database in React Native and PostgreSQL
database in Rust backend are connected successfully?
```

Task 2 - Backend API Architecture & Development

Step-by-Step Implementation with Roo Code Prompts

In this step, we will design and develop the backend API using **Rust with Actix-Web**. The backend will handle authentication, user management, matchmaking, blockchain interactions, and payment processing.

Step 1: Define API Architecture & Endpoints

Objective:

Structure the API with well-defined routes and modular architecture to handle all required functionalities.

Action Plan:

1. **Define API structure** – Organize routes into modules (e.g., Authentication, Users, Matchmaking, Payments, Blockchain).
2. **Set up API versioning** – Ensure future compatibility.
3. **Use RESTful principles** – Ensure clean, scalable API endpoints.
4. **Plan for WebSockets** – Enable real-time notifications.

Roo Code Prompts for Step 1:

1️⃣ To generate a modular Actix-Web project structure:

```
How can I structure an Actix-Web project using modules for
authentication, matchmaking, payments, and blockchain interactions?
```

2️⃣ To set up API versioning in Actix-Web:

```
How do I implement API versioning in an Actix-Web project?
```

3️⃣ To implement WebSockets in Actix-Web:

```
Generate a simple WebSocket implementation for real-time notifications
in Actix-Web.
```

Step 2: Implement User Authentication (JWT-based)

Objective:

Secure the API with JWT authentication for login, signup, and protected routes.

Action Plan:

1. **Install JWT libraries** – Use `jsonwebtoken` crate.
2. **Create an authentication module** – Handle signup, login, and token validation.
3. **Secure protected routes** – Require valid JWT tokens for access.
4. **Implement token expiration & refresh mechanism.**

Roo Code Prompts for Step 2:

1️⃣ To install JWT authentication dependencies:

```
Generate the Cargo command to install jsonwebtoken and bcrypt for
authentication in Actix-Web.
```

2️⃣ To implement JWT-based authentication in Actix-Web:

```
Generate Rust code to implement JWT authentication for an Actix-Web
API, including signup, login, and token verification.
```

3️⃣ To protect routes using JWT middleware:

```
How can I create middleware in Actix-Web to protect routes using JWT
authentication?
```

Step 3: Implement User & Role Management

Objective:

Allow different user roles (e.g., Local Helper, Non-Local Seeker, Admin) with appropriate permissions.

Action Plan:

1. **Define user roles & permissions** – Store in database.
2. **Create endpoints for user management** – Get user info, update profile, delete account.
3. **Implement admin control features** – Manage users and disputes.

Roo Code Prompts for Step 3:

1️⃣ To define user roles and permissions:

```
How can I implement user roles and permissions in a Rust API using
Actix-Web and Diesel ORM?
```

2️⃣ **To create user management endpoints:**

```
Generate Actix-Web routes for user management (get profile, update
profile, delete account).
```

Step 4: Implement AI-powered Matchmaking System

Objective:

Match users based on location, skillset, rating, and availability using AI.

Action Plan:

1. **Integrate AI model for matchmaking** – Train model using user data.
2. **Use location-based ranking** – Prioritize nearby helpers.
3. **Implement rating-based prioritization** – Higher-rated helpers get higher priority.

Roo Code Prompts for Step 4:

1️⃣ **To implement AI matchmaking using user attributes:**

```
How can I integrate an AI-based matchmaking system in Rust,
considering location, skillset, and rating?
```

2️⃣ **To use geolocation in Rust for matchmaking:**

```
Generate Rust code to calculate the distance between two users based
on their geolocation.
```

Step 5: Implement Fraud Detection System

Objective:

Detect suspicious activities using behavioral analytics.

Action Plan:

1. **Monitor user activities** – Identify unusual behavior.
2. **Use AI-powered fraud detection** – Flag risky interactions.
3. **Restrict fraudulent accounts** – Implement account bans.

Roo Code Prompts for Step 5:

1️⃣ To track suspicious behavior in Rust:

```
How can I implement fraud detection in Actix-Web by monitoring user
behavior patterns?
```

2️⃣ To integrate AI for fraud detection:

```
Generate a simple AI model in Rust that detects fraudulent user
behavior based on activity logs.
```

Step 6: Implement Escrow Payment System

Objective:

Hold payments securely until task completion to prevent fraud.

Action Plan:

1. **Integrate blockchain escrow** – Store funds securely.
2. **Release payments upon task verification** – Only after confirmation.
3. **Allow dispute resolution mechanism** – Escrow remains locked if a dispute arises.

Roo Code Prompts for Step 6:

1️⃣ To implement a simple escrow system in Rust:

```
How can I implement a smart contract-based escrow system for secure
payments in Rust?
```

2️⃣ To manage dispute resolution for escrow payments:

```
Generate Rust code to handle dispute resolution in a smart contract
escrow system.
```

Step 7: API Testing & Documentation

Objective:

Ensure API endpoints work correctly and document them for frontend integration.

Action Plan:

1. **Use Postman for testing** – Validate API functionality.
2. **Generate OpenAPI documentation** – Use `utoipa` crate.

Roo Code Prompts for Step 7:

1️⃣ To generate OpenAPI documentation for Actix-Web:

```
Generate OpenAPI documentation for an Actix-Web API using utoipa
crate.
```

2️⃣ To automate API testing with Postman:

```
How can I create a Postman collection to test my Actix-Web API
endpoints?
```

Task 3 - Frontend Development (React Native App)

Objective: Develop the Localite mobile application using React Native, ensuring a cross-platform experience for **iOS and Android**. The frontend will interact with our Rust-based backend via REST APIs and integrate **AI-powered matchmaking, real-time notifications, and blockchain payments**.

Step 1: Setup React Native Development Environment

Action Plan:

1. **Install Node.js, React Native CLI, and Expo** for faster development.
2. **Configure dependencies** – Navigation, state management, API handling, etc.
3. **Ensure cross-platform compatibility** – Use platform-specific styles where required.

Roo Code Prompts for Step 1:

1️⃣ To install and set up React Native for a new project:

```
How can I set up a new React Native project with Expo for
cross-platform development?
```

2️⃣ To configure React Navigation in a React Native app:

```
Generate the required setup to implement navigation in a React Native
app using React Navigation.
```

Step 2: Implement Authentication (Login/Signup) UI & API Integration

Action Plan:

1. **Design UI for authentication screens** (Login, Signup, Forgot Password).
2. **Connect with backend authentication API** using JWT.
3. **Implement secure storage for authentication tokens** (AsyncStorage or SecureStore).

Roo Code Prompts for Step 2:

1️⃣ To create a user authentication UI in React Native:

```
Generate a React Native UI for login and signup screens using React
Native Paper or Material UI.
```

2️⃣ To securely store JWT tokens in React Native:

```
How can I securely store authentication tokens in a React Native app?
```

Step 3: Implement User Dashboard & Profile Management

Action Plan:

1. **Create a user dashboard** displaying tasks, notifications, and wallet balance.
2. **Develop a profile management section** for editing user details.
3. **Integrate backend APIs for fetching user details and updating profiles.**

Roo Code Prompts for Step 3:

1️⃣ To create a user dashboard UI in React Native:

```
Generate a React Native UI for a user dashboard displaying tasks,
notifications, and wallet balance.
```

2️⃣ To integrate a profile update API in React Native:

```
How can I connect a React Native app to a Rust-based Actix-Web backend
for profile management?
```

Step 4: Implement AI-powered Matchmaking System UI

Action Plan:

1. **Create a "Find Help" UI** for posting tasks and receiving proposals.
2. **Display recommended local users** based on AI matchmaking.
3. **Use location-based ranking to show the nearest helpers first.**

Roo Code Prompts for Step 4:

1️⃣ To implement AI-based recommendations in a React Native app:

```
How can I integrate AI-based recommendations in a React Native app,
prioritizing location and skillset?
```

2️⃣ To implement real-time matchmaking notifications:

```
Generate a React Native implementation for real-time notifications
using Firebase or WebSockets.
```

Step 5: Implement Fraud Detection UI

Action Plan:

1. **Warn users when suspicious activity is detected.**
2. **Provide a reporting mechanism for users to flag scams.**
3. **Integrate a risk score indicator for local users.**

Roo Code Prompts for Step 5:

1️⃣ To show a fraud risk score in the UI:

```
Generate a React Native UI component to display a fraud risk score
with color indicators (green, yellow, red).
```

2️⃣ To implement a reporting system for fraud detection:

```
How can I create a report button in React Native that submits fraud
complaints to the backend?
```

Step 6: Implement Escrow Payment System UI

Action Plan:

1. **Create a secure wallet screen** to show balance and transaction history.
2. **Allow payments to be held in escrow** until task completion.
3. **Implement a dispute resolution UI.**

Roo Code Prompts for Step 6:

1️⃣ To build a wallet UI in React Native:

```
Generate a React Native UI for a user wallet showing balance,
transaction history, and escrow payments.
```

2️⃣ To integrate blockchain payments in React Native:

```
How can I connect a React Native app to a Rust-based blockchain
backend for processing transactions?
```

Step 7: API Integration & Testing

Action Plan:

1. **Integrate all backend APIs** with Axios or Fetch API.
2. **Test API responses** with Postman before frontend implementation.
3. **Ensure error handling and security** in API requests.

Roo Code Prompts for Step 7:

1️⃣ **To handle API errors in React Native gracefully:**

```
How can I implement global error handling for API requests in React
Native using Axios?
```

2️⃣ **To write unit tests for API calls in React Native:**

```
Generate test cases for API integration in React Native using Jest.
```

Task 4 - Blockchain Foundation & Smart Contracts

Objective: Implement Localite's **custom blockchain and smart contract system** to facilitate secure transactions, reward distribution, and escrow payments. This step involves setting up the blockchain infrastructure, writing smart contracts, and integrating with the backend.

Step 1: Setting Up the Blockchain Infrastructure

Action Plan:

1. **Choose the consensus mechanism** (PoS for efficiency and staking rewards).

2. **Set up a Rust-based blockchain framework** like Substrate for scalability.
3. **Deploy a testnet** to test transactions before going live.

Roo Code Prompts for Step 1:

1️⃣To set up a Rust-based blockchain network:

```
How can I create a Rust-based blockchain using Substrate for a
decentralized platform like Localite?
```

2️⃣To configure a local testnet for blockchain transactions:

```
Generate step-by-step instructions to deploy a local blockchain
testnet with Rust and Substrate.
```

Step 2: Writing Smart Contracts for Localite

Action Plan:

1. **Develop smart contracts for transactions, rewards, and escrow.**
2. **Write logic for staking rewards and fraud detection triggers.**
3. **Ensure security using best blockchain practices.**

Roo Code Prompts for Step 2:

1️⃣To create a smart contract for escrow payments in Rust:

```
Generate a Rust-based smart contract for handling escrow payments
securely using Substrate.
```

2️⃣To write a staking smart contract for user rewards:

```
Generate a Rust-based smart contract that allows users to stake their
earned tokens and receive rewards.
```

Step 3: Blockchain API Integration with Backend

Action Plan:

1. **Connect smart contracts with Actix-Web backend.**
2. **Implement API endpoints for sending transactions and checking balances.**
3. **Ensure secure communication between the backend and blockchain.**

Roo Code Prompts for Step 3:

1️⃣ To connect Rust-based smart contracts with an Actix-Web backend:

```
How can I integrate a Rust-based blockchain smart contract with an
Actix-Web backend for secure transactions?
```

2️⃣ To fetch blockchain transaction data via API calls:

```
Generate an API in Actix-Web that retrieves transaction history from a
Rust-based blockchain.
```

Step 4: Frontend Integration for Blockchain Transactions

Action Plan:

1. **Implement wallet UI for users to check balances and send transactions.**
2. **Integrate escrow and staking functionalities into the React Native app.**
3. **Ensure smooth blockchain interactions on the frontend.**

Roo Code Prompts for Step 4:

1️⃣ To create a blockchain wallet UI in React Native:

```
Generate a React Native UI for a blockchain wallet that displays
balance, transactions, and staking rewards.
```

2️⃣ To implement blockchain-based payments in React Native:

```
How can I integrate a Rust-based blockchain payment system into a
React Native app for seamless transactions?
```

Step 5: Security & Smart Contract Testing

Action Plan:

1. **Perform rigorous testing of smart contracts to prevent exploits.**
2. **Use tools like Rust's built-in testing framework to validate security.**
3. **Deploy smart contracts to a testnet before launching on the mainnet.**

Roo Code Prompts for Step 5:

1️⃣**To write and run security tests for Rust-based smart contracts:**

```
Generate a test suite in Rust to check for vulnerabilities in smart
contracts deployed on Substrate.
```

2️⃣**To deploy smart contracts on a testnet for security auditing:**

```
How can I deploy and test Rust-based smart contracts on a testnet
before launching them on the mainnet?
```

Task 5 - AI & Machine Learning Setup

Objective: Implement AI/ML functionalities to power Localite's matchmaking, fraud detection, and reputation systems. This phase aims to integrate models that use location, skillset, rating, and activity data to enhance user experience and platform security.

---

1. Define AI Use Cases

- **Location-Based Matchmaking:**
  Prioritize local helper notifications based on proximity, skillset, and user ratings. The system should rank nearby users higher, ensuring that non-locals receive timely assistance from the most relevant helpers.

- **Fraud Detection:**
  Develop an anomaly detection model that monitors both online and physical activities to flag suspicious behavior. This model should analyze patterns from user activity logs,

transaction histories, and engagement metrics to detect potential scams or fraud.

- **Reputation & Incentive System:**
  Create an AI-driven reputation system that assigns scores to local users based on past performance, user feedback, and task completion success. The model should reward consistent, helpful behavior while penalizing unreliable activity.

---

2. Action Plan & Implementation Steps

Step 2.1: Set Up AI Development Environment

- **Install Necessary Libraries:**
  Ensure that TensorFlow, PyTorch, or any Rust-based ML libraries (e.g., Linfa) are installed for model development.
- **Configure Data Pipeline:**
  Set up data collection mechanisms from user interactions (with proper anonymization) for training purposes.

Roo Code Prompt:

```
How do I set up a new Python virtual environment and install
TensorFlow, PyTorch, and any Rust ML libraries (like Linfa) needed for
developing AI models?
```

Step 2.2: Develop the Location-Based Matchmaking Model

- **Design the Model:**
  Create a model that inputs user data (geolocation, skillset, ratings) and outputs a ranked list of local users.
- **Implement Distance Calculation:**
  Use geolocation algorithms to compute the distance between the non-local user and potential local helpers.

Roo Code Prompts:

```
Generate Python code to calculate the Haversine distance between two
sets of coordinates.
```

```
Generate a sample neural network model using TensorFlow that takes
user features (including location, skills, and ratings) and outputs a
match score.
```

Step 2.3: Implement Fraud Detection

- **Design the Anomaly Detection Model:**
  Utilize unsupervised learning techniques to detect unusual patterns in user behavior.
- **Integrate Multi-Domain Data:**
  Combine online interaction data with physical activity logs (if available) to create a comprehensive risk profile.

Roo Code Prompts:

```
Generate a simple anomaly detection model using an autoencoder in
TensorFlow to identify unusual activity patterns in a dataset.

Provide a prompt for integrating multi-dimensional data (user activity
logs, transaction data) for fraud detection in a Python-based model.
```

Step 2.4: Build the Reputation Scoring Algorithm

- **Design a Scoring Mechanism:**
  Develop a reputation model (e.g., reinforcement learning or a simpler weighted scoring system) to evaluate the quality of local assistance.
- **Reward & Penalty Mechanism:**
  Ensure the model outputs actionable scores that can directly influence user rewards and lower transaction fees for high-reputation users.

Roo Code Prompts:

```
Generate a Python script that implements a weighted scoring system for
users based on task completion, user ratings, and feedback.

Generate code for a basic reinforcement learning model using PyTorch
to adjust user reputation scores over time.
```

Step 2.5: Integration with Backend & API Exposure

- **Create AI Service APIs:**
  Develop endpoints in the Actix-Web backend that call the AI models for matchmaking, fraud detection, and reputation scoring.
- **Ensure Scalability:**
  Use asynchronous processing to handle real-time AI predictions without impacting API performance.

Roo Code Prompts:

```
Generate Rust code to create an asynchronous Actix-Web endpoint that
calls an external Python-based AI service for matchmaking
recommendations.

Provide a prompt to integrate AI model responses into the backend API,
ensuring the results are properly formatted for frontend consumption.
```

---

3. Testing & Validation

- **Unit Testing:**
  Write unit tests for each AI model component to ensure accuracy and stability.
- **Integration Testing:**
  Simulate end-to-end scenarios where the backend calls the AI service and returns recommendations or fraud alerts.
- **Monitoring & Feedback:**
  Implement logging and monitoring to continuously refine AI models based on real user data (ensuring privacy and compliance).

Roo Code Prompt:

```
Generate test cases in Python (using pytest) for the anomaly detection
model to verify its performance on a sample dataset.

Generate Rust code for integration tests in Actix-Web that simulate
API calls to the AI matchmaking endpoint.
```

---

4. Documentation & Code Comments

- **Inline Comments & Docstrings:**
  Every function, model, and API endpoint should have detailed comments explaining its purpose, input parameters, and output.
- **API Documentation:**
  Use tools like Swagger/OpenAPI (with the `utoipa` crate) to document AI service endpoints.
- **Model Documentation:**
  Maintain a README file in the AI module directory describing model architecture, training data, and performance metrics.

Roo Code Prompt:

```
Generate a template README for the AI module that documents the
purpose, usage, and training details of the matchmaking and fraud
detection models.
```

Conclusion

This detailed breakdown provides a structured roadmap for integrating AI/ML capabilities into Localite. By following these steps and using the provided Roo Code prompts in VS Code, you can efficiently build, test, and integrate AI models that power location-based matchmaking, fraud detection, and reputation scoring in your decentralized help platform.

Task 6 – Security & Testing Strategy

Objective: Establish robust security measures and a comprehensive testing strategy to ensure that Localite is secure, reliable, and scalable. This phase involves implementing best practices for data protection, authentication, network security, and thorough testing of all components—including the backend, blockchain, AI models, and mobile application.

---

1. Security Measures

1.1 Data Encryption & Secure Communication

- **Action:**

    - Encrypt sensitive user data (e.g., passwords, personal information, transactions) using AES-256.
    - Use HTTPS (TLS) for all API communications.
- **Roo Code Prompts:**
    - *"Generate Rust code to encrypt data using AES-256 in an Actix-Web endpoint."*
    - *"Provide instructions for setting up TLS/SSL in an Actix-Web server."*

1.2 Authentication & Access Control

- **Action:**
    - Implement JWT-based authentication for secure API access.
    - Integrate two-factor authentication (2FA) for critical actions (especially payments).
    - Use RBAC to restrict access to sensitive endpoints.
- **Roo Code Prompts:**
    - *"Generate Rust code to implement JWT authentication middleware in Actix-Web."*
    - *"How can I add 2FA in a Rust-based API, and what libraries are recommended?"*

1.3 DDoS & Network Protection

- **Action:**
    - Set up rate limiting and request throttling to mitigate DDoS attacks.
    - Use a service like Cloudflare to protect public API endpoints.
- **Roo Code Prompts:**
    - *"Generate middleware in Actix-Web for rate limiting incoming requests."*
    - *"Guide me on configuring Cloudflare for an Actix-Web application."*

1.4 Blockchain & Smart Contract Security

- **Action:**
    - Conduct security audits on smart contracts before mainnet deployment.
    - Implement rigorous testing for staking, escrow, and reward distribution contracts.
- **Roo Code Prompts:**
    - *"Generate guidelines for performing a security audit on Rust-based smart contracts using Substrate."*
    - *"How can I write tests for my blockchain smart contracts in Rust?"*

---

2. Testing Strategy

2.1 Unit Testing

- **Action:**
    - Write unit tests for each module (authentication, help requests, matchmaking, blockchain interactions).
    - Use Rust's built-in testing framework for backend modules.
    - Use Jest for React Native components.
- **Roo Code Prompts:**
    - *"Generate sample unit tests for a Rust function using the built-in testing framework."*
    - *"How do I write unit tests in React Native using Jest?"*

2.2 Integration Testing

- **Action:**
    - Ensure end-to-end functionality by testing the interaction between the frontend and backend APIs.
    - Use Postman to validate API endpoints.
- **Roo Code Prompts:**
    - *"Generate a Postman collection template for testing key Actix-Web API endpoints."*
    - *"Provide a sample integration test for a Rust Actix-Web endpoint."*

2.3 Blockchain Testing

- **Action:**
    - Deploy smart contracts on a testnet and simulate various transaction scenarios (including staking and escrow).
    - Use tools such as Ganache (if applicable) or Substrate’s testing modules.
- **Roo Code Prompts:**
    - *"Generate a test suite in Rust for smart contract transactions on a Substrate testnet."*

2.4 Load & Performance Testing

- **Action:**
    - Simulate high-traffic scenarios to ensure the backend and blockchain can handle peak loads.
    - Use tools like Locust or JMeter to perform load tests.
- **Roo Code Prompts:**
    - *"Generate instructions to set up Locust for load testing an Actix-Web API."*
    - *"Provide a JMeter test plan template for testing blockchain API performance."*

2.5 Security Testing & Penetration Testing

- **Action:**

    - Conduct penetration tests on the backend and smart contracts to identify vulnerabilities.
    - Use automated tools and manual reviews.
- **Roo Code Prompts:**
    - *"Guide me through creating a penetration testing checklist for an Actix-Web application."*
    - *"Generate guidelines for performing security tests on Rust-based smart contracts."*

---

3. Documentation & Continuous Monitoring

3.1 Code Documentation

- **Action:**
    - Add inline comments and docstrings in both Rust and React Native codebases.
    - Use tools like Swagger/OpenAPI (with the `utoipa` crate) to document API endpoints.
- **Roo Code Prompts:**
    - *"Generate a sample docstring template for a Rust function in Actix-Web."*
    - *"How do I set up OpenAPI documentation for an Actix-Web API using utoipa?"*

3.2 Continuous Integration/Continuous Deployment (CI/CD)

- **Action:**
    - Implement CI/CD pipelines (using Jenkins, GitHub Actions, or GitLab CI) for automated testing and deployment.
    - Integrate automated security scans and code quality checks.
- **Roo Code Prompts:**
    - *"Generate a GitHub Actions YAML configuration for CI/CD of a Rust Actix-Web project."*
    - *"Provide instructions to integrate security scanning in a CI/CD pipeline for a blockchain-based project."*

3.3 Monitoring & Logging

- **Action:**
    - Set up logging for all API interactions and blockchain transactions.
    - Use monitoring tools (e.g., Prometheus, Grafana) to track performance and security events.
- **Roo Code Prompts:**
    - *"How do I integrate Prometheus monitoring in an Actix-Web application?"*

    - *"Generate a basic Grafana dashboard configuration for monitoring API performance."*

---

4. Conclusion

This Security & Testing Strategy ensures that Localite is built with a strong emphasis on security, reliability, and performance. By leveraging best practices in encryption, authentication, and rigorous testing methodologies, the platform will be robust against attacks and scalable to meet user demand.

Task 7 – Important References for Documentation & Code Comments

Objective: Ensure that every part of the Localite codebase is well-documented, from backend APIs and smart contracts to mobile UI components and AI modules. This task aims to create a uniform documentation standard that improves code maintainability, eases future development, and facilitates collaboration. Proper documentation will include inline code comments, comprehensive docstrings, external API documentation, and code reference guides.

---

1. Code Commenting & Docstring Standards

1.1 Backend (Rust / Actix-Web)

- **Inline Comments:**
    - Describe the purpose of each function, module, and complex logic block.
    - Reference related business logic or external documentation when necessary.
- **Docstrings:**
    - Use Rust's documentation comments (`///`) for each public function, detailing parameters, return types, and potential errors.

Example Prompt for Roo Code:

```
Generate a Rust docstring for an Actix-Web handler function that
creates a new user. Include details about the expected JSON request,
response structure, and error handling.
```

1.2 Blockchain & Smart Contracts (Rust / Substrate)

- **Inline Comments:**
  - Clearly annotate smart contract functions, especially the ones handling escrow, staking, and token transactions.
- **Docstrings:**
  - Use detailed comments to explain the logic of consensus mechanisms, transaction fees, and security features.

Example Prompt for Roo Code:

```
Generate detailed inline comments for a Rust-based smart contract
function that processes escrow payments using Substrate.
```

1.3 AI & Machine Learning Modules (Python/Rust)

- **Inline Comments:**
  - Comment on model architectures, input features, and expected outputs for matchmaking, fraud detection, and reputation scoring.
- **Docstrings:**
  - Use Python docstrings (triple quotes) for functions and classes, specifying the purpose, parameters, return types, and example usage.

Example Prompt for Roo Code:

```
Generate a Python docstring for an AI function that calculates the
Haversine distance between two geolocations, including parameter
descriptions and an example.
```

1.4 Frontend (React Native & React.js)

- **Inline Comments:**
  - Document key UI components, state management logic, and API integration calls.
- **Docstrings:**
  - Use JSDoc comments for React components to describe props, state, and lifecycle methods.

Example Prompt for Roo Code:

```
Generate JSDoc comments for a React Native component that displays the
user dashboard, including prop type descriptions and expected
behavior.
```

---

2. API Documentation

2.1 OpenAPI/Swagger Integration

- **Backend API Documentation:**
  - Use the `utoipa` crate (for Rust) or Swagger to auto-generate comprehensive API documentation.
  - Ensure that each endpoint is well-documented with request/response schemas, authentication requirements, and error codes.

Example Prompt for Roo Code:

```
Generate an OpenAPI specification for an Actix-Web endpoint that
handles user login, including request and response schemas.
```

---

3. External Documentation References

3.1 Repository README

- **Overview:**
  - Create a comprehensive README file that covers project overview, setup instructions, coding standards, and contribution guidelines.
- **Content:**
  - Project purpose, technology stack, development roadmap, and key contacts.

Example Prompt for Roo Code:

```
Generate a template for a comprehensive README.md for the Localite
project, including sections for introduction, setup, usage, and
contribution guidelines.
```

3.2 Module-Specific Documentation

- **Separate Markdown Files:**
  - Maintain detailed documentation in separate files for each major module (Backend, Blockchain, AI, Frontend).
  - Include architecture diagrams, flowcharts, and API endpoints.

Example Prompt for Roo Code:

```
Generate a template for a module documentation file (e.g.,
BACKEND_DOCS.md) that outlines the architecture, key components, and
API endpoints for the Localite backend.
```

---

4. Code Repository Best Practices

- **Commit Messages:**
  - Use clear, descriptive commit messages referencing specific features or fixes.
- **Branching Strategy:**
  - Follow a Git branching model (e.g., Git Flow) to separate development, testing, and production code.
- **Code Reviews:**
  - Ensure that all code merges are peer-reviewed with emphasis on documentation quality.

---

5. Continuous Integration/Deployment (CI/CD) Integration

- **Documentation Checks:**
  - Integrate tools that check for documentation coverage and enforce inline comments during CI/CD pipelines.

Example Prompt for Roo Code:

```
Generate a GitHub Actions configuration snippet that runs a
documentation coverage tool for a Rust project.
```

Conclusion

This documentation framework will ensure that every component of the Localite codebase is well-documented, from API endpoints to smart contract logic and UI components. By integrating these standards and using AI prompts via Roo Code in VS Code, development will be smoother, more maintainable, and ready for future scaling and enhancements.