Peridot Protocol

Introduction

High-Level Overview

Peridot Protocol is a cross-chain lending and borrowing platform designed to unify liquidity and abstract complexity in decentralized finance (DeFi). Built using a **Hub & Spoke architecture** and powered by **Wormhole**, Peridot allows users to deposit collateral on one blockchain and seamlessly borrow assets on another — without requiring them to manage multiple wallets, bridges, or gas fees.

The protocol is optimized for both experienced DeFi users and newcomers. For crypto-savvy users, it offers seamless, composable access to multi-chain capital and advanced use cases such as perpetual trading.

Core components of the architecture include:

- **Hub Contract:** A smart contract acting as the single source of truth for user positions, collateral, borrowing logic, and liquidations.

- **Spoke Contracts:** Lightweight smart contracts deployed across supported chains, acting as user entry points and forwarding actions to the Hub via Wormhole.

- **Wormhole Token Bridge and Messaging Layer:** Enabling secure and verifiable asset transfers and cross-chain intent messaging between Spoke chains and the Hub.

- **Oracles (Pyth, Witnet):** Providing real-time price feeds for collateral valuation and liquidation risk assessment.

Peridot Protocol is designed to provide a unified, capital-efficient, and user-friendly experience that turns fragmented DeFi ecosystems into one seamless global lending market.

Definitions, Acronyms and Abbreviations

Term / Acronym	Definition
Peridot Protocol	A cross-chain lending and borrowing platform using a Hub & Spoke model to unify liquidity across blockchains.
DeFi (Decentralized Finance)	Blockchain-based financial applications that operate without centralized intermediaries.
Hub & Spoke Model	An architectural design where a central contract (Hub) manages logic and user states, and peripheral contracts (Spokes) handle user interactions across blockchains.
Smart Contract	Code deployed on a blockchain that executes automatically under defined conditions.

Collateral	Assets deposited by users to secure loans and borrowing power within Peridot.
Liquidity	The availability of assets in markets to be borrowed, repaid or liquidated.
Oracle	A service that delivers real-world data (such as asset prices) to blockchain smart contracts.
Pyth Network	A decentralized oracle network providing real-time price feeds for financial assets.
Witnet	A decentralized oracle network delivering real-world data to smart contracts in a trust-minimized manner.
Wormhole	A cross-chain messaging protocol facilitating secure interoperability and token transfers between blockchains.
Token Bridge	Mechanism allowing assets to move seamlessly across blockchain networks.
NTT (Native Token Transfer)	Wormhole feature enabling native token transfers across supported chains.
Peridottroller (Controller)	Core smart contract managing borrowing, repayments, liquidations and market validations.
PErc20 / PEther	Collateral token contracts responsible for minting, redeeming, and accruing interest on deposited assets.
Interest Rate Model	Smart contract calculating borrowing rates based on market utilization.
Liquidation	Automatic repayment process when a user's position falls below the collateral requirement.
Gas Fees	Fees paid to execute transactions and smart contract operations on blockchains.
WalletConnect	Protocol enabling connection between users' wallets and decentralized applications.

AppKit	Toolkit for building dApps with wallet integration.
The Graph	Decentralized protocol for querying blockchain data and indexing events.
Foundry	Smart contract development and testing toolkit, primarily for EVM-compatible chains.
Rust	Systems programming language used to write Soroban smart contracts.
Solidity	A programming language for smart contracts on EVM-compatible blockchains (used optionally in cross-chain Spokes).
TypeScript	Typed superset of JavaScript used for frontend and backend services.

Architecture constraints

Regulatory Constraints

- **No KYC/AML Integration**: Peridot Protocol is designed as a permissionless DeFi platform, intentionally omitting Know Your Customer (KYC) and Anti-Money Laundering (AML) procedures to maintain user privacy and inclusivity.

- **Jurisdictional Compliance**: While the protocol itself does not enforce regional restrictions, users are responsible for adhering to their local regulations regarding cryptocurrency usage and DeFi participation.

Software Constraints

- **Soroban Smart Contract Platform**: Development is centered on Soroban, Stellar's smart contract platform, which utilizes WebAssembly (WASM) and Rust. This choice influences contract design, emphasizing performance, safety, and compatibility within the Stellar network.

- **Stellar Network Limitations**: Operating within the Stellar ecosystem imposes certain constraints, such as transaction throughput limits and specific consensus

mechanisms, which must be considered in protocol design.

- **Cross-Chain Interoperability**: Integration with Wormhole for cross-chain functionality requires adherence to its protocols and standards, impacting how assets and messages are transferred between networks.

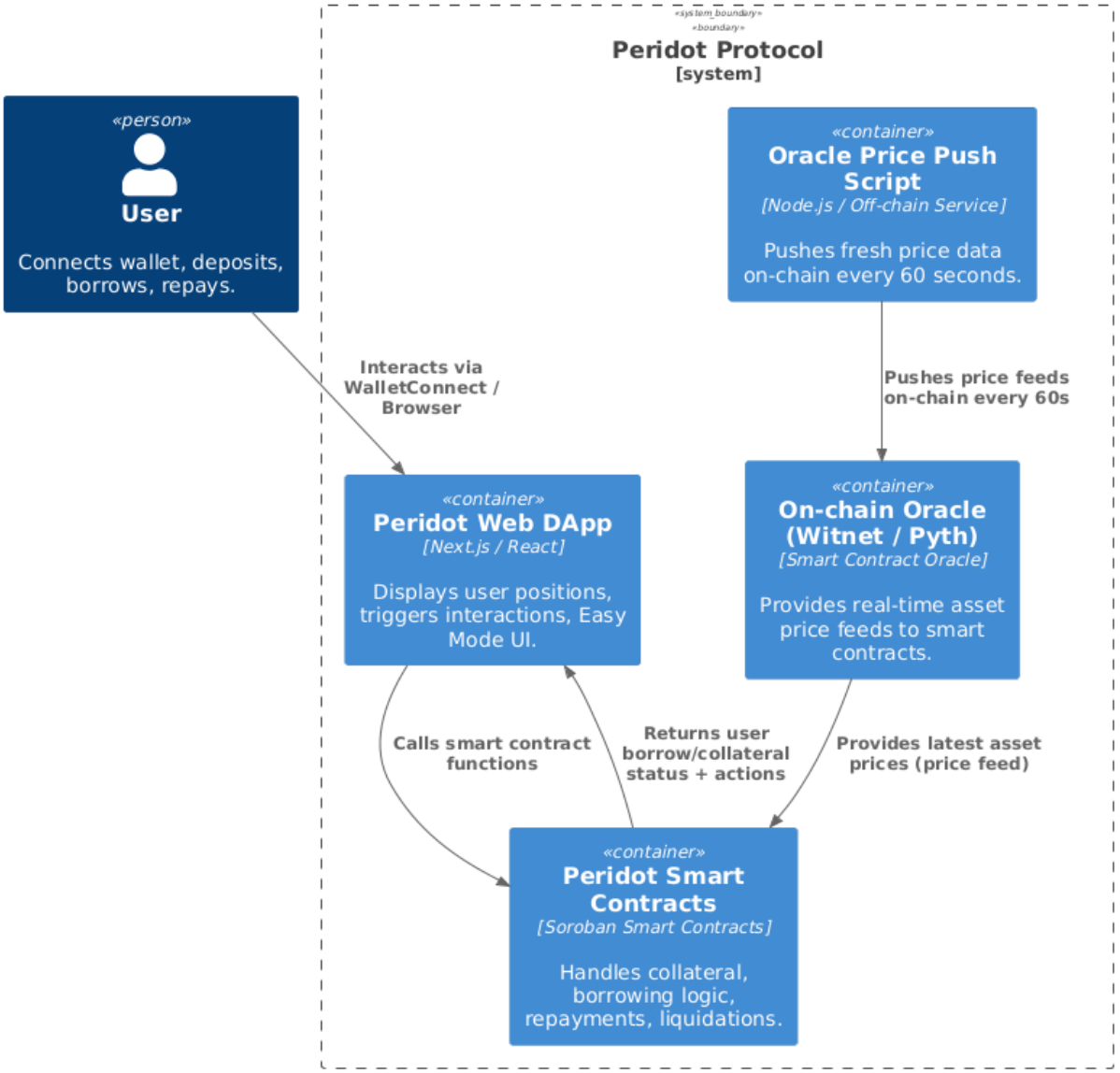
Hardware Constraints

- **Node Requirements**: Running a full node on the Stellar network necessitates specific hardware capabilities, including sufficient storage, memory, and processing power to handle the ledger's data and transaction validation processes.

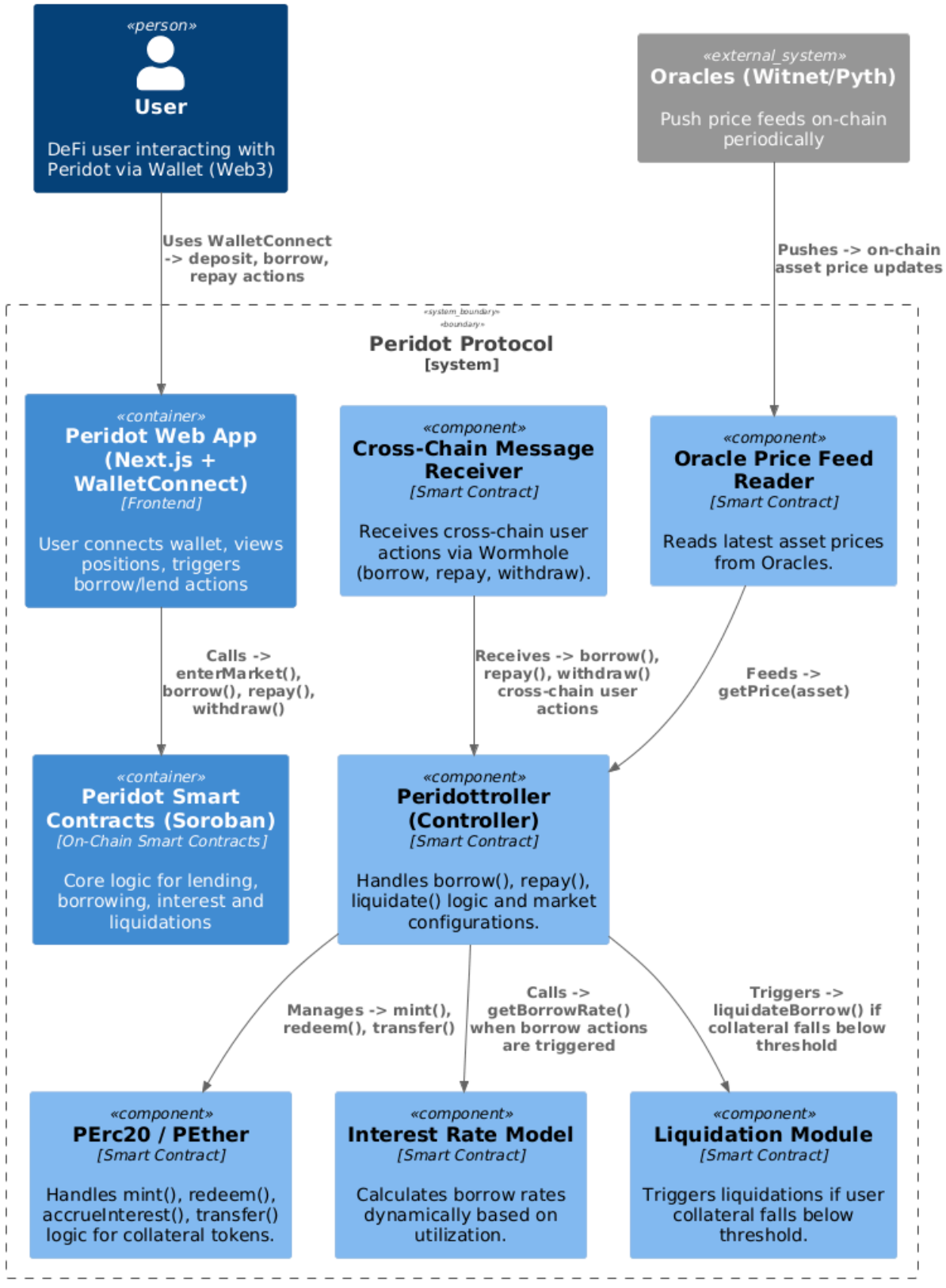
- **Scalability Considerations**: As the user base grows, infrastructure must scale accordingly to maintain performance and reliability.

Architecture Overview

C4 L1 Diagram: High-Level Architecture



C4 L2 Diagram: Zoom into the Peridot System



Deliverables

Deliverable 1: Core Lending & Borrowing (MVP / Single Chain)

User Story:

> As a DeFi user, I want to supply collateral and borrow assets on Peridot in a simple and transparent way, so that I can access capital without selling my assets.

Acceptance Criteria:

- Smart contracts (Peridottroller, PErc20, Interest Model) deployed on Soroban.
- Collateral supply, borrow, repay and liquidate flows operational.
- Frontend MVP allowing wallet connection, viewing balances, and initiating transactions.

Deliverable 2: Cross-Chain Borrowing & Repayment (Hub & Spoke Model via Wormhole)

User Story:

> As a cross-chain user, I want to deposit collateral on one chain and borrow or repay on another chain seamlessly, so that I can optimize liquidity across ecosystems without manually bridging.

Acceptance Criteria:

- Spoke contracts deployed on minimum 2 chains and connected to the Hub.
- Cross-chain deposits and borrow actions forwarded via Wormhole.
- Successful cross-chain repay + withdraw actions routed back to the originating chain.
- Automated price feeds integrated to enforce borrow limits.

Deliverable 3: Yield Generation & Capital Efficiency Features

User Story:

> As a liquidity provider, I want my deposited assets to earn yield automatically based on borrowing demand, so that I can passively grow my holdings and benefit from market activity.

Acceptance Criteria:

- Dynamic Interest Rate Model implemented and optimized.
- Frontend updated to show real-time APY and earned yield.
- Initial liquidity incentives prepared to bootstrap markets.

Primary Contracts

Peridottroller (Controller)

> The main coordinator and validator of lending & borrowing actions.

Key Methods:

- enterMarket() → User enables collateral
- borrow() → Checks collateral, calls Oracle + Interest Rate model, then issues borrowed asset
- repayBorrow() → Repays outstanding borrow and updates state
- liquidateBorrow() → Allows others to liquidate undercollateralized users
- getAccountLiquidity() → Calculates user liquidity (used for borrowing and liquidation checks)

PErc20 / PEther (Collateral Tokens)

User deposit token representation + interest accrual

Key Methods:

- mint() → Mints collateral tokens when user deposits
- redeem() → Redeems user's deposited tokens
- accrueInterest() → Accrues interest on borrows based on utilization

---

Oracle Price Reader

Reads updated prices from Oracles (Pyth / Witnet)

Key Methods:

- getPrice(asset) → Gets latest price for collateral evaluation
- updatePrice(asset, price) → (Only Oracle can call) Updates price feed

---

Interest Rate Model

Dynamic interest rate calculation based on market utilization

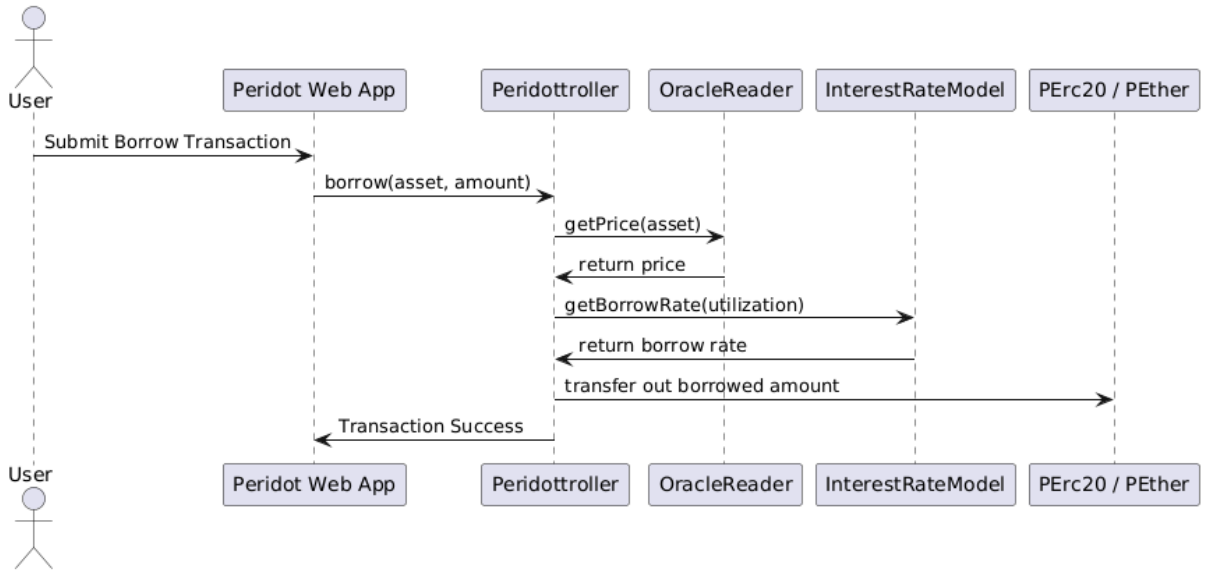
Key Methods:

- getBorrowRate(utilization) → Returns borrow APR based on current market utilization

---

Liquidation Module (within Peridottroller)

Internal logic triggered via liquidateBorrow()



Technology Stack

Smart Contract Platform

- **Soroban (Stellar)** (Smart Contracts → Rust + Soroban SDK)
- **Rust (Solana)** (Spoke Chains & to make Stellar Tokens available on Solana via NTT)
- **Solidity (EVM)** (Spoke Chains & to make Stellar Tokens available on all EVM Chains via NTT & Token Bridge)

Frontend

Core Framework & Language

- Next.js (v15.2.4) - React framework for production
- React (v19.1.0) - UI library
- TypeScript - Type-safe JavaScript

UI Components & Styling

- Tailwind CSS- Utility-first CSS framework
- shadcn/ui - Component library built on Radix UI
- Lucide React- Icon library
- Framer Motion - Animation library
- Embla Carousel - Carousel component
- Sonner - Toast notifications

- Vaul - Drawer component

State Management & Data Handling

- React Hook Form - Form handling
- Zod - Schema validation
- Wagmi - Ethereum hooks
- Viem - Ethereum TypeScript interface
- Ethers.js - Ethereum library

Analytics & Monitoring

- PostHog - Product analytics and user behavior tracking
  - Page view tracking
  - User behavior analytics
  - Feature flag management
  - Custom event tracking

User Alerting via Mail / Wallet

- Dialect SDK

Wallet Integration

- Wallet Connect Multichain Appkit

Bridge Integration

- **Wormhole TokenBridge** (Cross-chain interoperability)

Oracles

- **Pyth Network** and/or **Witnet** (Price feeds)

Cross-Chain Messaging

- **Wormhole TokenBridge** (Cross-chain interoperability)
- **Wormhole SDK**
- **Wormhole NTT CLI**

Backend (Oracle Feed + Monitoring)

- **Node.js / Typescript Script** → Push Oracle price data to smart contract every 60 seconds

Database

- No heavy backend data store required, but optionally:
    - **PostgreSQL (optional)** → For indexing events + user interface history (optional) & capturing User Mails

Dev Tools

- **Foundry** →Cross-Chain Smart Contract testing framework
- **Rust/Cargo + Soroban CLI** → Contract deployment and local testing
- **Docker / CI Pipelines** → Test + deploy workflow

Automated Testing

- **Smart Contracts:** Stellar/Rust Tests, Foundry Tests