Synapse Ledger: Go + React TSX Developer Blueprint

A Decentralized Data Marketplace for AI Training

Optimized for Go Backend + React TypeScript + Pinata Storage

1. Executive Summary

Synapse Ledger is a decentralized data marketplace that enables individuals to monetize their anonymized data while providing AI developers with ethically-sourced training datasets. The platform uses smart contracts for transparent royalty distribution, Pinata for reliable IPFS storage, and implements privacy-preserving mechanisms to protect contributor data.

Tech Stack Overview:

- **Frontend**: React 18 + TypeScript + Vite + Tailwind CSS
- **Backend**: Go + Gin/Fiber + PostgreSQL
- **Smart Contracts**: Solidity + Hardhat + OpenZeppelin
- **Storage**: Pinata IPFS + backup redundancy
- **Blockchain**: Polygon (with Ethereum L2 compatibility)

2. Problem Context & Vision

Current Problem

The AI boom is powered by massive datasets scraped from the internet without consent or compensation. Large corporations monetize public and private data while original creators receive nothing, creating an imbalanced and opaque data economy. AI developers also struggle with data provenance verification and quality assurance.

Our Solution

Synapse Ledger creates a transparent, ethical data economy using blockchain technology:

- **For Data Contributors**: Voluntary participation with fair compensation
- **For AI Developers**: Access to verified, ethically-sourced datasets
- **For the Ecosystem**: Immutable provenance and transparent transactions

Long-term Vision

- Establish industry standards for ethical AI data sourcing
- Create the largest marketplace for verified training data
- Enable sustainable economic models for individual data creators
- Bridge the gap between data rights and AI development needs

3. Features & User Stories

Core Features

For Data Contributors

- **Data Upload & Anonymization**: Upload photos, text, code repositories with automatic anonymization
- **Stake Management**: Choose specific AI projects to contribute data to
- **Royalty Tracking**: Real-time view of earnings from data usage
- **Privacy Controls**: Granular control over data sharing and anonymization levels

For AI Developers

- **Data Marketplace**: Browse available datasets with metadata and quality scores
- **Purchase & Access**: Secure payment processing and API access to datasets
- **Usage Analytics**: Track data consumption and associated costs
- **Provenance Verification**: Complete audit trail for regulatory compliance

For Platform

- **Smart Contract Automation**: Automatic royalty distribution based on usage
- **Quality Verification**: Community-driven data quality scoring
- **Privacy Preservation**: On-device preprocessing and federated learning compatibility
- **Governance**: DAO-based platform governance and parameter updates

User Stories

As a Data Contributor:

- I want to upload my anonymized photos and receive payment when they're used for AI training
- I want to see transparent tracking of how much I've earned and from which projects
- I want to maintain complete anonymity while still receiving payments

As an AI Developer:

- I want to purchase ethically-sourced training data with verified provenance
- I want programmatic API access to datasets after purchase verification
- I want to verify the ethical compliance of my training data for regulatory purposes

As a Platform User:

- I want to participate in governance decisions about platform parameters
- I want to stake tokens to verify data quality and earn rewards
- I want to see transparent analytics about platform usage and economics

4. Full System Architecture

Frontend Layer (React + TypeScript)

- **Framework**: React 18 with TypeScript + Vite for fast development
- **Styling**: Tailwind CSS with custom design system and dark theme
- **Web3 Integration**: Wagmi v2 + RainbowKit for wallet connections
- **State Management**: Zustand for global state, TanStack Query for server state
- **HTTP Client**: Axios with interceptors for API communication
- **Authentication**: JWT tokens with Web3 signature verification

Smart Contract Layer (Solidity)

- **Development**: Hardhat with TypeScript, OpenZeppelin contracts
- **Main Contracts**: DataRegistry, RoyaltyDistributor, SynapseToken (ERC-20)
- **Standards**: ERC-721 for data NFTs, ERC-20 for utility token
- **Security**: ReentrancyGuard, Ownable, Pausable patterns

Backend Layer (Go)

- **Framework**: Gin (or Fiber) for high-performance HTTP server
- **Database**: PostgreSQL with GORM for ORM functionality
- **Authentication**: JWT middleware with Web3 signature validation
- **File Processing**: Go standard library for image/text processing
- **Event Listening**: go-ethereum client for smart contract event monitoring
- **Background Jobs**: Go routines with channels for async processing

Database Layer (PostgreSQL)

- **User Management**: Contributors and developer account information
- **Data Metadata**: File descriptions, quality scores, usage analytics
- **Transaction History**: Payment records and royalty distributions
- **Caching**: Redis for session management and temporary data
- **Search**: PostgreSQL full-text search for data discovery

Storage Layer (Pinata IPFS)

- **Primary**: Pinata for reliable IPFS pinning and management
- **Backup**: Redundant pinning across multiple IPFS nodes
- **Metadata**: JSON metadata stored on IPFS with CID references
- **Access Control**: Signed URLs and JWT-based access for purchased data
- **CDN**: Pinata's dedicated gateway for fast content delivery

Blockchain Infrastructure

- **Primary Network**: Polygon for low-cost transactions
- **Development**: Mumbai testnet for development and testing
- **Node Provider**: Alchemy for reliable RPC connections
- **Wallet Support**: MetaMask, WalletConnect, Coinbase Wallet

5. Smart Contract Design

Core Contracts

DataRegistry.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.19;

import "@openzeppelin/contracts/security/ReentrancyGuard.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/security/Pausable.sol";

contract DataRegistry is Ownable, ReentrancyGuard, Pausable {
    struct DataPool {
        address creator;
```

```
        string ipfsHash;
        string metadataHash;
        uint256 pricePerAccess;
        uint256 totalContributions;
        bool isActive;
        mapping(address =&gt; uint256) contributorShares;
    }

    mapping(uint256 =&gt; DataPool) public dataPools;
    mapping(address =&gt; uint256[]) public creatorPools;
    uint256 public nextPoolId;

    event DataPoolCreated(uint256 indexed poolId, address indexed creator, string ipfsHash);
    event DataPurchased(uint256 indexed poolId, address indexed buyer, uint256 amount);
    event ContributionAdded(uint256 indexed poolId, address indexed contributor, uint256 share);

    function createDataPool(
        string memory _ipfsHash,
        string memory _metadataHash,
        uint256 _pricePerAccess
    ) external returns (uint256) {
        require(bytes(_ipfsHash).length &gt; 0, "Invalid IPFS hash");
        require(_pricePerAccess &gt; 0, "Price must be greater than 0");

        uint256 poolId = nextPoolId++;
        DataPool storage pool = dataPools[poolId];
        pool.creator = msg.sender;
        pool.ipfsHash = _ipfsHash;
        pool.metadataHash = _metadataHash;
        pool.pricePerAccess = _pricePerAccess;
        pool.isActive = true;

        creatorPools[msg.sender].push(poolId);

        emit DataPoolCreated(poolId, msg.sender, _ipfsHash);
        return poolId;
    }

    function purchaseDataAccess(uint256 _poolId) external payable nonReentrant {
        DataPool storage pool = dataPools[_poolId];
        require(pool.isActive, "Pool is not active");
        require(msg.value &gt;= pool.pricePerAccess, "Insufficient payment");

        // Distribute royalties to contributors
        _distributeRoyalties(_poolId, msg.value);

        emit DataPurchased(_poolId, msg.sender, msg.value);
    }

    function _distributeRoyalties(uint256 _poolId, uint256 _amount) internal {
        // Implementation for royalty distribution
    }
}
```

RoyaltyDistributor.sol

```
contract RoyaltyDistributor is Ownable, ReentrancyGuard {
    mapping(address =&gt; uint256) public pendingRoyalties;
    mapping(uint256 =&gt; mapping(address =&gt; uint256)) public poolContributions;

    event RoyaltyDistributed(uint256 indexed poolId, address indexed contributor, uint256 amount);
    event RoyaltyClaimed(address indexed contributor, uint256 amount);

    function distributeRoyalties(uint256 _poolId, uint256 _totalAmount) external {
        // Calculate and distribute royalties based on contributions
        // Implementation details...
    }

    function claimRoyalties() external nonReentrant {
```

```
        uint256 amount = pendingRoyalties[msg.sender];
        require(amount &gt; 0, "No royalties to claim");

        pendingRoyalties[msg.sender] = 0;
        payable(msg.sender).transfer(amount);

        emit RoyaltyClaimed(msg.sender, amount);
    }
}
```

6. Go Backend Architecture

Project Structure

```
backend/
├── cmd/
│   └── server/
│       └── main.go
├── internal/
│   ├── api/
│   │   ├── handlers/
│   │   ├── middleware/
│   │   └── routes/
│   ├── auth/
│   ├── blockchain/
│   ├── config/
│   ├── models/
│   ├── services/
│   └── storage/
├── pkg/
│   ├── ipfs/
│   ├── utils/
│   └── validator/
├── migrations/
├── docker/
└── go.mod
```

Core Models and Database Schema

```
// internal/models/models.go
package models

import (
    "time"
    "github.com/google/uuid"
    "gorm.io/gorm"
)

type User struct {
    ID          uuid.UUID `gorm:"type:uuid;primary_key;default:gen_random_uuid()" json:"id"`
    WalletAddr  string    `gorm:"unique;not null" json:"wallet_address"`
    Username    string    `gorm:"unique" json:"username,omitempty"`
    Email       string    `gorm:"unique" json:"email,omitempty"`
    Role        string    `gorm:"default:'contributor'" json:"role"` // contributor, developer, admin
    CreatedAt   time.Time `json:"created_at"`
    UpdatedAt   time.Time `json:"updated_at"`

    DataPools   []DataPool `gorm:"foreignKey:CreatorID" json:"data_pools,omitempty"`
    Purchases   []Purchase `gorm:"foreignKey:BuyerID" json:"purchases,omitempty"`
}

type DataPool struct {
    ID              uuid.UUID `gorm:"type:uuid;primary_key;default:gen_random_uuid()" json:"id"`
    BlockchainID    uint64    `gorm:"unique;not null" json:"blockchain_id"`
    CreatorID       uuid.UUID `gorm:"type:uuid;not null" json:"creator_id"`
    IPFSHash        string    `gorm:"not null" json:"ipfs_hash"`
    MetadataHash    string    `json:"metadata_hash"`
```

```
    Name            string    `gorm:"not null" json:"name"`
    Description     string    `json:"description"`
    DataType        string    `gorm:"not null" json:"data_type"` // image, text, code, audio
    PricePerAccess  string    `gorm:"not null" json:"price_per_access"` // Wei amount as string
    QualityScore    float64   `gorm:"default:0" json:"quality_score"`
    TotalPurchases  uint64    `gorm:"default:0" json:"total_purchases"`
    IsActive        bool      `gorm:"default:true" json:"is_active"`
    CreatedAt       time.Time `json:"created_at"`
    UpdatedAt       time.Time `json:"updated_at"`

    Creator         User       `gorm:"foreignKey:CreatorID" json:"creator,omitempty"`
    Purchases       []Purchase `gorm:"foreignKey:DataPoolID" json:"purchases,omitempty"`
    Contributors    []Contribution `gorm:"foreignKey:DataPoolID" json:"contributors,omitempty"`
}

type Purchase struct {
    ID           uuid.UUID `gorm:"type:uuid;primary_key;default:gen_random_uuid()" json:"id"`
    DataPoolID   uuid.UUID `gorm:"type:uuid;not null" json:"data_pool_id"`
    BuyerID      uuid.UUID `gorm:"type:uuid;not null" json:"buyer_id"`
    Amount       string    `gorm:"not null" json:"amount"` // Wei amount as string
    TxHash       string    `gorm:"unique;not null" json:"tx_hash"`
    BlockNumber  uint64    `gorm:"not null" json:"block_number"`
    AccessToken  string    `gorm:"unique" json:"access_token"`
    ExpiresAt    time.Time `json:"expires_at"`
    CreatedAt    time.Time `json:"created_at"`

    DataPool     DataPool `gorm:"foreignKey:DataPoolID" json:"data_pool,omitempty"`
    Buyer        User     `gorm:"foreignKey:BuyerID" json:"buyer,omitempty"`
}

type Contribution struct {
    ID           uuid.UUID `gorm:"type:uuid;primary_key;default:gen_random_uuid()" json:"id"`
    DataPoolID   uuid.UUID `gorm:"type:uuid;not null" json:"data_pool_id"`
    ContributorID uuid.UUID `gorm:"type:uuid;not null" json:"contributor_id"`
    SharePercent float64   `gorm:"not null" json:"share_percent"`
    TotalEarned  string    `gorm:"default:'0'" json:"total_earned"` // Wei amount as string
    CreatedAt    time.Time `json:"created_at"`
    UpdatedAt    time.Time `json:"updated_at"`

    DataPool     DataPool `gorm:"foreignKey:DataPoolID" json:"data_pool,omitempty"`
    Contributor  User     `gorm:"foreignKey:ContributorID" json:"contributor,omitempty"`
}
```

Main Server Setup

```
// cmd/server/main.go
package main

import (
    "log"
    "os"

    "github.com/gin-gonic/gin"
    "synapse-ledger/internal/api/routes"
    "synapse-ledger/internal/blockchain"
    "synapse-ledger/internal/config"
    "synapse-ledger/internal/models"
    "gorm.io/driver/postgres"
    "gorm.io/gorm"
)

func main() {
    // Load configuration
    cfg := config.Load()

    // Initialize database
    db, err := gorm.Open(postgres.Open(cfg.DatabaseURL), &amp;gorm.Config{})
    if err != nil {
        log.Fatal("Failed to connect to database:", err)
    }
```

```
    // Auto-migrate models
    if err := db.AutoMigrate(&amp;models.User{}, &amp;models.DataPool{}, &amp;models.Purchase{}, &amp;mod
        log.Fatal("Failed to migrate database:", err)
    }

    // Initialize blockchain client
    blockchainClient, err := blockchain.NewClient(cfg.RPCUrl, cfg.PrivateKey)
    if err != nil {
        log.Fatal("Failed to initialize blockchain client:", err)
    }

    // Start event listener in background
    go blockchainClient.StartEventListener(db)

    // Initialize Gin router
    router := gin.Default()

    // Setup routes
    routes.SetupRoutes(router, db, blockchainClient)

    // Start server
    port := os.Getenv("PORT")
    if port == "" {
        port = "8080"
    }

    log.Printf("Server starting on port %s", port)
    if err := router.Run(":" + port); err != nil {
        log.Fatal("Failed to start server:", err)
    }
}
```

Blockchain Event Listener

```
// internal/blockchain/event_listener.go
package blockchain

import (
    "context"
    "log"
    "math/big"

    "github.com/ethereum/go-ethereum"
    "github.com/ethereum/go-ethereum/accounts/abi"
    "github.com/ethereum/go-ethereum/common"
    "github.com/ethereum/go-ethereum/core/types"
    "github.com/ethereum/go-ethereum/ethclient"
    "gorm.io/gorm"
    "synapse-ledger/internal/models"
)

type EventListener struct {
    client   *ethclient.Client
    db       *gorm.DB
    contract common.Address
    abi      abi.ABI
}

func NewEventListener(client *ethclient.Client, db *gorm.DB, contractAddr string) (*EventListener, error)
    // Load contract ABI
    contractABI, err := abi.JSON(strings.NewReader(DataRegistryABI))
    if err != nil {
        return nil, err
    }

    return &amp;EventListener{
        client:   client,
        db:       db,
        contract: common.HexToAddress(contractAddr),
```

```
        abi:      contractABI,
    }, nil
}

func (el *EventListener) StartListening() {
    // Create filter query
    query := ethereum.FilterQuery{
        Addresses: []common.Address{el.contract},
    }

    // Subscribe to logs
    logs := make(chan types.Log)
    sub, err := el.client.SubscribeFilterLogs(context.Background(), query, logs)
    if err != nil {
        log.Fatal("Failed to subscribe to logs:", err)
    }

    for {
        select {
        case err := &lt;-sub.Err():
            log.Printf("Event subscription error: %v", err)
            return
        case vLog := &lt;-logs:
            if err := el.processLog(vLog); err != nil {
                log.Printf("Error processing log: %v", err)
            }
        }
    }
}

func (el *EventListener) processLog(vLog types.Log) error {
    switch vLog.Topics[0].Hex() {
    case el.abi.Events["DataPoolCreated"].ID.Hex():
        return el.handleDataPoolCreated(vLog)
    case el.abi.Events["DataPurchased"].ID.Hex():
        return el.handleDataPurchased(vLog)
    default:
        log.Printf("Unknown event: %s", vLog.Topics[0].Hex())
    }
    return nil
}

func (el *EventListener) handleDataPoolCreated(vLog types.Log) error {
    // Parse event data
    event, err := el.parseDataPoolCreatedEvent(vLog)
    if err != nil {
        return err
    }

    // Update database
    return el.db.Model(&amp;models.DataPool{}).
        Where("blockchain_id = ?", event.PoolId.String()).
        Updates(map[string]interface{}{
            "tx_hash":      vLog.TxHash.Hex(),
            "block_number": vLog.BlockNumber,
            "is_active":    true,
        }).Error
}

func (el *EventListener) handleDataPurchased(vLog types.Log) error {
    // Parse event data and create purchase record
    event, err := el.parseDataPurchasedEvent(vLog)
    if err != nil {
        return err
    }

    // Create purchase record
    purchase := &amp;models.Purchase{
        DataPoolID:  el.getDataPoolByBlockchainID(event.PoolId),
        BuyerID:     el.getUserByWallet(event.Buyer.Hex()),
        Amount:      event.Amount.String(),
```

```
        TxHash:      vLog.TxHash.Hex(),
        BlockNumber: vLog.BlockNumber,
    }

    return el.db.Create(purchase).Error
}
```

IPFS Integration with Pinata

```
// pkg/ipfs/pinata.go
package ipfs

import (
    "bytes"
    "encoding/json"
    "fmt"
    "io"
    "mime/multipart"
    "net/http"
    "os"
)

type PinataClient struct {
    APIKey    string
    APISecret string
    BaseURL   string
}

type PinataResponse struct {
    IpfsHash    string `json:"IpfsHash"`
    PinSize     int    `json:"PinSize"`
    Timestamp   string `json:"Timestamp"`
    isDuplicate bool   `json:"isDuplicate"`
}

type PinataMetadata struct {
    Name      string            `json:"name"`
    KeyValues map[string]string `json:"keyvalues,omitempty"`
}

func NewPinataClient() *PinataClient {
    return &amp;PinataClient{
        APIKey:    os.Getenv("PINATA_API_KEY"),
        APISecret: os.Getenv("PINATA_SECRET_API_KEY"),
        BaseURL:   "https://api.pinata.cloud",
    }
}

func (p *PinataClient) PinFile(file io.Reader, filename string, metadata PinataMetadata) (*PinataResponse
    var buf bytes.Buffer
    writer := multipart.NewWriter(&amp;buf)

    // Add file part
    part, err := writer.CreateFormFile("file", filename)
    if err != nil {
        return nil, err
    }

    if _, err := io.Copy(part, file); err != nil {
        return nil, err
    }

    // Add metadata
    metadataJSON, _ := json.Marshal(map[string]interface{}{
        "name":      metadata.Name,
        "keyvalues": metadata.KeyValues,
    })

    if err := writer.WriteField("pinataMetadata", string(metadataJSON)); err != nil {
        return nil, err
```

```
    }

    if err := writer.Close(); err != nil {
        return nil, err
    }

    // Create request
    req, err := http.NewRequest("POST", p.BaseURL+"/pinning/pinFileToIPFS", &amp;buf)
    if err != nil {
        return nil, err
    }

    req.Header.Set("Content-Type", writer.FormDataContentType())
    req.Header.Set("pinata_api_key", p.APIKey)
    req.Header.Set("pinata_secret_api_key", p.APISecret)

    // Execute request
    client := &amp;http.Client{}
    resp, err := client.Do(req)
    if err != nil {
        return nil, err
    }
    defer resp.Body.Close()

    if resp.StatusCode != http.StatusOK {
        body, _ := io.ReadAll(resp.Body)
        return nil, fmt.Errorf("pinata API error: %s", string(body))
    }

    var pinataResp PinataResponse
    if err := json.NewDecoder(resp.Body).Decode(&amp;pinataResp); err != nil {
        return nil, err
    }

    return &amp;pinataResp, nil
}

func (p *PinataClient) PinJSON(data interface{}, metadata PinataMetadata) (*PinataResponse, error) {
    payload := map[string]interface{}{
        "pinataContent":  data,
        "pinataMetadata": metadata,
    }

    jsonData, err := json.Marshal(payload)
    if err != nil {
        return nil, err
    }

    req, err := http.NewRequest("POST", p.BaseURL+"/pinning/pinJSONToIPFS", bytes.NewBuffer(jsonData))
    if err != nil {
        return nil, err
    }

    req.Header.Set("Content-Type", "application/json")
    req.Header.Set("pinata_api_key", p.APIKey)
    req.Header.Set("pinata_secret_api_key", p.APISecret)

    client := &amp;http.Client{}
    resp, err := client.Do(req)
    if err != nil {
        return nil, err
    }
    defer resp.Body.Close()

    var pinataResp PinataResponse
    if err := json.NewDecoder(resp.Body).Decode(&amp;pinataResp); err != nil {
        return nil, err
    }

    return &amp;pinataResp, nil
}
```

7. React Frontend Integration

Project Structure

```
frontend/
├── src/
│   ├── components/
│   │   ├── common/
│   │   ├── marketplace/
│   │   ├── upload/
│   │   └── wallet/
│   ├── hooks/
│   │   ├── useContract.ts
│   │   ├── useIPFS.ts
│   │   └── useAuth.ts
│   ├── services/
│   │   ├── api.ts
│   │   ├── blockchain.ts
│   │   └── ipfs.ts
│   ├── stores/
│   │   ├── authStore.ts
│   │   └── marketplaceStore.ts
│   ├── types/
│   ├── utils/
│   └── App.tsx
├── public/
├── vite.config.ts
└── package.json
```

Smart Contract Integration Hooks

```
// src/hooks/useContract.ts
import { useContractRead, useContractWrite, useWaitForTransaction } from 'wagmi'
import { parseEther, formatEther } from 'viem'
import { DATA_REGISTRY_ADDRESS, DATA_REGISTRY_ABI } from '../constants/contracts'
import { useAuthStore } from '../stores/authStore'

export const useDataRegistry = () =&gt; {
  const { address } = useAuthStore()

  // Read operations
  const { data: poolCount } = useContractRead({
    address: DATA_REGISTRY_ADDRESS,
    abi: DATA_REGISTRY_ABI,
    functionName: 'nextPoolId',
    watch: true,
  })

  const { data: userPools } = useContractRead({
    address: DATA_REGISTRY_ADDRESS,
    abi: DATA_REGISTRY_ABI,
    functionName: 'creatorPools',
    args: [address as `0x${string}`],
    enabled: !!address,
    watch: true,
  })

  // Write operations
  const {
    writeAsync: createPoolAsync,
    isLoading: isCreatingPool
  } = useContractWrite({
    address: DATA_REGISTRY_ADDRESS,
    abi: DATA_REGISTRY_ABI,
    functionName: 'createDataPool',
  })

  const {
```

```
    writeAsync: purchaseAccessAsync,
    isLoading: isPurchasing
  } = useContractWrite({
    address: DATA_REGISTRY_ADDRESS,
    abi: DATA_REGISTRY_ABI,
    functionName: 'purchaseDataAccess',
  })

  const createDataPool = async (
    ipfsHash: string,
    metadataHash: string,
    priceInEth: string
  ) =&gt; {
    try {
      const tx = await createPoolAsync({
        args: [ipfsHash, metadataHash, parseEther(priceInEth)],
      })

      // Wait for transaction confirmation
      const receipt = await useWaitForTransaction({ hash: tx.hash })

      // Extract pool ID from event logs
      const poolCreatedLog = receipt.logs.find(log =&gt;
        log.topics[0] === '0x...' // DataPoolCreated event signature
      )

      if (poolCreatedLog) {
        const poolId = parseInt(poolCreatedLog.topics[1], 16)
        return { success: true, poolId, txHash: tx.hash }
      }

      return { success: true, txHash: tx.hash }
    } catch (error) {
      console.error('Failed to create data pool:', error)
      throw error
    }
  }

  const purchaseDataAccess = async (poolId: number, priceInEth: string) =&gt; {
    try {
      const tx = await purchaseAccessAsync({
        args: [poolId],
        value: parseEther(priceInEth),
      })

      return { success: true, txHash: tx.hash }
    } catch (error) {
      console.error('Failed to purchase data access:', error)
      throw error
    }
  }

  return {
    // Read data
    poolCount: poolCount ? Number(poolCount) : 0,
    userPools: userPools || [],

    // Write operations
    createDataPool,
    purchaseDataAccess,

    // Loading states
    isCreatingPool,
    isPurchasing,
  }
}
```

IPFS Upload Hook

```
// src/hooks/useIPFS.ts
import { useState } from 'react'
import { uploadToIPFS, uploadJSONToIPFS } from '../services/ipfs'

interface UploadProgress {
  progress: number
  stage: 'preparing' | 'uploading' | 'pinning' | 'complete'
}

export const useIPFS = () =&gt; {
  const [uploading, setUploading] = useState(false)
  const [progress, setProgress] = useState&lt;UploadProgress&gt;({
    progress: 0,
    stage: 'preparing'
  })

  const uploadFile = async (file: File, metadata: Record&lt;string, any&gt;) =&gt; {
    setUploading(true)
    setProgress({ progress: 10, stage: 'preparing' })

    try {
      // Step 1: Prepare file and metadata
      const fileMetadata = {
        name: file.name,
        type: file.type,
        size: file.size,
        ...metadata
      }

      setProgress({ progress: 30, stage: 'uploading' })

      // Step 2: Upload file to IPFS via Pinata
      const fileResult = await uploadToIPFS(file, {
        name: `data-${Date.now()}`,
        keyvalues: {
          type: 'data-file',
          originalName: file.name,
        }
      })

      setProgress({ progress: 70, stage: 'pinning' })

      // Step 3: Upload metadata JSON to IPFS
      const metadataResult = await uploadJSONToIPFS(fileMetadata, {
        name: `metadata-${Date.now()}`,
        keyvalues: {
          type: 'metadata',
          dataHash: fileResult.IpfsHash,
        }
      })

      setProgress({ progress: 100, stage: 'complete' })

      return {
        dataHash: fileResult.IpfsHash,
        metadataHash: metadataResult.IpfsHash,
        metadata: fileMetadata,
      }
    } catch (error) {
      console.error('IPFS upload failed:', error)
      throw error
    } finally {
      setUploading(false)
      setTimeout(() =&gt; {
        setProgress({ progress: 0, stage: 'preparing' })
      }, 2000)
    }
  }
```

```
  const uploadJSON = async (data: any, metadata: Record&lt;string, any&gt;) =&gt; {
    setUploading(true)

    try {
      const result = await uploadJSONToIPFS(data, {
        name: metadata.name || `json-${Date.now()}`,
        keyvalues: metadata.keyvalues || {}
      })

      return {
        hash: result.IpfsHash,
        size: result.PinSize,
      }
    } catch (error) {
      console.error('JSON upload failed:', error)
      throw error
    } finally {
      setUploading(false)
    }
  }

  return {
    uploadFile,
    uploadJSON,
    uploading,
    progress,
  }
}
```

API Service Layer

```
// src/services/api.ts
import axios, { AxiosInstance, AxiosRequestConfig } from 'axios'
import { useAuthStore } from '../stores/authStore'

class APIService {
  private client: AxiosInstance

  constructor() {
    this.client = axios.create({
      baseURL: import.meta.env.VITE_API_BASE_URL || 'http://localhost:8080/api',
      timeout: 30000,
    })

    // Request interceptor to add auth token
    this.client.interceptors.request.use(
      (config) =&gt; {
        const { token } = useAuthStore.getState()
        if (token) {
          config.headers.Authorization = `Bearer ${token}`
        }
        return config
      },
      (error) =&gt; Promise.reject(error)
    )

    // Response interceptor for error handling
    this.client.interceptors.response.use(
      (response) =&gt; response,
      (error) =&gt; {
        if (error.response?.status === 401) {
          useAuthStore.getState().logout()
        }
        return Promise.reject(error)
      }
    )
  }

  // Auth endpoints
  async authenticateWithWallet(walletAddress: string, signature: string, message: string) {
```

```
    const response = await this.client.post('/auth/wallet', {
      wallet_address: walletAddress,
      signature,
      message,
    })
    return response.data
  }

  // Data pool endpoints
  async getDataPools(filters?: {
    type?: string
    minPrice?: string
    maxPrice?: string
    creator?: string
    page?: number
    limit?: number
  }) {
    const response = await this.client.get('/pools', { params: filters })
    return response.data
  }

  async getDataPool(id: string) {
    const response = await this.client.get(`/pools/${id}`)
    return response.data
  }

  async createDataPool(poolData: {
    name: string
    description: string
    data_type: string
    ipfs_hash: string
    metadata_hash: string
    price_per_access: string
  }) {
    const response = await this.client.post('/pools', poolData)
    return response.data
  }

  // Purchase endpoints
  async getPurchases(userId?: string) {
    const response = await this.client.get('/purchases', {
      params: userId ? { user_id: userId } : {}
    })
    return response.data
  }

  async createPurchase(purchaseData: {
    data_pool_id: string
    amount: string
    tx_hash: string
    block_number: number
  }) {
    const response = await this.client.post('/purchases', purchaseData)
    return response.data
  }

  // Analytics endpoints
  async getAnalytics(timeRange: '7d' | '30d' | '90d' = '30d') {
    const response = await this.client.get('/analytics', {
      params: { time_range: timeRange }
    })
    return response.data
  }

  async getUserStats(userId: string) {
    const response = await this.client.get(`/users/${userId}/stats`)
    return response.data
  }
}

export const apiService = new APIService()
```

Data Upload Component

```
// src/components/upload/DataUploadForm.tsx
import React, { useState } from 'react'
import { useIPFS } from '../../hooks/useIPFS'
import { useDataRegistry } from '../../hooks/useContract'
import { apiService } from '../../services/api'
import { toast } from 'react-hot-toast'

interface DataUploadFormProps {
  onSuccess?: (poolId: number) =&gt; void
}

export const DataUploadForm: React.FC&lt;DataUploadFormProps&gt; = ({ onSuccess }) =&gt; {
  const [file, setFile] = useState&lt;File | null&gt;(null)
  const [formData, setFormData] = useState({
    name: '',
    description: '',
    dataType: 'image',
    priceInEth: '0.01',
  })
  const [isSubmitting, setIsSubmitting] = useState(false)

  const { uploadFile, uploading, progress } = useIPFS()
  const { createDataPool, isCreatingPool } = useDataRegistry()

  const handleFileChange = (e: React.ChangeEvent&lt;HTMLInputElement&gt;) =&gt; {
    const selectedFile = e.target.files?.[0]
    if (selectedFile) {
      setFile(selectedFile)
      if (!formData.name) {
        setFormData(prev =&gt; ({ ...prev, name: selectedFile.name }))
      }
    }
  }

  const handleSubmit = async (e: React.FormEvent) =&gt; {
    e.preventDefault()
    if (!file) return

    setIsSubmitting(true)

    try {
      // Step 1: Upload to IPFS
      toast.loading('Uploading to IPFS...', { id: 'upload' })

      const { dataHash, metadataHash } = await uploadFile(file, {
        name: formData.name,
        description: formData.description,
        dataType: formData.dataType,
        originalName: file.name,
        size: file.size,
        uploadedAt: new Date().toISOString(),
      })

      // Step 2: Create pool on blockchain
      toast.loading('Creating data pool on blockchain...', { id: 'upload' })

      const { poolId, txHash } = await createDataPool(
        dataHash,
        metadataHash,
        formData.priceInEth
      )

      // Step 3: Save to backend database
      toast.loading('Saving to database...', { id: 'upload' })

      await apiService.createDataPool({
        name: formData.name,
        description: formData.description,
        data_type: formData.dataType,
```

```
        ipfs_hash: dataHash,
        metadata_hash: metadataHash,
        price_per_access: formData.priceInEth,
      })

      toast.success('Data pool created successfully!', { id: 'upload' })

      // Reset form
      setFile(null)
      setFormData({
        name: '',
        description: '',
        dataType: 'image',
        priceInEth: '0.01',
      })

      onSuccess?.(poolId)
    } catch (error) {
      console.error('Upload failed:', error)
      toast.error('Failed to create data pool. Please try again.', { id: 'upload' })
    } finally {
      setIsSubmitting(false)
    }
  }

  const isLoading = uploading || isCreatingPool || isSubmitting

  return (
    &lt;form onSubmit={handleSubmit} className="space-y-6 bg-gray-800 p-6 rounded-lg"&gt;
      <div>
        &lt;label className="block text-sm font-medium text-gray-300 mb-2"&gt;
          Upload File
        &lt;/label&gt;
        &lt;input
          type="file"
          onChange={handleFileChange}
          accept="image/*,.txt,.csv,.json"
          className="block w-full text-sm text-gray-300 file:mr-4 file:py-2 file:px-4 file:rounded-full f
          disabled={isLoading}
        /&gt;
        {file &amp;&amp; (
          <p>
            Selected: {file.name} ({(file.size / 1024 / 1024).toFixed(2)} MB)
          </p>
        )}
      </div>

      <div>
        &lt;label className="block text-sm font-medium text-gray-300 mb-2"&gt;
          Pool Name
        &lt;/label&gt;
        &lt;input
          type="text"
          value={formData.name}
          onChange={(e) =&gt; setFormData(prev =&gt; ({ ...prev, name: e.target.value }))}
          className="w-full px-3 py-2 bg-gray-700 border border-gray-600 rounded-md text-white focus:outl
          required
          disabled={isLoading}
        /&gt;
      </div>

      <div>
        &lt;label className="block text-sm font-medium text-gray-300 mb-2"&gt;
          Description
        &lt;/label&gt;
        &lt;textarea
          value={formData.description}
          onChange={(e) =&gt; setFormData(prev =&gt; ({ ...prev, description: e.target.value }))}
          rows={3}
          className="w-full px-3 py-2 bg-gray-700 border border-gray-600 rounded-md text-white focus:outl
          disabled={isLoading}
```

```
        /&gt;
      </div>

      <div>
        <div>
          &lt;label className="block text-sm font-medium text-gray-300 mb-2"&gt;
            Data Type
          &lt;/label&gt;
          &lt;select
            value={formData.dataType}
            onChange={(e) =&gt; setFormData(prev =&gt; ({ ...prev, dataType: e.target.value }))}
            className="w-full px-3 py-2 bg-gray-700 border border-gray-600 rounded-md text-white focus:ou
            disabled={isLoading}
          &gt;
            &lt;option value="image"&gt;Image&lt;/option&gt;
            &lt;option value="text"&gt;Text&lt;/option&gt;
            &lt;option value="code"&gt;Code&lt;/option&gt;
            &lt;option value="audio"&gt;Audio&lt;/option&gt;
          &lt;/select&gt;
        </div>

        <div>
          &lt;label className="block text-sm font-medium text-gray-300 mb-2"&gt;
            Price (ETH)
          &lt;/label&gt;
          &lt;input
            type="number"
            step="0.001"
            min="0.001"
            value={formData.priceInEth}
            onChange={(e) =&gt; setFormData(prev =&gt; ({ ...prev, priceInEth: e.target.value }))}
            className="w-full px-3 py-2 bg-gray-700 border border-gray-600 rounded-md text-white focus:ou
            required
            disabled={isLoading}
          /&gt;
        </div>
      </div>

      {uploading &amp;&amp; (
        <div>
          <div>
            <span>Upload Progress</span>
            <span>{progress.progress}%</span>
          </div>
          <div>
            <div>
          </div>
          <p>{progress.stage}</p>
        </div>
      )}

      &lt;button
        type="submit"
        disabled={!file || isLoading}
        className="w-full bg-blue-600 hover:bg-blue-700 disabled:bg-gray-600 disabled:cursor-not-allowed
      &gt;
        {isLoading ? (
          <span>
            &lt;svg className="animate-spin -ml-1 mr-3 h-5 w-5 text-white" xmlns="http://www.w3.org/2000/
              &lt;circle className="opacity-25" cx="12" cy="12" r="10" stroke="currentColor" strokeWidth=
              &lt;path className="opacity-75" fill="currentColor" d="M4 12a8 8 0 018-8V0C5.373 0 0 5.373
            &lt;/svg&gt;
            Creating Pool...
          </span>
        ) : (
          'Create Data Pool'
        )}
      &lt;/button&gt;
    &lt;/form&gt;
  )
}
```

8. Infrastructure & Deployment Plan

Go Backend Deployment (Railway)

Dockerfile

```
# Build stage
FROM golang:1.21-alpine AS builder

WORKDIR /app

# Install dependencies
COPY go.mod go.sum ./
RUN go mod download

# Copy source code
COPY . .

# Build the application
RUN CGO_ENABLED=0 GOOS=linux go build -a -installsuffix cgo -o main ./cmd/server

# Final stage
FROM alpine:latest

RUN apk --no-cache add ca-certificates
WORKDIR /root/

# Copy the binary from builder stage
COPY --from=builder /app/main .

# Expose port
EXPOSE 8080

# Run the application
CMD ["./main"]
```

Railway Configuration

```
// railway.json
{
  "build": {
    "builder": "DOCKERFILE",
    "dockerfilePath": "Dockerfile"
  },
  "deploy": {
    "startCommand": "./main",
    "healthcheckPath": "/health",
    "healthcheckTimeout": 100
  }
}
```

Frontend Deployment (Vercel)

Vite Configuration

```
// vite.config.ts
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'
import { nodePolyfills } from 'vite-plugin-node-polyfills'

export default defineConfig({
  plugins: [
    react(),
    nodePolyfills({
```

```
      // Whether to polyfill `node:` protocol imports.
      protocolImports: true,
    }),
  ],
  define: {
    global: 'globalThis',
  },
  resolve: {
    alias: {
      '@': '/src',
    },
  },
  build: {
    rollupOptions: {
      external: [],
    },
  },
})
```

Vercel Configuration

```
// vercel.json
{
  "framework": "vite",
  "buildCommand": "npm run build",
  "devCommand": "npm run dev",
  "installCommand": "npm install",
  "outputDirectory": "dist",
  "env": {
    "VITE_CHAIN_ID": "137",
    "VITE_DATA_REGISTRY_ADDRESS": "@data_registry_address",
    "VITE_ALCHEMY_API_KEY": "@alchemy_api_key",
    "VITE_API_BASE_URL": "@api_base_url"
  }
}
```

PostgreSQL Database Setup

Migration Scripts

```
-- migrations/001_initial_schema.up.sql
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";

CREATE TABLE users (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    wallet_addr VARCHAR(42) UNIQUE NOT NULL,
    username VARCHAR(50) UNIQUE,
    email VARCHAR(255) UNIQUE,
    role VARCHAR(20) DEFAULT 'contributor',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE data_pools (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    blockchain_id BIGINT UNIQUE NOT NULL,
    creator_id UUID NOT NULL REFERENCES users(id),
    ipfs_hash VARCHAR(64) NOT NULL,
    metadata_hash VARCHAR(64),
    name VARCHAR(255) NOT NULL,
    description TEXT,
    data_type VARCHAR(50) NOT NULL,
    price_per_access DECIMAL(78,0) NOT NULL, -- Wei amount
    quality_score DECIMAL(3,2) DEFAULT 0,
    total_purchases BIGINT DEFAULT 0,
    is_active BOOLEAN DEFAULT true,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
```

```
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE purchases (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    data_pool_id UUID NOT NULL REFERENCES data_pools(id),
    buyer_id UUID NOT NULL REFERENCES users(id),
    amount DECIMAL(78,0) NOT NULL,
    tx_hash VARCHAR(66) UNIQUE NOT NULL,
    block_number BIGINT NOT NULL,
    access_token VARCHAR(255) UNIQUE,
    expires_at TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE contributions (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    data_pool_id UUID NOT NULL REFERENCES data_pools(id),
    contributor_id UUID NOT NULL REFERENCES users(id),
    share_percent DECIMAL(5,2) NOT NULL,
    total_earned DECIMAL(78,0) DEFAULT 0,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Indexes for performance
CREATE INDEX idx_users_wallet ON users(wallet_addr);
CREATE INDEX idx_data_pools_creator ON data_pools(creator_id);
CREATE INDEX idx_data_pools_type ON data_pools(data_type);
CREATE INDEX idx_purchases_pool ON purchases(data_pool_id);
CREATE INDEX idx_purchases_buyer ON purchases(buyer_id);
CREATE INDEX idx_contributions_pool ON contributions(data_pool_id);
```

Environment Configuration

Backend Environment Variables

```
# .env
# Database
DATABASE_URL=postgresql://user:password@localhost:5432/synapse_ledger
REDIS_URL=redis://localhost:6379

# Blockchain
RPC_URL=https://polygon-mainnet.g.alchemy.com/v2/YOUR_KEY
MUMBAI_RPC_URL=https://polygon-mumbai.g.alchemy.com/v2/YOUR_KEY
PRIVATE_KEY=your_private_key_here
DATA_REGISTRY_ADDRESS=0x...

# IPFS
PINATA_API_KEY=your_pinata_api_key
PINATA_SECRET_API_KEY=your_pinata_secret

# Auth
JWT_SECRET=your_jwt_secret_here
JWT_EXPIRES_IN=24h

# Server
PORT=8080
GIN_MODE=release
```

Frontend Environment Variables

```
# .env
VITE_CHAIN_ID=137
VITE_DATA_REGISTRY_ADDRESS=0x...
VITE_ALCHEMY_API_KEY=your_alchemy_key
VITE_PINATA_API_KEY=your_pinata_key
VITE_API_BASE_URL=https://your-api.railway.app/api
```

9. Security, Testing & Monitoring

Go Backend Testing

```
// internal/api/handlers/pools_test.go
package handlers

import (
    "bytes"
    "encoding/json"
    "net/http"
    "net/http/httptest"
    "testing"

    "github.com/gin-gonic/gin"
    "github.com/stretchr/testify/assert"
    "github.com/stretchr/testify/mock"
    "synapse-ledger/internal/models"
)

func TestCreateDataPool(t *testing.T) {
    // Setup test database
    db := setupTestDB()
    defer teardownTestDB(db)

    // Create test user
    user := &amp;models.User{
        WalletAddr: "0x1234567890123456789012345678901234567890",
        Role:       "contributor",
    }
    db.Create(user)

    // Setup router
    router := gin.New()
    handler := NewPoolHandler(db, nil)
    router.POST("/pools", handler.CreatePool)

    // Test data
    poolData := map[string]interface{}{
        "name":             "Test Pool",
        "description":      "Test description",
        "data_type":        "image",
        "ipfs_hash":        "QmTest123",
        "metadata_hash":    "QmMeta456",
        "price_per_access": "1000000000000000000", // 1 ETH in Wei
    }

    jsonData, _ := json.Marshal(poolData)

    // Create request
    req, _ := http.NewRequest("POST", "/pools", bytes.NewBuffer(jsonData))
    req.Header.Set("Content-Type", "application/json")
    req.Header.Set("Authorization", "Bearer "+generateTestJWT(user.ID))

    // Record response
    w := httptest.NewRecorder()
    router.ServeHTTP(w, req)

    // Assertions
```

```
    assert.Equal(t, http.StatusCreated, w.Code)

    var response map[string]interface{}
    err := json.Unmarshal(w.Body.Bytes(), &amp;response)
    assert.NoError(t, err)
    assert.Equal(t, "Test Pool", response["name"])

    // Verify database record
    var pool models.DataPool
    err = db.Where("name = ?", "Test Pool").First(&amp;pool).Error
    assert.NoError(t, err)
    assert.Equal(t, user.ID, pool.CreatorID)
}

func TestGetDataPools(t *testing.T) {
    // Setup test database with sample data
    db := setupTestDB()
    defer teardownTestDB(db)

    // Create test pools
    user := createTestUser(db)
    createTestPools(db, user.ID, 5)

    // Setup router
    router := gin.New()
    handler := NewPoolHandler(db, nil)
    router.GET("/pools", handler.GetPools)

    // Test request
    req, _ := http.NewRequest("GET", "/pools?page=1&amp;limit=10", nil)
    w := httptest.NewRecorder()
    router.ServeHTTP(w, req)

    // Assertions
    assert.Equal(t, http.StatusOK, w.Code)

    var response map[string]interface{}
    err := json.Unmarshal(w.Body.Bytes(), &amp;response)
    assert.NoError(t, err)

    data := response["data"].([]interface{})
    assert.Len(t, data, 5)
}
```

React Component Testing

```
// src/components/__tests__/DataUploadForm.test.tsx
import { render, screen, fireEvent, waitFor } from '@testing-library/react'
import { vi } from 'vitest'
import { DataUploadForm } from '../upload/DataUploadForm'
import { useIPFS } from '../../hooks/useIPFS'
import { useDataRegistry } from '../../hooks/useContract'

// Mock hooks
vi.mock('../../hooks/useIPFS')
vi.mock('../../hooks/useContract')

const mockUseIPFS = useIPFS as vi.MockedFunction&lt;typeof useIPFS&gt;
const mockUseDataRegistry = useDataRegistry as vi.MockedFunction&lt;typeof useDataRegistry&gt;

describe('DataUploadForm', () =&gt; {
  beforeEach(() =&gt; {
    mockUseIPFS.mockReturnValue({
      uploadFile: vi.fn(),
      uploading: false,
      progress: { progress: 0, stage: 'preparing' },
    })

    mockUseDataRegistry.mockReturnValue({
      createDataPool: vi.fn(),
```

```
    isCreatingPool: false,
    poolCount: 0,
    userPools: [],
    purchaseDataAccess: vi.fn(),
    isPurchasing: false,
  })
})

test('renders upload form correctly', () =&gt; {
  render(&lt;DataUploadForm /&gt;)

  expect(screen.getByLabelText(/upload file/i)).toBeInTheDocument()
  expect(screen.getByLabelText(/pool name/i)).toBeInTheDocument()
  expect(screen.getByLabelText(/description/i)).toBeInTheDocument()
  expect(screen.getByLabelText(/data type/i)).toBeInTheDocument()
  expect(screen.getByLabelText(/price/i)).toBeInTheDocument()
})

test('handles file selection', () =&gt; {
  render(&lt;DataUploadForm /&gt;)

  const fileInput = screen.getByLabelText(/upload file/i) as HTMLInputElement
  const file = new File(['test content'], 'test.jpg', { type: 'image/jpeg' })

  fireEvent.change(fileInput, { target: { files: [file] } })

  expect(fileInput.files?.[0]).toBe(file)
  expect(screen.getByText(/selected: test.jpg/i)).toBeInTheDocument()
})

test('submits form with valid data', async () =&gt; {
  const mockUploadFile = vi.fn().mockResolvedValue({
    dataHash: 'QmTest123',
    metadataHash: 'QmMeta456',
  })

  const mockCreateDataPool = vi.fn().mockResolvedValue({
    poolId: 1,
    txHash: '0xtest',
  })

  mockUseIPFS.mockReturnValue({
    uploadFile: mockUploadFile,
    uploading: false,
    progress: { progress: 0, stage: 'preparing' },
  })

  mockUseDataRegistry.mockReturnValue({
    createDataPool: mockCreateDataPool,
    isCreatingPool: false,
    poolCount: 0,
    userPools: [],
    purchaseDataAccess: vi.fn(),
    isPurchasing: false,
  })

  render(&lt;DataUploadForm /&gt;)

  // Fill form
  const fileInput = screen.getByLabelText(/upload file/i)
  const nameInput = screen.getByLabelText(/pool name/i)
  const submitButton = screen.getByRole('button', { name: /create data pool/i })

  const file = new File(['test'], 'test.jpg', { type: 'image/jpeg' })
  fireEvent.change(fileInput, { target: { files: [file] } })
  fireEvent.change(nameInput, { target: { value: 'Test Pool' } })

  // Submit form
  fireEvent.click(submitButton)

  await waitFor(() =&gt; {
```

```
      expect(mockUploadFile).toHaveBeenCalledWith(
        file,
        expect.objectContaining({
          name: 'Test Pool',
        })
      )
    })

    await waitFor(() =&gt; {
      expect(mockCreateDataPool).toHaveBeenCalledWith(
        'QmTest123',
        'QmMeta456',
        '0.01'
      )
    })
  })
})
```

Security Configuration

Go Security Middleware

```
// internal/api/middleware/security.go
package middleware

import (
    "net/http"
    "time"

    "github.com/gin-contrib/cors"
    "github.com/gin-contrib/secure"
    "github.com/gin-gonic/gin"
)

func SecurityHeaders() gin.HandlerFunc {
    return secure.New(secure.Config{
        SSLRedirect:          true,
        SSLHost:              "your-domain.com",
        STSSeconds:           31536000,
        STSIncludeSubdomains: true,
        FrameDeny:            true,
        ContentTypeNosniff:   true,
        BrowserXssFilter:     true,
        IENoOpen:            true,
        ReferrerPolicy:      "strict-origin-when-cross-origin",
    })
}

func CORS() gin.HandlerFunc {
    return cors.New(cors.Config{
        AllowOrigins:     []string{"https://your-frontend.vercel.app"},
        AllowMethods:     []string{"GET", "POST", "PUT", "DELETE", "OPTIONS"},
        AllowHeaders:     []string{"Origin", "Content-Type", "Authorization"},
        ExposeHeaders:    []string{"Content-Length"},
        AllowCredentials: true,
        MaxAge:          12 * time.Hour,
    })
}

func RateLimit() gin.HandlerFunc {
    // Implementation with go-rate or similar library
    return gin.HandlerFunc(func(c *gin.Context) {
        // Rate limiting logic
        c.Next()
    })
}
```

10. Build Responsibility Map

Smart Contracts & Blockchain

Component	Description	Recommended Approach	Reason
DataRegistry.sol	Core marketplace contract with pool management	**Code Yourself**	Critical for understanding business logic and Solidity mastery
RoyaltyDistributor.sol	Automated payment distribution logic	**Code Yourself**	Complex financial logic requires careful manual implementation
SynapseToken.sol	ERC-20 utility token with staking	⚙ **Hybrid**	Use OpenZeppelin template, customize staking logic manually
DataNFT.sol	ERC-721 for data ownership certificates	**Let AI Handle**	Standard NFT implementation with minimal customization
Hardhat Tests	Comprehensive unit and integration tests	**Code Yourself**	Essential for understanding contract behavior and security
Deployment Scripts	Mumbai and Polygon deployment automation	⚙ **Hybrid**	AI generate template, customize for specific deployment needs

Go Backend Development

Component	Description	Recommended Approach	Reason
Database Models (GORM)	User, DataPool, Purchase, Contribution structs	**Code Yourself**	Important for understanding Go structs and GORM relationships
Blockchain Event Listener	Smart contract event monitoring with go-ethereum	**Code Yourself**	Critical for understanding Web3 backend integration
Authentication Middleware	JWT and Web3 signature verification	**Code Yourself**	Security-critical component requiring manual implementation
IPFS Service (Pinata)	File upload and pinning service integration	**Code Yourself**	Learn IPFS concepts and HTTP client patterns in Go
API Handlers	CRUD operations for pools, purchases, users	⚙ **Hybrid**	AI generate basic CRUD, manually implement business logic
Database Migrations	PostgreSQL schema and indexes	**Let AI Handle**	Standard SQL with minimal customization needed
Background Jobs	Royalty calculation and distribution workers	**Code Yourself**	Complex business logic with Go routines and channels
Configuration Management	Environment variables and application config	**Let AI Handle**	Standard configuration patterns

React Frontend Development

Component	Description	Recommended Approach	Reason
Contract Integration Hooks	Wagmi-based hooks for contract interactions	**Code Yourself**	Critical for mastering Web3 frontend integration
IPFS Upload Hook	File upload with progress tracking	**Code Yourself**	Important for understanding async operations and state management
Data Upload Form	Complex form with file upload and Web3 integration	**Code Yourself**	Combines multiple concepts: forms, file handling, Web3 calls
Wallet Connection Logic	MetaMask and WalletConnect integration	⚙ **Hybrid**	Use RainbowKit base, customize connection flow
API Service Layer	Axios-based HTTP client with interceptors	**Code Yourself**	Important for understanding API integration patterns
Marketplace Components	Browse and filter data pools interface	**Let AI Handle**	Standard CRUD interface with filtering
Landing Page	Marketing and informational content	**Let AI Handle**	Static content with minimal business logic
State Management	Zustand stores for global state	⚙ **Hybrid**	AI generate structure, manually implement business logic

Testing & DevOps

Component	Description	Recommended Approach	Reason
Go Unit Tests	Handler and service layer testing	**Code Yourself**	Important for understanding Go testing patterns
React Component Tests	Frontend component testing with Vitest	⚙ **Hybrid**	AI generate test structure, manually implement Web3 scenarios
Smart Contract Tests	Hardhat/Foundry contract testing	**Code Yourself**	Essential for contract security and understanding
Docker Configuration	Go backend containerization	**Let AI Handle**	Standard Docker patterns for Go applications
CI/CD Pipelines	GitHub Actions for deployment	**Let AI Handle**	Standard deployment automation
Monitoring Setup	Application monitoring and alerting	**Let AI Handle**	Standard monitoring configurations

11. Suggested AI Prompts for Each AI-Coded Part

Go Backend Components

Database Migrations

```
Create PostgreSQL migration scripts for a Web3 data marketplace:
- Tables: users, data_pools, purchases, contributions
- Use UUID primary keys with gen_random_uuid()
- Include proper foreign key relationships
- Add indexes for performance on wallet addresses, pool IDs, and timestamps
- Include both up and down migration files
- Use decimal(78,0) for Wei amounts to handle large numbers
- Add created_at and updated_at timestamps with defaults
```

Docker Configuration

```
Create a production-ready Dockerfile for a Go application:
- Use multi-stage build with golang:1.21-alpine as builder
- Install ca-certificates in final stage
- Copy binary from builder stage
- Use non-root user for security
- Expose port 8080
- Include healthcheck endpoint
- Optimize for minimal image size
- Add proper labels and metadata
```

Configuration Management

```
Generate Go configuration management using Viper:
- Support environment variables and config files
- Include database, blockchain, IPFS, and auth configs
- Validation for required fields
- Default values for development
- Struct tags for JSON and env mapping
- Load from .env file and environment variables
- Type-safe configuration with proper error handling
```

React Frontend Components

Landing Page

```
Create a modern landing page for "Synapse Ledger" Web3 data marketplace:
- Hero section: "Monetize Your Data Ethically While Powering AI Innovation"
- Features: Data ownership, Fair compensation, Privacy preservation, Transparent transactions
- How it works: 3 steps with icons - Upload → Stake → Earn
- Stats section with animated counters (contributors, data pools, total earned)
- CTA buttons: "Start Contributing" and "Browse Data"
- Dark theme with blue (#3B82F6) and purple (#8B5CF6) accents
- Responsive design with Tailwind CSS
- React 18 with TypeScript
- Smooth scroll animations
```

Marketplace Filter Components

```
Create React marketplace filtering components:
- SearchBar with debounced input (500ms delay)
- FilterSidebar with: data type checkboxes, price range slider, quality score filter, date range picker
- SortDropdown: price (low/high), quality score, newest, most popular
- ResultsHeader showing count and active filters
- Clear filters button
- TypeScript interfaces for all filter states
- Tailwind CSS dark theme styling
```

```
- State management with useState and useEffect
- Export FilterState interface and default values
```

Responsive Layout Template

```
Generate a responsive layout for Web3 dApp:
- Header: logo, nav menu (Dashboard, Marketplace, Upload, Profile), wallet connect button
- Sidebar: collapsible navigation for dashboard pages
- Main content area with proper container and spacing
- Footer with links and social media icons
- Mobile hamburger menu with slide-out navigation
- Dark/light theme toggle (store preference in localStorage)
- Use Tailwind CSS with responsive breakpoints
- React 18 with TypeScript
- Smooth transitions and animations
```

DevOps & Infrastructure

GitHub Actions Workflows

```
Create GitHub Actions workflows for Go + React + Solidity project:

Workflow 1 - Smart Contracts:
- Trigger: push to main, pull requests
- Install Node.js and dependencies
- Run Hardhat tests with coverage
- Deploy to Mumbai testnet on main branch pushes
- Verify contracts on PolygonScan
- Store deployment artifacts

Workflow 2 - Go Backend:
- Trigger: changes in backend/ directory
- Setup Go 1.21, PostgreSQL for testing
- Run tests with coverage reporting
- Build and deploy to Railway on main branch
- Run database migrations
- Health check after deployment

Workflow 3 - React Frontend:
- Trigger: changes in frontend/ directory
- Setup Node.js, install dependencies
- Run tests and build optimized bundle
- Deploy to Vercel on main branch
- Update environment variables from secrets
```

Monitoring Configuration

```
Create comprehensive monitoring setup for Web3 dApp:
- Application monitoring: structured logging with logrus
- Health check endpoints for database and external services
- Metrics collection: HTTP request duration, database query times, blockchain RPC calls
- Error tracking: capture and categorize errors by severity
- Performance monitoring: memory usage, goroutine counts, response times
- Blockchain monitoring: contract event processing delays, gas price tracking
- Alert configuration: Slack/email notifications for critical errors
- Dashboard configuration for Grafana with key business metrics
- Privacy-compliant analytics without PII
```

Environment Configuration

```
Create environment configuration templates:
- .env.example with all required variables and descriptions
- Separate configs for development, staging, production
- Variables needed:
  - Database: PostgreSQL connection string, pool settings
  - Blockchain: RPC URLs for Polygon/Mumbai, private key, contract addresses
  - IPFS: Pinata API keys and gateway URLs
  - Authentication: JWT secret and expiration
  - External APIs: Alchemy, analytics services
  - Server: port, cors origins, rate limiting
- Validation schema with required/optional fields
- Security best practices documentation
- Docker Compose for local development
```

12. Development Phases & Milestones

Phase 1: Smart Contract MVP (3-4 weeks)

Focus: Core blockchain functionality

Week 1-2: Contract Development

- DataRegistry.sol with pool creation and purchase functions
- RoyaltyDistributor.sol for automated payment distribution
- Basic SynapseToken.sol (ERC-20) implementation
- Comprehensive unit tests with Hardhat

Week 3-4: Testing & Deployment

- Integration tests between contracts
- Gas optimization and security review
- Deploy to Mumbai testnet
- Contract verification on PolygonScan

Deliverables:

- ✅ Deployed and verified smart contracts on testnet
- ✅ >95% test coverage on all contract functions
- ✅ Security analysis with Slither
- ✅ Deployment scripts and documentation

Phase 2: Go Backend Development (3-4 weeks)

Focus: API and blockchain integration

Week 1-2: Core Backend Setup

- Go project structure with Gin framework
- PostgreSQL database setup with GORM
- User authentication with JWT and Web3 signatures
- Basic CRUD APIs for users and data pools

Week 3-4: Blockchain Integration

- Smart contract event listener with go-ethereum
- IPFS integration with Pinata Go SDK
- Background job processing for royalty distribution

- Comprehensive API testing

Deliverables:

- ✅ Complete REST API deployed on Railway
- ✅ Real-time blockchain event processing
- ✅ IPFS file upload and management
- ✅ Authentication and authorization system

Phase 3: React Frontend Integration (3-4 weeks)

Focus: User interface and Web3 connection

Week 1-2: Core Frontend Setup

- React 18 + TypeScript + Vite project setup
- Tailwind CSS configuration and design system
- Wagmi v2 + RainbowKit wallet integration
- Basic routing and layout components

Week 3-4: Feature Implementation

- Data upload form with IPFS integration
- Marketplace browsing and filtering interface
- Purchase flow with MetaMask integration
- Real-time updates from blockchain events

Deliverables:

- ✅ Complete frontend application deployed on Vercel
- ✅ Wallet integration with multiple providers
- ✅ Working data upload and marketplace features
- ✅ Responsive design with mobile support

Phase 4: Testing & Production Deployment (2-3 weeks)

Focus: Production readiness and security

Week 1-2: Comprehensive Testing

- End-to-end testing with Playwright
- Load testing for high traffic scenarios
- Security audit preparation
- Bug fixes and performance optimizations

Week 2-3: Production Launch

- Mainnet smart contract deployment
- Production environment configuration
- Monitoring and analytics setup
- Documentation and user guides

Deliverables:

- ✅ Production application on Polygon mainnet
- ✅ Monitoring and alerting systems
- ✅ Security audit completed
- ✅ User documentation and support

13. Future Enhancements & Roadmap

Phase 5: Advanced Features (3-6 months post-launch)

Multi-Chain Support

- **Ethereum L2 Integration**: Deploy contracts on Arbitrum and Optimism
- **Cross-Chain Bridge**: Token bridging with LayerZero or similar
- **Unified API**: Single backend handling multiple blockchain networks
- **Gas Optimization**: Dynamic network selection based on fees

Advanced Privacy Features

- **Zero-Knowledge Proofs**: zk-SNARKs for contribution verification without revealing data
- **Federated Learning Support**: APIs for on-device model training
- **Homomorphic Encryption**: Computation on encrypted datasets
- **Privacy Metrics**: Quantifiable privacy scores for each dataset

Enterprise Features

- **API Gateway**: Rate-limited enterprise API access
- **Bulk Operations**: Efficient batch processing for large datasets
- **SLA Guarantees**: Service level agreements for data availability
- **Custom Smart Contracts**: Template system for enterprise-specific contracts

Phase 6: DAO Governance (6-12 months post-launch)

Token Governance

- **Governance Extensions**: Extend SynapseToken with voting capabilities
- **Proposal System**: Community proposals for platform parameters
- **Treasury Management**: DAO-controlled development fund
- **Delegation**: Allow token holders to delegate voting power

Community Features

- **Data Quality DAO**: Community-driven data verification and scoring
- **Dispute Resolution**: Decentralized arbitration for data quality disputes
- **Research Grants**: DAO funding for ethical AI research
- **Developer Incentives**: Grant program for ecosystem development

Long-term Vision (1-2 years)

Industry Standards

- **Data Provenance Standards**: Contribute to industry-wide ethical AI standards
- **Regulatory Compliance**: Proactive engagement with data protection authorities
- **Academic Partnerships**: Collaborate with universities on AI ethics research
- **Open Source Initiative**: Gradual open-sourcing of platform components

Mobile & IoT Expansion

- **React Native App**: Cross-platform mobile application
- **IoT Integration**: Direct data contribution from IoT devices
- **Edge Computing**: On-device processing for enhanced privacy
- **Offline Capabilities**: Data preparation without constant connectivity

This comprehensive blueprint provides you with a clear development roadmap that leverages your Go backend expertise while building critical Web3 integration skills. The Build Responsibility Map ensures you focus on the most valuable learning opportunities while using AI assistance to accelerate routine development tasks.

Focus your manual coding efforts on the blockchain event listener, contract integration hooks, and IPFS upload logic - these are the core Web3 integration patterns that will strengthen your full-stack blockchain development skills.