Web3 Global Tax System - Architecture & Development Roadmap

🏗️ System Overview

A privacy-first Web3 application that revolutionizes tax compliance by allowing users to upload encrypted invoices and generate Zero-Knowledge Proofs for tax verification without revealing sensitive transaction details.

🎯 Core Principles

- **Privacy First**: Raw invoice data never exposed to government or third parties
- **Zero-Knowledge**: Mathematical proofs without revealing underlying data
- **Decentralized**: Blockchain-based verification and compliance
- **User-Friendly**: Hybrid authentication for mass adoption

🏛️ Microservices Architecture

1. API Gateway Service

- **Tech Stack**: NestJS, Express Gateway
- **Responsibilities**:
  - Route requests to appropriate microservices
  - Rate limiting and authentication middleware
  - CORS handling for React frontend
  - Request/response logging
- **Port**: 3000

2. User Management Service

- **Tech Stack**: NestJS, PostgreSQL, Passport.js
- **Responsibilities**:
  - Hybrid authentication (Google OAuth + Wallet)
  - User profile management
  - Wallet assignment for Google-authenticated users
  - Session management
- **Database**: User profiles, wallet mappings, auth tokens

- **Port**: 3001

3. **Invoice Management Service**

- **Tech Stack**: NestJS, MongoDB, Multer, Crypto-JS
- **Responsibilities**:
  - Invoice file upload and validation
  - Client-side encryption key management
  - IPFS integration for encrypted storage
  - Invoice metadata indexing
- **Database**: Invoice metadata, encryption hashes, IPFS references
- **Port**: 3002

4. **ZK Proof Service**

- **Tech Stack**: NestJS, Circom, SnarkJS
- **Responsibilities**:
  - ZK circuit management
  - Proof generation coordination (client-side heavy lifting)
  - Proof validation before blockchain submission
  - Witness generation utilities
- **Port**: 3003

5. **Government Dashboard Service**

- **Tech Stack**: NestJS, PostgreSQL, Redis (caching)
- **Responsibilities**:
  - Taxpayer compliance monitoring
  - Verification status management
  - Government entity permissions
  - Audit trail logging
- **Database**: Compliance records, audit logs, government entity profiles
- **Port**: 3004

6. **Blockchain Integration Service**

- **Tech Stack**: NestJS, MultiversX SDK (@multiversx/sdk-js)

- **Responsibilities**:
  - Smart contract interactions
  - Transaction monitoring
  - Wallet integration
  - On-chain data synchronization
- **Port**: 3005

7. **Business Categorization Service** (Phase 3)

- **Tech Stack**: NestJS, PostgreSQL, Machine Learning libs
- **Responsibilities**:
  - Business size classification
  - Tax bracket determination
  - Revenue analysis
  - Compliance scoring
- **Port**: 3006

💻 Frontend Architecture

React + TypeScript Application

- **Tech Stack**: React 18, TypeScript, Vite, TailwindCSS
- **Key Libraries**:
  - `@multiversx/sdk-dapp` - MultiversX wallet integration
  - `@multiversx/sdk-web-wallet-provider` - Web wallet support
  - `axios` - API communication
  - `react-query` - Server state management
  - `zustand` - Client state management
  - `react-hook-form` - Form management
  - `crypto-js` - Client-side encryption
  - `snarkjs` - ZK proof generation

Component Structure

```
src/
├── components/
│   ├── common/        # Reusable UI components
│   ├── auth/          # Authentication components
│   ├── invoices/      # Invoice management
│   ├── dashboard/     # User dashboard
│   ├── government/    # Government portal
│   └── zk-proof/      # ZK proof components
├── hooks/             # Custom React hooks
├── services/          # API service layer
├── store/             # Zustand stores
├── utils/             # Utility functions
└── types/             # TypeScript definitions
```

🔗 Smart Contracts (MultiversX)

Tax Compliance Contract

```
rust
```

```rust
// Core contract structure
#[multiversx_sc::contract]
pub trait TaxComplianceContract {
  // Storage
  #[storage_mapper("taxpayer_status")]
  fn taxpayer_status(&self) -> SingleValueMapper<TaxpayerStatus>;

  #[storage_mapper("proof_registry")]
  fn proof_registry(&self) -> MapMapper<ManagedAddress, ProofData>;

  // Endpoints
  #[endpoint]
  fn submit_zk_proof(&self, proof: ZKProof, public_signals: Vec<u64>);

  #[endpoint]
  fn verify_compliance(&self, taxpayer: ManagedAddress) -> ComplianceStatus;

  #[endpoint]
  fn calculate_tax_owed(&self, taxpayer: ManagedAddress) -> BigUint;

  #[endpoint]
  fn pay_taxes(&self) #[payable("EGLD")];
}
```

On-Chain Data Structures

rust

```
#[derive(TypeAbi, TopEncode, TopDecode, PartialEq, Clone)]
pub struct ProofData {
    pub proof_hash: ManagedBuffer,
    pub public_signals: ManagedVec<u64>,
    pub timestamp: u64,
    pub verification_status: VerificationStatus,
}

#[derive(TypeAbi, TopEncode, TopDecode, PartialEq, Clone)]
pub enum ComplianceStatus {
    Valid,
    Invalid,
    Pending,
    Overdue,
}
```

🔐 Privacy-First Data Flow

Invoice Upload & Encryption Flow

1. User selects invoice files in React app
2. Client-side encryption using user's wallet-derived key
3. Encrypted data uploaded to Invoice Management Service
4. Service stores encrypted data on IPFS
5. Invoice metadata stored in MongoDB (no raw data)
6. User can decrypt and view own invoices locally

ZK Proof Generation Flow

1. User requests proof generation for specific period
2. Client downloads encrypted invoices from IPFS
3. Client-side decryption using wallet key
4. ZK circuit calculates total tax owed without revealing details
5. Proof submitted to blockchain via Blockchain Service
6. Raw invoice data never leaves user's control

Government Verification Flow

1. Government entity accesses dashboard

2. Queries blockchain for taxpayer compliance status
3. Views aggregated compliance data (Valid/Invalid/Pending)
4. Cannot access raw invoice data or transaction details
5. Can verify mathematical correctness of tax calculations

🗄️ Database Schemas

User Management Database (PostgreSQL)

```sql
-- Users table
CREATE TABLE users (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  email VARCHAR(255) UNIQUE,
  google_id VARCHAR(255) UNIQUE,
  wallet_address VARCHAR(62) UNIQUE,
  created_at TIMESTAMP DEFAULT NOW(),
  updated_at TIMESTAMP DEFAULT NOW()
);

-- Wallet assignments for Google auth users
CREATE TABLE wallet_assignments (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  user_id UUID REFERENCES users(id),
  wallet_address VARCHAR(62) UNIQUE,
  private_key_encrypted TEXT, -- Encrypted with user's Google-derived key
  created_at TIMESTAMP DEFAULT NOW()
);
```

Invoice Metadata Database (MongoDB)

```javascript
```

```
// Invoice collection schema
{
 _id: ObjectId,
 userId: String, // User UUID or wallet address
 originalFilename: String,
 encryptedFilename: String,
 ipfsHash: String,
 fileSize: Number,
 uploadDate: Date,
 encryptionHash: String, // Hash of encryption key for verification
 metadata: {
  invoiceDate: Date,
  taxYear: Number,
  category: String // business, personal, etc.
 }
}
```

Government Dashboard Database (PostgreSQL)

```sql
-- Government entities
CREATE TABLE government_entities (
   id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
   name VARCHAR(255) NOT NULL,
   jurisdiction VARCHAR(100),
   public_key VARCHAR(255),
   permissions JSONB,
   created_at TIMESTAMP DEFAULT NOW()
);

-- Compliance monitoring
CREATE TABLE compliance_records (
   id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
   taxpayer_address VARCHAR(62),
   tax_year INTEGER,
   status compliance_status_enum,
   proof_hash VARCHAR(66),
   verified_at TIMESTAMP,
   government_entity_id UUID REFERENCES government_entities(id)
);
```

🚀 Development Timeline (320 Hours / 8 Months)

Phase 1: Foundation & MVP (Months 1-3, ~120 hours)

Month 1 (40 hours)

- [ ] **Week 1-2**: Setup development environment
  - Docker containerization for all services
  - Database setup (PostgreSQL + MongoDB)
  - Basic NestJS microservices skeleton
  - API Gateway configuration
- [ ] **Week 3-4**: User Management Service
  - Google OAuth integration
  - Basic wallet assignment
  - User registration/login flows
  - JWT token management

Month 2 (40 hours)

- [ ] **Week 1-2**: Invoice Management Service
  - File upload functionality
  - Basic client-side encryption
  - IPFS integration setup
  - MongoDB schema implementation
- [ ] **Week 3-4**: React Frontend Foundation
  - Project setup with Vite + TypeScript
  - Basic routing structure
  - Authentication components
  - Invoice upload interface

Month 3 (40 hours)

- [ ] **Week 1-2**: MultiversX Integration
  - Wallet connection in React
  - Basic smart contract deployment
  - Blockchain Integration Service
  - Simple transaction handling

- [ ] **Week 3-4**: ZK Proof MVP
  - Basic Circom circuit for tax calculation
  - Client-side proof generation
  - Smart contract proof verification
  - End-to-end testing

Phase 2: Core Features (Months 4-6, ~120 hours)

Month 4 (40 hours)

- [ ] **Week 1-2**: Government Dashboard Service
  - Government entity management
  - Basic compliance monitoring
  - Permission system
  - Admin interface components
- [ ] **Week 3-4**: Enhanced ZK Proofs
  - Complex tax calculation circuits
  - Multiple invoice aggregation
  - Proof optimization
  - Error handling

Month 5 (40 hours)

- [ ] **Week 1-2**: Smart Contract Enhancement
  - Automated tax calculation
  - Compliance status management
  - Tax payment functionality
  - Event emission for frontend
- [ ] **Week 3-4**: Frontend Enhancement
  - Dashboard with invoice history
  - ZK proof generation UI
  - Government verification interface
  - Responsive design

Month 6 (40 hours)

- [ ] **Week 1-2**: Privacy & Security

- Advanced encryption implementation
- Key management improvements
- Security audit preparation
- IPFS optimization

- [ ] **Week 3-4**: Integration Testing
  - End-to-end testing
  - Service integration validation
  - Performance optimization
  - Bug fixes

Phase 3: Advanced Features & Polish (Months 7-8, ~80 hours)

Month 7 (40 hours)

- [ ] **Week 1-2**: Business Categorization Service
  - Machine learning model for business classification
  - Dynamic tax bracket assignment
  - Revenue analysis algorithms
  - Integration with main system
- [ ] **Week 3-4**: Advanced Dashboard Features
  - Analytics and reporting
  - Multi-year tax history
  - Compliance trends
  - Export functionality

Month 8 (40 hours)

- [ ] **Week 1-2**: Final Polish
  - UI/UX improvements
  - Performance optimization
  - Documentation completion
  - Deployment preparation
- [ ] **Week 3-4**: Thesis Preparation
  - Demo preparation
  - Documentation finalization
  - Presentation materials

- Final testing and validation

🐳 Docker Configuration

docker-compose.yml

```
yaml
```

```
version: '3.8'

services:
  # Databases
  postgres:
    image: postgres:15
    environment:
      POSTGRES_DB: tax_system
      POSTGRES_USER: admin
      POSTGRES_PASSWORD: password
    volumes:
      - postgres_data:/var/lib/postgresql/data
    ports:
      - "5432:5432"

  mongodb:
    image: mongo:6
    environment:
      MONGO_INITDB_DATABASE: invoices
    volumes:
      - mongo_data:/data/db
    ports:
      - "27017:27017"

  redis:
    image: redis:7-alpine
    ports:
      - "6379:6379"

  # Microservices
  api-gateway:
    build: ./services/api-gateway
    ports:
      - "3000:3000"
    depends_on:
      - user-service
      - invoice-service
    environment:
      - NODE_ENV=development

  user-service:
    build: ./services/user-management
    ports:
```

```yaml
      - "3001:3001"
    depends_on:
      - postgres
    environment:
      - DATABASE_URL=postgresql://admin:password@postgres:5432/tax_system
      - GOOGLE_CLIENT_ID=${GOOGLE_CLIENT_ID}
      - GOOGLE_CLIENT_SECRET=${GOOGLE_CLIENT_SECRET}

  invoice-service:
    build: ./services/invoice-management
    ports:
      - "3002:3002"
    depends_on:
      - mongodb
    environment:
      - MONGODB_URL=mongodb://mongodb:27017/invoices
      - IPFS_GATEWAY=${IPFS_GATEWAY}

  zk-proof-service:
    build: ./services/zk-proof
    ports:
      - "3003:3003"
    environment:
      - NODE_ENV=development

  government-service:
    build: ./services/government-dashboard
    ports:
      - "3004:3004"
    depends_on:
      - postgres
      - redis

  blockchain-service:
    build: ./services/blockchain-integration
    ports:
      - "3005:3005"
    environment:
      - MULTIVERSX_NETWORK=devnet
      - SMART_CONTRACT_ADDRESS=${CONTRACT_ADDRESS}

  # Frontend
  frontend:
    build: ./frontend
    ports:
      - "5173:5173"
    environment:
      - VITE_API_URL=http://localhost:3000
      - VITE_MULTIVERSX_NETWORK=devnet

volumes:
  postgres_data:
  mongo_data:
```

🔧 Key Technical Decisions

Why These Technologies?

1. **NestJS for Microservices**:
    - Built-in dependency injection
    - Excellent TypeScript support
    - Easy service communication
    - Built-in validation and documentation
2. **React + TypeScript Frontend**:
    - Excellent Web3 ecosystem
    - Strong MultiversX SDK support
    - Better client-side ZK proof generation
    - Flexible component architecture
3. **PostgreSQL + MongoDB Hybrid**:
    - PostgreSQL for relational data (users, compliance)
    - MongoDB for flexible invoice metadata
    - Redis for caching government queries
4. **Client-Side ZK Proof Generation**:
    - Maximum privacy (data never leaves client)
    - Reduces server computational load
    - Modern browsers can handle complex calculations
    - Better user trust and adoption

Privacy Guarantees

1. **Encryption**: All invoice data encrypted client-side with wallet-derived keys
2. **Zero-Knowledge**: Mathematical proofs without data revelation
3. **Decentralized Storage**: IPFS for censorship resistance
4. **Minimal Metadata**: Only essential information stored off-chain

---

🎯 Success Metrics

Technical Milestones

- [ ] End-to-end invoice upload and encryption
- [ ] Successful ZK proof generation and verification
- [ ] Smart contract automated tax calculation
- [ ] Government dashboard compliance monitoring
- [ ] Complete user authentication flow

Performance Targets

- ZK proof generation: < 30 seconds for typical invoice batch
- File encryption/decryption: < 5 seconds per invoice
- Dashboard load time: < 3 seconds
- 99% uptime for all microservices

Thesis Requirements

- Complete working prototype
- Comprehensive technical documentation
- Performance analysis and benchmarks
- Security and privacy analysis
- Future enhancement roadmap

This architecture provides a solid foundation for your bachelor's thesis while maintaining flexibility for future enhancements. The modular design allows you to focus on core functionality first, then add advanced features as time permits.

Next Steps:

1. Set up the development environment with Docker
2. Start with Phase 1, Month 1 tasks

3. Maintain regular progress tracking
4. Document learnings and challenges for thesis writeup