

Career Discovery Platform

System Architecture & Design Document

Version: 1.0

Date: October 23, 2025

Project: Algorand Hackseries #2

Platform: Career Discovery with Blockchain Verification

Executive Summary

The Career Discovery Platform is a full-stack web application that leverages **Algorand blockchain technology** to create transparent, verifiable career guidance and credential records. The platform combines modern web technologies (React, Node.js) with blockchain innovation to solve the problem of unverifiable career achievements and lack of transparency in traditional career guidance systems.

Key Features:

- AI-powered career discovery quizzes
- Blockchain-verified career credentials
- Immutable achievement records on Algorand
- Employer verification system
- Decentralized user profiles

1. System Architecture Overview

1.1 High-Level Architecture

The platform follows a **layered architecture pattern** with clear separation of concerns:

User Layer → **Presentation Layer** → **API Layer** → **Business Logic** → **Data Layer**

Architecture Layers:

1. Presentation Layer (Frontend)

- Technology: React.js
- Purpose: User interface and experience
- Components: Quiz interface, dashboard, career explorer

2. API Gateway Layer

- Technology: Express.js

- Purpose: Request routing, authentication, rate limiting
- Endpoints: REST API for all operations

3. Business Logic Layer

- Services: User, Quiz, Career, Algorand
- Purpose: Core application logic
- Patterns: Service-oriented architecture

4. Blockchain Layer

- Platform: Algorand TestNet
- Purpose: Immutable record storage
- SDK: algosdk (JavaScript)

5. Data Persistence Layer

- Database: MongoDB/PostgreSQL
- Purpose: User data, quiz questions, temporary data
- Blockchain: Permanent credential records

1.2 Technology Stack

Frontend

- **Framework:** React 18
- **State Management:** React Hooks (useState, useEffect, useContext)
- **HTTP Client:** Axios
- **Routing:** React Router v6
- **UI Components:** Custom components with CSS
- **Icons:** React Icons

Backend

- **Runtime:** Node.js v18+
- **Framework:** Express.js v4
- **Blockchain SDK:** algosdk v2
- **Authentication:** JWT (future implementation)
- **Validation:** Express Validator

Blockchain

- **Network:** Algorand TestNet
- **Node Provider:** algonode.cloud
- **Consensus:** Pure Proof-of-Stake (PPoS)
- **Transaction Finality:** 4.5 seconds

Database

- **Primary:** MongoDB (flexible schema for user data)
- **Alternative:** PostgreSQL (structured data)
- **Caching:** Redis (future implementation)

DevOps

- **Version Control:** Git/GitHub
- **Testing:** Jest, Cypress
- **Deployment:** Docker, Kubernetes (future)
- **CI/CD:** GitHub Actions

2. Component Architecture

2.1 Frontend Architecture

Component Hierarchy

```
App.js (Root)
├── Router
│   ├── HomePage
│   ├── QuizPage
│   │   ├── QuizHeader
│   │   ├── QuestionCard
│   │   ├── ProgressBar
│   │   └── SubmitButton
│   ├── ResultsPage
│   │   ├── CareerCard
│   │   ├── SkillsChart
│   │   └── RecommendationList
│   ├── DashboardPage
│   │   ├── UserProfile
│   │   ├── AlgorandWallet
│   │   └── AchievementsList
```

```
|| | └── CareerPathway  
| └── VerificationPage  
| └── TransactionSearch  
| └── CredentialViewer  
| └── BlockchainProof  
```
```

## State Management Strategy

### Local State (useState):

- Component-specific UI state
- Form inputs
- Toggle states

### Context API (useContext):

- User authentication status
- Algorand wallet connection
- Global theme settings

### Server State (React Query - future):

- API data caching
- Background data refresh
- Optimistic updates

## 2.2 Backend Architecture

### Service Layer Pattern

```
```javascript  
// Service Layer Structure  
services/  
|── userService.js // User management  
|── quizService.js // Quiz logic  
|── careerService.js // Career recommendations  
|── algorandService.js // Blockchain operations  
└── verificationService.js // Credential verification  
```
```

### API Routes Structure

```
```javascript  
routes/  
|── userRoutes.js // POST /register, /login  
|── quizRoutes.js // GET /questions, POST /submit
```

```
|── algorandRoutes.js // POST /create-account, /transaction  
|── careerRoutes.js // GET /recommendations, /:id  
└── verificationRoutes.js // GET /verify/:txId  
```
```

### 3. Algorand Blockchain Integration

#### 3.1 Blockchain Architecture

##### Connection Setup

```
```javascript  
const algosdk = require('algosdk');  
  
// Connect to Algorand TestNet  
const algodClient = new algosdk.Algodv2(  
  "", // API token (empty for public nodes)  
  "https://testnet-api.algonode.cloud",  
  "  
);  
```
```

##### Account Management

###### Account Creation Flow:

1. Generate cryptographic key pair
2. Derive Algorand address (58-character base32)
3. Create 25-word mnemonic for recovery
4. Store address in database
5. Return credentials to user

###### Security Considerations:

- Private keys never stored on server
- Mnemonic returned once during creation
- User responsible for key custody
- Non-custodial architecture

#### 3.2 Transaction Design

## Career Credential Transaction Structure

```
```javascript
{
  type: 'pay', // Payment transaction
  from: userAddress, // User's Algorand address
  to: userAddress, // Self-send (0 ALGO)
  amount: 0, // No ALGO transferred
  fee: 1000, // Network fee (0.001 ALGO)
  note: {
    type: 'career_credential',
    userId: 'user123',
    quizId: 'quiz456',
    results: {
      topCareer: 'Blockchain Developer',
      score: 95,
      skills: ['JavaScript', 'Algorand', 'Smart Contracts'],
      timestamp: 1698012345000
    },
    signature: 'platform_signature_hash'
  }
}
```
```

```

Why Self-Send?

- Creates blockchain record without transferring value
- Note field stores credential data
- Immutable and publicly verifiable
- Cost-effective (only transaction fee)

3.3 Smart Contract Architecture (Future)

Planned Smart Contract Functions

Credential Contract:

```
```
issue_credential(user_address, credential_data)
verify_credential(credential_id)
revoke_credential(credential_id, reason)
update_credential_status(credential_id, status)
```
```

```

#### Career NFT Contract:

```
```
mint_career_achievement(user, achievement_data)
```
```

```

```
transfer_achievement(from, to, achievement_id)
query_user_achievements(user_address)
```
```

## 4. Data Architecture

### 4.1 Database Schema

#### User Collection/Table

```
```json
{
  "_id": "ObjectId",
  "email": "user@example.com",
  "name": "John Doe",
  "algorandAddress": "ABCDEF...",
  "createdAt": "2025-10-23T...",
  "profile": {
    "education": "Bachelor's in CS",
    "experience": "2 years",
    "interests": ["blockchain", "AI"]
  },
  "quizResults": ["quizResult_id1", "quizResult_id2"]
}
````
```

#### Quiz Collection/Table

```
```json
{
  "_id": "ObjectId",
  "userId": "user_id",
  "quizType": "AI/ML Career Discovery",
  "questions": [...],
  "answers": [...],
  "results": {
    "topCareer": "Blockchain Developer",
    "matchPercentage": 95,
    "skills": [...]
  },
  "algorandTxId": "TX123...",
  "submittedAt": "2025-10-23T...",
  "verified": true
}
````
```

## Career Collection/Table

```
```json
{
  "_id": "ObjectId",
  "title": "Blockchain Developer",
  "description": "Build decentralized applications...",
  "requiredSkills": ["JavaScript", "Algorand", "Solidity"],
  "salaryRange": "$100k - $150k",
  "category": "Technology",
  "pathways": [...],
  "resources": [...]
}
```
```

```

4.2 Blockchain Data Structure

On-Chain Data (Algorand):

- Transaction IDs (permanent, immutable)
- Credential hashes
- Timestamp proofs
- User addresses

Off-Chain Data (Database):

- User profiles
- Quiz questions and answers
- Detailed career information
- User preferences

Hybrid Approach Benefits:

- Blockchain: Verification and immutability
- Database: Query performance and flexibility
- Optimal cost and performance balance

5. API Design

5.1 RESTful Endpoints

User Endpoints

```
```
POST /api/users/register
POST /api/users/login
GET /api/users/profile
PUT /api/users/profile
DELETE /api/users/account
```
```

Quiz Endpoints

```
```
GET /api/quiz/questions?category=AI
POST /api/quiz/submit
GET /api/quiz/results/:id
GET /api/quiz/history
```
```

Algorand Endpoints

```
```
POST /api/algorand/create-account
GET /api/algorand/balance/:address
POST /api/algorand/transaction
GET /api/algorand/transaction/:txId
POST /api/algorand/verify-credential
```
```

Career Endpoints

```
```
GET /api/careers
GET /api/careers/:id
GET /api/careers/recommendations
POST /api/careers/search
```
```

5.2 Request/Response Format

Standard Response Structure

Success Response:

```
```json
{
 "success": true,
 "data": {
 "id": "1234567890",
 "name": "John Doe",
 "email": "john.doe@example.com",
 "experience": "5 years"
 }
}```
```

```
// Actual data
},
"message": "Operation successful",
"timestamp": 1698012345000
}
```

```

Error Response:

```
```json
{
"success": false,
"error": {
"code": "INVALID_INPUT",
"message": "Email is required",
"details": {...}
},
"timestamp": 1698012345000
}
```

```

6. Security Architecture

6.1 Authentication & Authorization

Current Implementation:

- Basic session-based authentication
- User credentials in database

Future Implementation:

- JWT (JSON Web Tokens)
- Role-based access control (RBAC)
- OAuth 2.0 for social login
- Algorand wallet connection for authentication

6.2 Data Security

In Transit:

- HTTPS/TLS encryption
- Secure WebSocket connections
- API rate limiting

At Rest:

- Database encryption

- Password hashing (bcrypt)
- Encrypted environment variables

Blockchain Security:

- Non-custodial wallet architecture
- Client-side transaction signing
- No private keys on server

6.3 Input Validation

```
```javascript
// Example validation middleware
const validateQuizSubmission = (req, res, next) => {
 const { userId, answers } = req.body;

 if (!userId) {
 return res.status(400).json({
 success: false,
 error: 'User ID required'
 });
 }

 if (!Array.isArray(answers) || answers.length === 0) {
 return res.status(400).json({
 success: false,
 error: 'Valid answers required'
 });
 }

 next();
};

```

```

7. Scalability & Performance

7.1 Scaling Strategy

Horizontal Scaling

- Load balancer (Nginx)
- Multiple API server instances
- Database replication
- CDN for static assets

Vertical Scaling

- Optimize database queries
- Redis caching layer
- Connection pooling
- Code optimization

7.2 Caching Strategy

Browser Cache:

- Static assets (images, CSS, JS)
- Cache-Control headers

Application Cache (Redis):

- User sessions
- Quiz questions
- Career data
- Algorand balance checks

Database Query Cache:

- Frequently accessed data
- Aggregation results

7.3 Performance Metrics

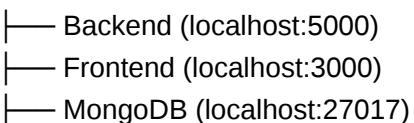
Target Metrics:

- API Response Time: < 200ms
- Page Load Time: < 2 seconds
- Algorand Transaction: < 5 seconds
- Database Queries: < 50ms
- Uptime: 99.9%

8. Deployment Architecture

8.1 Development Environment

Local Machine



└── Algorand TestNet (remote)

...

8.2 Production Environment (Planned)

...

Cloud Infrastructure (AWS/Azure/GCP)

 └── Load Balancer

 └── API Servers (Auto-scaling)

 | └── Docker containers

 | └── Kubernetes pods

 └── Database Cluster

 | └── Primary (read/write)

 | └── Replicas (read-only)

 └── Cache Layer (Redis)

 └── CDN (CloudFront/Cloudflare)

 └── Monitoring (Prometheus/Grafana)

...

8.3 CI/CD Pipeline

...

GitHub → Actions → Build → Test → Deploy

 └── Staging

 └── Production

...

Pipeline Stages:

1. Code commit triggers pipeline
2. Run linting and formatting
3. Run unit tests
4. Run integration tests
5. Build Docker images
6. Push to container registry
7. Deploy to staging
8. Run E2E tests
9. Manual approval
10. Deploy to production

9. Integration Architecture

9.1 External Services

Algorand Network

- **Purpose:** Blockchain operations
- **Connection:** algosdk
- **Endpoint:** testnet-api.algonode.cloud
- **Fallback:** Multiple node providers

Wallet Integration (Future)

- **Pera Wallet:** Mobile + browser extension
- **Defly Wallet:** Advanced features
- **MyAlgo Wallet:** Web-based
- **WalletConnect:** Universal connector

Third-Party APIs (Future)

- **Job Boards:** Indeed, LinkedIn APIs
- **Learning Platforms:** Coursera, Udemy
- **Career Data:** O*NET, Bureau of Labor Statistics

9.2 Webhook Architecture (Future)

Incoming Webhooks:

- Payment confirmations
- Algorand transaction notifications
- Third-party integrations

Outgoing Webhooks:

- Notify users of credential issuance
- Alert employers of new verifications
- Integration with learning platforms

10. Monitoring & Observability

10.1 Application Monitoring

Metrics to Track:

- Request rate
- Error rate
- Response time
- CPU/Memory usage
- Database connections

Tools:

- Prometheus (metrics collection)
- Grafana (visualization)
- ELK Stack (log aggregation)

10.2 Blockchain Monitoring

Algorand Metrics:

- Transaction success rate
- Transaction confirmation time
- Account balances
- Network health
- Gas fee trends

Tools:

- AlgoExplorer API
- Custom monitoring scripts
- Alerting for failed transactions

10.3 User Analytics

Metrics:

- Active users
- Quiz completion rate
- Career path selections
- Verification requests
- Time spent on platform

Tools:

- Google Analytics

- Custom analytics dashboard
- User behavior tracking

11. Disaster Recovery & Backup

11.1 Backup Strategy

Database Backups:

- Automated daily backups
- Point-in-time recovery
- Geo-redundant storage
- Retention: 30 days

Blockchain Data:

- Already immutable and replicated
- Node providers handle redundancy
- Transaction IDs stored in database backup

11.2 Disaster Recovery Plan

RTO (Recovery Time Objective): 1 hour

RPO (Recovery Point Objective): 5 minutes

Failover Strategy:

1. Automated health checks
2. Automatic failover to standby
3. Alert operations team
4. Restore from backup if needed
5. Post-mortem analysis

12. Future Enhancements

12.1 Phase 2 Features (Next 2 Months)

1. Smart Contract Deployment

- AlgoKit integration
- Credential issuance contract
- Automated verification

2. Wallet Integration

- Pera Wallet connection
- Multi-wallet support
- Transaction signing UI

3. Career NFTs

- Achievement badges as NFTs
- Tradeable credentials
- Employer verification dashboard

12.2 Phase 3 Features (6 Months)

1. AI-Powered Recommendations

- Machine learning models
- Personalized career paths
- Job matching algorithm

2. Marketplace

- Job listings
- Course recommendations
- Mentorship connections

3. DAO Governance

- Community voting on features
- Platform fee distribution
- Decentralized moderation

12.3 Scaling to MainNet

Migration Checklist:

- [] Comprehensive security audit
- [] Smart contract formal verification
- [] Load testing (10,000+ concurrent users)
- [] Legal compliance review
- [] User education on MainNet costs
- [] Gradual rollout plan

13. Conclusion

The Career Discovery Platform represents a novel approach to solving transparency and verification issues in career guidance by leveraging **Algorand blockchain technology**. The architecture is designed with **modularity, scalability, and security** as core principles.

Key Architectural Strengths:

- ✓ Clear separation of concerns (layered architecture)
- ✓ Blockchain integration for immutability
- ✓ Scalable and performant design
- ✓ Security-first approach
- ✓ Future-ready with room for growth

Technical Innovation:

- Novel use of blockchain for credential verification
- Hybrid data storage (on-chain + off-chain)
- Non-custodial, user-centric design
- Low transaction costs using Algorand

This architecture provides a solid foundation for building a production-ready platform that can serve thousands of users while maintaining the integrity and transparency that blockchain technology enables.

Appendix A: Glossary

Algorand: Layer-1 blockchain using Pure Proof-of-Stake consensus

TestNet: Testing network for Algorand (free ALGO for testing)

MainNet: Production network for Algorand (real ALGO required)

alosdk: Official Algorand JavaScript SDK

Transaction ID: Unique identifier for blockchain transactions

Mnemonic: 25-word phrase for account recovery

Smart Contract: Self-executing code on blockchain

NFT: Non-Fungible Token (unique digital asset)

DAO: Decentralized Autonomous Organization

dApp: Decentralized Application

Appendix B: References

1. Algorand Developer Portal: <https://developer.algorand.org/>
2. AlgoKit Documentation: <https://github.com/algorandfoundation/algokit-cli>
3. React Documentation: <https://react.dev/>
4. Express.js Guide: <https://expressjs.com/>

5. MongoDB Documentation: <https://docs.mongodb.com/>

Document History:

| Version | Date | Author | Changes |
|---------|------------|---------------|-------------------------------|
| 1.0 | 2025-10-23 | Platform Team | Initial architecture document |

For questions or contributions, contact the development team.