ProjectX: Single-Chain Architecture with x402 on Avalanche

MVP Architecture for Avalanche Hackathon

Last Updated: December 7, 2025 Time Remaining: 1 day to submission

---

🎯 CRITICAL UPDATE: All-In on Avalanche

Game-Changing Insight: x402 has **native Avalanche C-Chain support** with facilitators specifically optimized for Avalanche! This means we can do EVERYTHING on one chain.

Why This Is Perfect

x402 on Avalanche uses USDC for payments, with facilitators maintaining hot wallets funded with AVAX to sponsor gas fees for all USDC payment settlements - clients only need USDC in their wallets, no AVAX required. This creates a unified experience where:

- **Escrow payments:** AVAX (for large gig payments)
- **Micropayments:** USDC via x402 (for platform fees)
- **Gas fees:** Abstracted away (facilitator pays)
- **Settlement speed:** ~2 seconds
- **Network:** 100% Avalanche C-Chain

🏗️ Simplified Technical Architecture

Single-Chain, Dual-Token Model

💡 Why This Architecture Is Superior

1. Single Chain = Simpler UX

- Users only need **one wallet** connected to Avalanche
- No bridge confusion
- No multi-chain state synchronization
- Easier to demo ("everything happens on Avalanche")

2. Two-Token Model = Clear Separation

- **AVAX:** Trust layer (escrow, gas for contract calls)
- **USDC:** Revenue layer (micropayments via x402)
- Both native to Avalanche C-Chain

3. Gas Abstraction via x402

The facilitator maintains a hot wallet funded with AVAX to sponsor gas fees for all USDC payment settlements, so clients only need USDC in their wallets. Users don't need AVAX for micropayments!

4. Fast & Cheap

- Avalanche C-Chain offers fast finality and x402 settles payments in approximately 2 seconds
- Gas costs on Avalanche: ~$0.001-0.01 per transaction
- x402 micropayments: $0.001-0.01 USDC range realistic on Avalanche C-Chain

5. Native Avalanche Facilitators

Ultravioleta DAO offers the x402 protocol optimized for Avalanche deployment on both mainnet and testnet, meaning production-ready infrastructure exists NOW.

💰 Revenue Model: x402 USDC on Avalanche

Core Principle

"Platform fees in USDC, gig payments in AVAX"

This creates two revenue streams:

1. **Micropayments (x402 USDC):** Platform operational revenue
2. **Escrow locks (AVAX):** Trust mechanism (no platform cut)

Revenue Streams (All via x402 on Avalanche)

1. Gig Posting Fees

```javascript
// Employer posts a featured gig
POST /api/gigs/featured
X-PAYMENT: 0.50 USDC (Avalanche C-Chain)

// Pricing tiers:
POST /api/gigs/create       // FREE (basic listing)
POST /api/gigs/featured     // $0.50 USDC (24h top placement)
POST /api/gigs/urgent       // $1.00 USDC (urgent badge)
POST /api/gigs/verified     // $0.25 USDC (verified employer badge)
```

Why USDC for fees?

- Stable pricing ($0.50 always = $0.50)
- No volatility risk
- Standard for business payments

2. Worker Applications

```javascript
// Worker applies to gig
POST /api/gigs/{id}/apply
X-PAYMENT: 0.02 USDC

// Quality filter:
// - Prevents spam applications
// - $0.02 is negligible for serious workers
// - Creates friction for bots
```

3. Badge Verification

```javascript
// Platform verifies skill + mints badge
POST /api/badges/verify
X-PAYMENT: 5.00 USDC

// Process:
// 1. Worker pays $5 USDC via x402
// 2. Admin reviews portfolio
// 3. Platform mints NFT badge on Avalanche
// 4. Badge appears in wallet
```

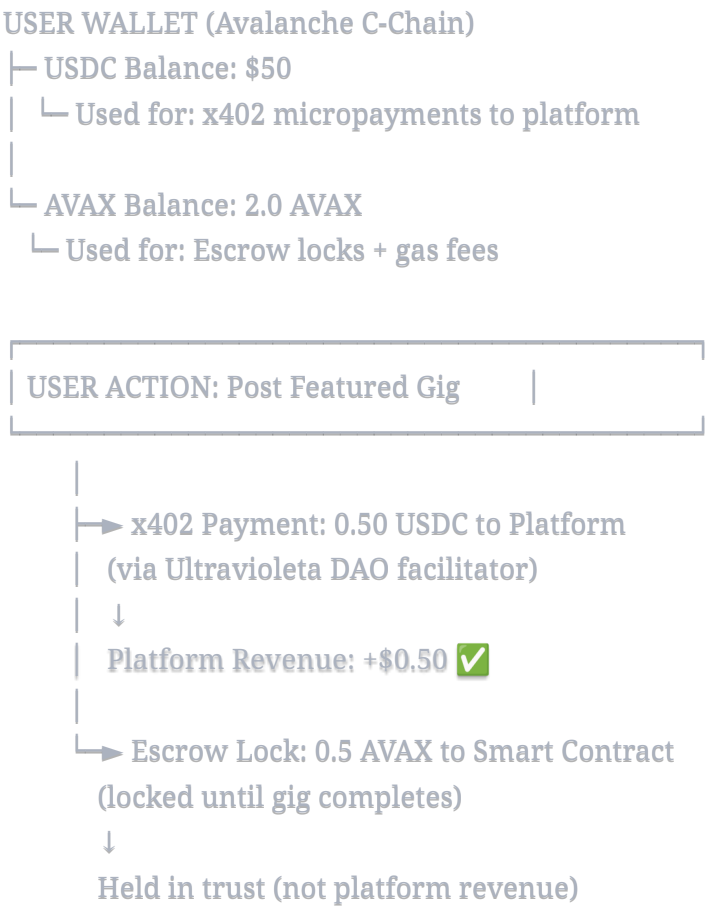
4. Premium Features

```javascript
// Advanced search with AI matching
POST /api/search/ai-match
X-PAYMENT: 0.10 USDC per query

// Analytics dashboard
GET /api/analytics/monthly
X-PAYMENT: 2.00 USDC (monthly access)

// Dispute resolution
POST /api/disputes/create
X-PAYMENT: 10.00 USDC (refundable if valid)
```

Revenue Flow Diagram



🔧 Implementation: x402 on Avalanche

Step 1: Backend Setup (Node.js + Hono)

Ultravioleta DAO provides production-ready x402 integration for Avalanche:

javascript

```javascript
import { Hono } from "hono";
import { paymentMiddleware } from "x402-hono";

const app = new Hono();

// Configure x402 for Avalanche C-Chain
app.use(paymentMiddleware(
  "0xYourWalletAddress", // Your USDC receiving address on Avalanche
  {
    "/api/gigs/featured": {
      price: "$0.50",
      network: "avalanche-c-chain", // or "avalanche-fuji" for testnet
      config: {
        description: "Feature your gig for 24 hours"
      }
    },
    "/api/gigs/urgent": {
      price: "$1.00",
      network: "avalanche-c-chain",
      config: {
        description: "Mark gig as urgent"
      }
    },
    "/api/badges/verify": {
      price: "$5.00",
      network: "avalanche-c-chain",
      config: {
        description: "Verify skill and mint badge NFT"
      }
    }
  },
  {
    url: 'https://facilitator.ultravioletadao.xyz' // Avalanche facilitator
  }
));

// Protected endpoint - requires x402 payment
app.post('/api/gigs/featured', async (c) => {
  // If code reaches here, $0.50 USDC payment succeeded
  const { gigId, title, description } = await c.req.json();

  // Mark gig as featured in database
  await db.gigs.update(gigId, {
    featured: true,
    featuredUntil: Date.now() + 86400000 // 24 hours
  });
```

```
  return c.json({
    success: true,
    message: 'Gig featured for 24 hours',
    revenue: 0.50 // Platform earned $0.50 USDC
  });
});

// Free endpoint - no payment required
app.get('/api/gigs', async (c) => {
  const gigs = await db.gigs.find({ status: 'open' });
  return c.json({ success: true, gigs });
});
```

Step 2: Client-Side (React + x402-client)

javascript

```javascript
import { X402Client } from "x402-client";

// Initialize x402 client for Avalanche
const client = new X402Client({
  privateKey: userWallet.privateKey,
  facilitatorUrl: 'https://facilitator.ultravioletadao.xyz',
  network: 'avalanche-c-chain' // or 'avalanche-fuji' for testnet
});

async function featureGig(gigId) {
  try {
    // First attempt: No payment
    let response = await fetch('/api/gigs/featured', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ gigId, title: 'Build landing page' })
    });

    // Server requires payment (HTTP 402)
    if (response.status === 402) {
      const paymentRequirements = await response.json();

      // Show user confirmation modal
      const confirmed = await showPaymentModal({
        amount: '$0.50 USDC',
        network: 'Avalanche C-Chain',
        purpose: 'Feature gig for 24 hours'
      });

      if (!confirmed) return;

      // x402 client handles payment automatically
      response = await client.post('/api/gigs/featured', {
        data: { gigId, title: 'Build landing page' }
      });
    }

    const result = await response.json();
    showNotification('Gig featured! ✨');

  } catch (error) {
    if (error.message.includes('insufficient USDC')) {
      showError('Please add USDC to your wallet on Avalanche');
    } else {
      showError('Payment failed: ' + error.message);
    }
  }
}
```

```
}
```

Step 3: Environment Configuration

```bash
# .env file

# Avalanche C-Chain (or Fuji testnet)
AVALANCHE_RPC_URL=https://api.avax-test.network/ext/bc/C/rpc
CHAIN_ID=43113  # Fuji testnet (43114 for mainnet)

# Smart Contracts (deployed on Avalanche)
ESCROW_CONTRACT_ADDRESS=0x...
BADGE_CONTRACT_ADDRESS=0x...

# x402 Configuration (Avalanche)
X402_PAYMENT_ADDRESS=0x...  # Your USDC receiving address
X402_FACILITATOR_URL=https://facilitator.ultravioletadao.xyz
X402_NETWORK=avalanche-fuji  # or avalanche-c-chain for mainnet

# Admin Wallet (for minting badges)
ADMIN_PRIVATE_KEY=0x...
ADMIN_ADDRESS=0x...

# USDC Token on Avalanche Fuji
USDC_ADDRESS=0x5425890298aed601595a70AB815c96711a31Bc65  # Fuji testnet USDC
```

🎯 Smart Contract Architecture

Contract 1: Escrow (AVAX Payments)

solidity

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "@openzeppelin/contracts/security/ReentrancyGuard.sol";

contract GigEscrow is ReentrancyGuard {
 enum GigStatus { OPEN, ASSIGNED, SUBMITTED, COMPLETED, CANCELLED, DISPUTED }

 struct Gig {
  address employer;
  address worker;
  uint256 paymentAmount;  // In AVAX
  GigStatus status;
  uint256 createdAt;
  uint256 completedAt;
 }

 mapping(uint256 => Gig) public gigs;
 uint256 public nextGigId;

 event GigCreated(uint256 indexed gigId, address indexed employer, address worker, uint256 amount);
 event WorkSubmitted(uint256 indexed gigId);
 event PaymentReleased(uint256 indexed gigId, address indexed worker, uint256 amount);
 event GigCancelled(uint256 indexed gigId);

 // Employer locks AVAX for gig
 function createGig(address _worker) external payable returns (uint256) {
  require(msg.value > 0, "Must lock payment");
  require(_worker != address(0), "Invalid worker address");

  uint256 gigId = nextGigId++;
  gigs[gigId] = Gig({
   employer: msg.sender,
   worker: _worker,
   paymentAmount: msg.value,
   status: GigStatus.ASSIGNED,
   createdAt: block.timestamp,
   completedAt: 0
  });

  emit GigCreated(gigId, msg.sender, _worker, msg.value);
  return gigId;
 }

 // Worker submits completed work
 function submitWork(uint256 _gigId) external {
  Gig storage gig = gigs[_gigId];
```

```
    require(msg.sender == gig.worker, "Only worker can submit");
    require(gig.status == GigStatus.ASSIGNED, "Wrong status");

    gig.status = GigStatus.SUBMITTED;
    gig.completedAt = block.timestamp;

    emit WorkSubmitted(_gigId);
  }

  // Employer approves and releases AVAX to worker
  function releasePayment(uint256 _gigId) external nonReentrant {
    Gig storage gig = gigs[_gigId];
    require(msg.sender == gig.employer, "Only employer");
    require(gig.status == GigStatus.SUBMITTED, "Work not submitted");

    gig.status = GigStatus.COMPLETED;

    // Transfer AVAX to worker
    (bool success, ) = payable(gig.worker).call{value: gig.paymentAmount}("");
    require(success, "Transfer failed");

    emit PaymentReleased(_gigId, gig.worker, gig.paymentAmount);
  }

  // Employer cancels before work submitted
  function cancelGig(uint256 _gigId) external nonReentrant {
    Gig storage gig = gigs[_gigId];
    require(msg.sender == gig.employer, "Only employer");
    require(
      gig.status == GigStatus.OPEN || gig.status == GigStatus.ASSIGNED,
      "Cannot cancel"
    );

    gig.status = GigStatus.CANCELLED;

    // Refund AVAX to employer
    (bool success, ) = payable(gig.employer).call{value: gig.paymentAmount}("");
    require(success, "Refund failed");

    emit GigCancelled(_gigId);
  }

  // View function
  function getGig(uint256 _gigId) external view returns (Gig memory) {
    return gigs[_gigId];
  }
}
```

Contract 2: Badge NFT (Reputation)

solidity

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
import "@openzeppelin/contracts/access/Ownable.sol";

contract SkillBadge is ERC721, Ownable {
 enum BadgeLevel { BEGINNER, INTERMEDIATE, EXPERT }

 struct Badge {
  string skillName;     // "Web Development"
  string iconURI;       // IPFS URL
  uint256 issuedAt;
  BadgeLevel level;
 }

 mapping(uint256 => Badge) public badges;
 mapping(address => uint256[]) private userBadges;
 uint256 public nextTokenId;

 event BadgeMinted(uint256 indexed tokenId, address indexed holder, string skillName);

 constructor() ERC721("SkillBadge", "SKILL") Ownable(msg.sender) {}

 // Platform mints badge (after x402 payment + verification)
 function mintBadge(
  address _to,
  string memory _skillName,
  string memory _iconURI,
  BadgeLevel _level
 ) external onlyOwner returns (uint256) {
  require(_to != address(0), "Invalid address");
  require(bytes(_skillName).length > 0, "Skill name required");

  uint256 tokenId = nextTokenId++;
  _safeMint(_to, tokenId);

  badges[tokenId] = Badge({
   skillName: _skillName,
   iconURI: _iconURI,
   issuedAt: block.timestamp,
   level: _level
  });

  userBadges[_to].push(tokenId);

  emit BadgeMinted(tokenId, _to, _skillName);
```

```
  return tokenId;
}

// Get all badge IDs for a user
function getBadges(address _user) external view returns (uint256[] memory) {
  return userBadges[_user];
}

// Get badge details
function getBadgeDetails(uint256 _tokenId) external view returns (Badge memory) {
  require(_ownerOf(_tokenId) != address(0), "Badge doesn't exist");
  return badges[_tokenId];
}

// Make badges non-transferable (soulbound)
function _update(
  address to,
  uint256 tokenId,
  address auth
) internal virtual override returns (address) {
  address from = _ownerOf(tokenId);

  // Allow minting (from = 0) and burning (to = 0)
  // Block transfers between users
  require(
    from == address(0) || to == address(0),
    "Badges are non-transferable"
  );

  return super._update(to, tokenId, auth);
}

// Override tokenURI for metadata
function tokenURI(uint256 _tokenId) public view override returns (string memory) {
  require(_ownerOf(_tokenId) != address(0), "Badge doesn't exist");
  Badge memory badge = badges[_tokenId];

  // Return JSON metadata
  return string(abi.encodePacked(
    '{"name":"', badge.skillName, ' Badge",',
    '"image":"', badge.iconURI, '",',
    '"description":"Verified skill badge on ProjectX",',
    '"attributes":[',
      '{"trait_type":"Skill","value":"', badge.skillName, '"},',
      '{"trait_type":"Level","value":"', _levelToString(badge.level), '"},',
      '{"trait_type":"Issued","value":', _toString(badge.issuedAt), '}',
    ']}'
  ));
}
```

```
  function _levelToString(BadgeLevel _level) private pure returns (string memory) {
    if (_level == BadgeLevel.BEGINNER) return "Beginner";
    if (_level == BadgeLevel.INTERMEDIATE) return "Intermediate";
    return "Expert";
  }

  function _toString(uint256 value) private pure returns (string memory) {
    if (value == 0) return "0";
    uint256 temp = value;
    uint256 digits;
    while (temp != 0) {
      digits++;
      temp /= 10;
    }
    bytes memory buffer = new bytes(digits);
    while (value != 0) {
      digits -= 1;
      buffer[digits] = bytes1(uint8(48 + uint256(value % 10)));
      value /= 10;
    }
    return string(buffer);
  }
}
```

💸 Complete Payment Flow Example

Scenario: Employer posts featured gig with worker

```
1. EMPLOYER ACTION: Post Featured Gig
   ├─► Frontend call: POST /api/gigs/featured
   ├─► Server responds: HTTP 402 Payment Required
   │   └─► Payment details: $0.50 USDC on Avalanche
   │
   ├─► User confirms in wallet modal
   ├─► x402 client signs payment (USDC.transferWithAuthorization)
   ├─► Ultravioleta facilitator verifies signature
   ├─► Facilitator submits on-chain (pays gas in AVAX)
   ├─► Backend receives confirmation: Payment settled ✅
   │   └─► Platform revenue: +$0.50 USDC
   │
   └─► Backend responds: 200 OK - Gig created

2. EMPLOYER ACTION: Lock Payment in Escrow
   ├─► Frontend calls: escrowContract.createGig(workerAddress)
   ├─► Transaction: { value: 0.5 AVAX, gas: ~50,000 }
   ├─► User signs with MetaMask
   ├─► AVAX locked in smart contract
   ├─► Event emitted: GigCreated(gigId: 42, amount: 0.5 AVAX)
   └─► Gig ID returned: 42

3. WORKER ACTION: Submit Work
   ├─► Frontend calls: escrowContract.submitWork(42)
   ├─► Transaction: { gas: ~30,000 }
   ├─► Status updated: ASSIGNED → SUBMITTED
   └─► Event emitted: WorkSubmitted(gigId: 42)

4. EMPLOYER ACTION: Approve & Release Payment
   ├─► Frontend calls: escrowContract.releasePayment(42)
   ├─► Transaction: { gas: ~50,000 }
   ├─► Smart contract transfers: 0.5 AVAX → worker wallet
   ├─► Status updated: SUBMITTED → COMPLETED
   └─► Event emitted: PaymentReleased(gigId: 42, amount: 0.5 AVAX)

FINAL STATE:
├─► Platform earned: $0.50 USDC (x402 payment)
├─► Worker earned: 0.5 AVAX (escrow release)
├─► Gas costs: ~$0.003 total (paid by employer + worker)
└─► Total platform fee: $0.50 vs Upwork's $100 (20% of $500 gig)
```

📊 Revenue Projections (All on Avalanche)

Conservative Estimates (Month 1-3)

Revenue Stream	Price (USDC)	Est. Usage/Month	Monthly Revenue
Featured Gigs	$0.50	100	$50
Urgent Gigs	$1.00	50	$50
Applications	$0.02	5,000	$100
Badge Verification	$5.00	50	$250
Analytics Access	$2.00	100	$200
Total			**$650/mo**

At Scale (Month 12)

Revenue Stream	Price (USDC)	Est. Usage/Month	Monthly Revenue
Featured Gigs	$0.50	5,000	$2,500
Urgent Gigs	$1.00	2,000	$2,000
Applications	$0.02	250,000	$5,000
Badge Verification	$5.00	2,000	$10,000
Premium Features	varies	varies	$15,000
Total			**$34,500/mo**

Annual Revenue: $414,000 (all in USDC on Avalanche)

Key Insight: On Avalanche C-Chain, gas costs per transaction tend to land in the low single-digit cents, making micropayments realistic in the few-cents range. This means we can profitably charge $0.01-1.00 per action.

🚀 MVP Implementation Plan (1 Day)

Hour 0-4: Smart Contracts

- ✅ Deploy Escrow contract to Avalanche Fuji
- ✅ Deploy Badge NFT to Avalanche Fuji
- ✅ Test with 0.1 AVAX escrow lock/release
- ✅ Mint 1 test badge to your wallet

Hour 4-8: Backend + x402

- ✅ Setup Hono server with x402-hono middleware
- ✅ Configure Ultravioleta facilitator for Avalanche Fuji
- ✅ Protect `/api/gigs/featured` with $0.50 USDC payment
- ✅ Test x402 flow with testnet USDC

Hour 8-12: Frontend

- ✅ Setup React + RainbowKit (Avalanche only)
- ✅ Create gig form + escrow integration
- ✅ Feature gig button (x402 payment modal)
- ✅ Display badges from smart contract

Testing Checklist

- [ ] x402 payment succeeds (USDC on Fuji)
- [ ] Escrow locks/releases AVAX correctly
- [ ] Badge mints to wallet address
- [ ] All transactions visible on Snowtrace
- [ ] Demo completes in under 2 minutes

🎯 Demo Script (2 Minutes)

Slide 1: Problem (15s)

> "Traditional gig platforms like Upwork extract 20% commissions and have no way to verify skills. Workers lose money, employers hire blindly."

Slide 2: Solution (20s)

> "ProjectX solves this with x402 micropayments and blockchain escrow—both on Avalanche. Instead of 20% fees, we charge $0.50 per featured gig. Skills are verified with NFT badges."

Slide 3: Live Demo (60s)

```
1. Connect wallet → Avalanche C-Chain
2. Create gig → Lock 0.5 AVAX in escrow
3. Feature gig → Pay $0.50 USDC via x402 (show payment modal)
4. Show on Snowtrace → Both transactions confirmed
5. Show revenue → Platform earned $0.50 in 2 seconds
```

Slide 4: Revenue Model (20s)

> "At scale, we project $34k/month from micropayments—all on Avalanche. Compare that to needing $4.2M in transaction volume with Upwork's 20% model."

Slide 5: Future (5s)

> "Next: Launch our own Avalanche L1 where badge holders become validators. Thank you!"

🔮 Post-MVP: Custom Avalanche L1

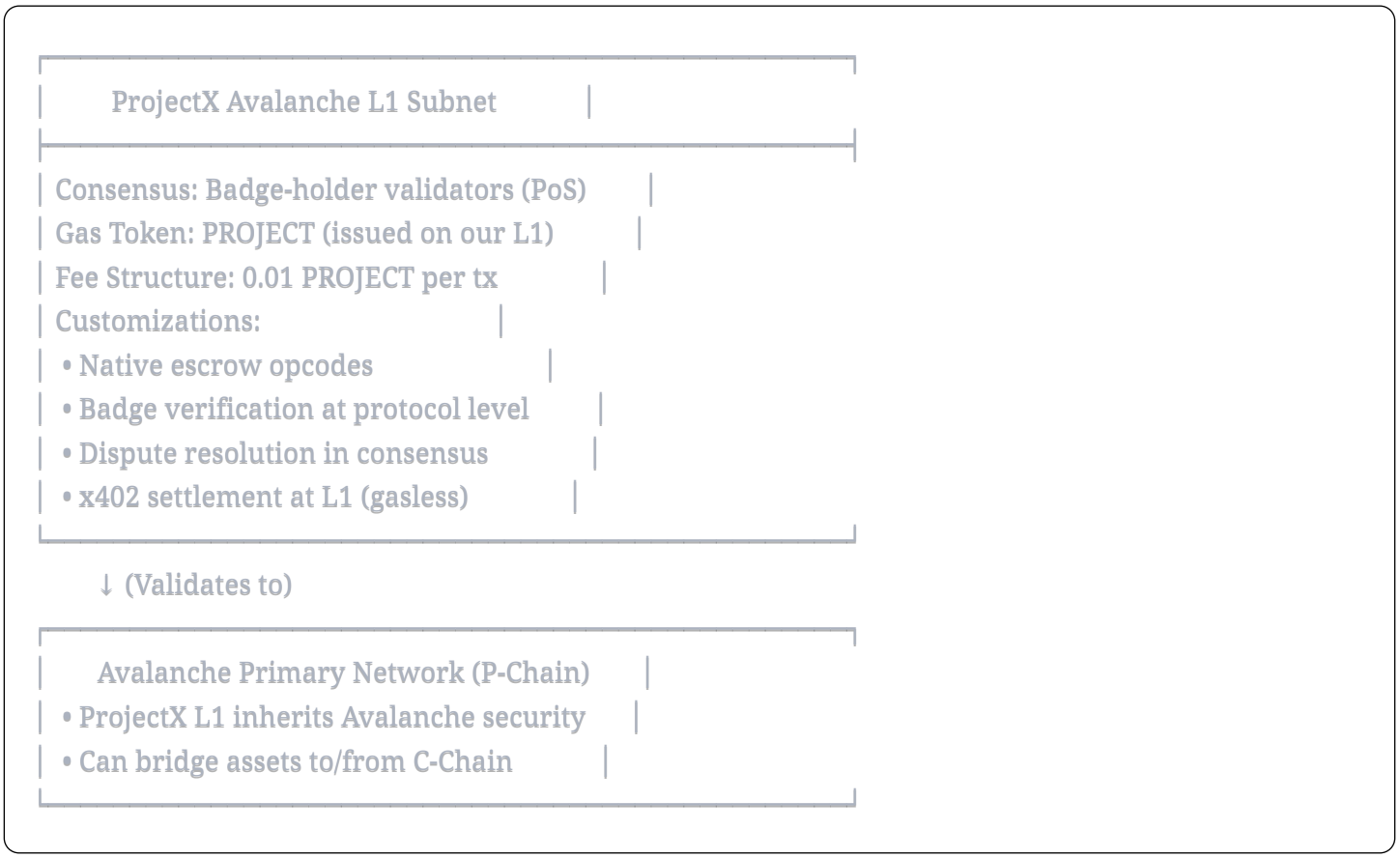
Why Launch a Custom L1?

Current State: ProjectX runs on Avalanche C-Chain (shared EVM)

Problem:

- Competing for blockspace with other dapps
- Generic gas pricing (can't optimize for our use case)
- Limited customization of consensus

Solution: ProjectX L1 (Avalanche Subnet)



L1 Advantages

1. **Custom VM for Gigs**
   - Escrow logic at protocol level (cheaper than smart contracts)
   - Native badge validation (no ERC-721 overhead)
   - x402 settlement built into consensus (gasless micropayments)
2. **Badge-Holder Validators**
   - Only users with verified badges can validate
   - Creates "skin in the game" (validators lose badges if malicious)
   - Stake badges instead of tokens
   - Earns validation rewards in PROJECT token
3. **Custom Fee Market**
   - Pay gas in PROJECT token, not AVAX
   - Dynamic pricing for peak times
   - Revenue directly to validator set (badge holders)
4. **Regulatory Advantages**
   - KYC validators only (for enterprise gigs)
   - Compliance at protocol level
   - Jurisdiction-aware transaction routing

Timeline to L1

- **Month 1-3:** Launch on C-Chain, prove product-market fit
- **Month 4-6:** Design L1 tokenomics + validator requirements
- **Month 7-9:** Deploy testnet L1, migrate contracts
- **Month 10-12:** Mainnet launch, onboard validators

🛠️ Quick Start Guide: x402 on Avalanche Fuji

Prerequisites

```bash
# Install dependencies
npm install hono x402-hono ethers dotenv

# Or with yarn
yarn add hono x402-hono ethers dotenv
```

Step 1: Get Testnet USDC on Fuji

javascript

```javascript
// Avalanche Fuji Testnet USDC
const FUJI_USDC = "0x5425890298aed601595a70AB815c96711a31Bc65";

// Option 1: Use Avalanche Faucet
// Visit: https://core.app/tools/testnet-faucet
// Connect wallet → Select "Fuji C-Chain" → Request AVAX + USDC

// Option 2: Swap testnet AVAX for USDC on Trader Joe (Fuji)
// 1. Get AVAX from faucet
// 2. Go to https://testnet.traderjoexyz.com
// 3. Swap AVAX → USDC
```

Step 2: Configure Backend (Hono + x402)

javascript

```javascript
// server.js
import { Hono } from 'hono';
import { cors } from 'hono/cors';
import { paymentMiddleware } from 'x402-hono';
import { ethers } from 'ethers';
import dotenv from 'dotenv';

dotenv.config();

const app = new Hono();

// Enable CORS for frontend
app.use('/*', cors({
  origin: 'http://localhost:5173',
  credentials: true
}));

// x402 configuration for Avalanche Fuji
const X402_CONFIG = {
  paymentAddress: process.env.X402_PAYMENT_ADDRESS, // Your USDC receiving address
  facilitatorUrl: 'https://facilitator.ultravioletadao.xyz',
  network: 'avalanche-fuji', // or 'avalanche-c-chain' for mainnet
  routes: {
    '/api/gigs/featured': {
      price: '$0.50',
      description: 'Feature your gig for 24 hours'
    },
    '/api/gigs/urgent': {
      price: '$1.00',
      description: 'Mark gig as urgent with badge'
    },
    '/api/gigs/:id/apply': {
      price: '$0.02',
      description: 'Apply to this gig'
    },
    '/api/badges/verify': {
      price: '$5.00',
      description: 'Verify skill and mint badge NFT'
    },
    '/api/search/advanced': {
      price: '$0.10',
      description: 'AI-powered gig matching'
    }
  }
};

// Apply x402 middleware
```

```
app.use("*", paymentMiddleware(
  X402_CONFIG.paymentAddress,
  X402_CONFIG.routes,
  { url: X402_CONFIG.facilitatorUrl }
));

// ===== PROTECTED ENDPOINTS (Require x402 Payment) =====

app.post('/api/gigs/featured', async (c) => {
  // Payment verified by middleware - code only runs if $0.50 USDC paid
  const { gigId, title, description } = await c.req.json();

  // Update database
  await db.gigs.updateOne(
    { _id: gigId },
    {
      $set: {
        featured: true,
        featuredUntil: new Date(Date.now() + 86400000) // 24 hours
      }
    }
  );

  // Log revenue
  await db.revenue.insertOne({
    type: 'featured_gig',
    amount: 0.50,
    currency: 'USDC',
    gigId,
    timestamp: new Date(),
    txHash: c.req.header('x-payment-tx') // x402 provides this
  });

  return c.json({
    success: true,
    message: 'Gig featured for 24 hours',
    revenue: '$0.50 USDC'
  });
});

app.post('/api/badges/verify', async (c) => {
  // Payment verified - $5.00 USDC received
  const { userAddress, skillName, portfolioUrl } = await c.req.json();

  // Queue for manual review
  const verificationId = await db.verifications.insertOne({
    userAddress,
    skillName,
    portfolioUrl,
    status: 'pending',
```

```
    paidAt: new Date(),
    amount: 5.00
  });

  return c.json({
    success: true,
    message: 'Verification request submitted. Review within 24h.',
    verificationId
  });
});

// ===== FREE ENDPOINTS (No Payment Required) =====

app.get('/api/gigs', async (c) => {
  const { status, requiredBadge, limit = 20, skip = 0 } = c.req.query();

  const query = {};
  if (status) query.status = status;
  if (requiredBadge) query.requiredBadge = requiredBadge;

  const gigs = await db.gigs
    .find(query)
    .sort({ featured: -1, createdAt: -1 })
    .skip(parseInt(skip))
    .limit(parseInt(limit))
    .toArray();

  return c.json({
    success: true,
    gigs,
    total: await db.gigs.countDocuments(query)
  });
});

app.get('/api/badges/:address', async (c) => {
  const { address } = c.req.param();

  // Read from Avalanche blockchain
  const provider = new ethers.JsonRpcProvider(process.env.AVALANCHE_RPC_URL);
  const badgeContract = new ethers.Contract(
    process.env.BADGE_CONTRACT_ADDRESS,
    BADGE_ABI,
    provider
  );

  // Get badge IDs
  const badgeIds = await badgeContract.getBadges(address);

  // Get details for each badge
```

```
  const badges = await Promise.all(
    badgeIds.map(async (id) => {
      const details = await badgeContract.getBadgeDetails(id);
      return {
        tokenId: id.toString(),
        skillName: details.skillName,
        iconURI: details.iconURI,
        level: details.level,
        issuedAt: new Date(Number(details.issuedAt) * 1000).toISOString()
      };
    })
  );

  return c.json({
    success: true,
    address,
    badges
  });
});

// Health check
app.get('/health', (c) => {
  return c.json({
    status: 'healthy',
    timestamp: new Date().toISOString(),
    network: 'avalanche-fuji',
    x402: 'enabled'
  });
});

// Start server
const port = process.env.PORT || 3000;
console.log(`🚀 Server running on http://localhost:${port}`);
console.log(`💰 x402 payments on Avalanche Fuji`);
console.log(`📍 Payment address: ${X402_CONFIG.paymentAddress}`);

export default app;
```

Step 3: Frontend Integration (React + x402-client)

javascript

```javascript
// hooks/useX402Payment.js
import { useState } from 'react';
import { X402Client } from 'x402-client';
import { useWalletClient } from 'wagmi';

export function useX402Payment() {
  const { data: walletClient } = useWalletClient();
  const [isProcessing, setIsProcessing] = useState(false);
  const [error, setError] = useState(null);

  const makePayment = async (endpoint, data, config = {}) => {
    if (!walletClient) {
      throw new Error('Wallet not connected');
    }

    setIsProcessing(true);
    setError(null);

    try {
      // Initialize x402 client
      const x402Client = new X402Client({
        signer: walletClient,
        facilitatorUrl: 'https://facilitator.ultravioletadao.xyz',
        network: 'avalanche-fuji'
      });

      // First attempt - no payment
      let response = await fetch(`${import.meta.env.VITE_API_URL}${endpoint}`, {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify(data)
      });

      // Server requires payment (HTTP 402)
      if (response.status === 402) {
        const paymentRequirements = await response.json();

        // Show confirmation modal if callback provided
        if (config.onPaymentRequired) {
          const confirmed = await config.onPaymentRequired(paymentRequirements);
          if (!confirmed) {
            setIsProcessing(false);
            return null;
          }
        }
```

```
      // Make payment with x402
      response = await x402Client.post(endpoint, {
        data,
        paymentRequirements
      });
    }

    const result = await response.json();
    setIsProcessing(false);
    return result;

  } catch (err) {
    setError(err.message);
    setIsProcessing(false);
    throw err;
  }
 };

 return { makePayment, isProcessing, error };
}
```

javascript

```
// components/FeatureGigButton.jsx
import { useState } from 'react';
import { useX402Payment } from '../hooks/useX402Payment';

export function FeatureGigButton({ gigId }) {
 const { makePayment, isProcessing } = useX402Payment();
 const [showModal, setShowModal] = useState(false);

 const handleFeatureGig = async () => {
  try {
   const result = await makePayment(
    '/api/gigs/featured',
    { gigId },
    {
     onPaymentRequired: async (requirements) => {
      // Show confirmation modal
      return new Promise((resolve) => {
       setShowModal({
        amount: requirements.price,
        description: requirements.description,
        onConfirm: () => {
         setShowModal(false);
         resolve(true);
        },
        onCancel: () => {
         setShowModal(false);
```

```
            resolve(false);
          }
        });
      });
    }
  }
);

  if (result?.success) {
    alert('Gig featured! ✨');
  }
 } catch (error) {
  console.error('Payment failed:', error);
  alert('Payment failed: ' + error.message);
 }
};

 return (
  <>
   <button
    onClick={handleFeatureGig}
    disabled={isProcessing}
    className="bg-purple-600 hover:bg-purple-700 text-white px-6 py-2 rounded-lg
          disabled:opacity-50 disabled:cursor-not-allowed"
   >
    {isProcessing ? 'Processing...' : '⭐ Feature Gig ($0.50)'}
   </button>

   {showModal && (
    <PaymentModal
     amount={showModal.amount}
     description={showModal.description}
     onConfirm={showModal.onConfirm}
     onCancel={showModal.onCancel}
    />
   )}
  </>
 );
}

function PaymentModal({ amount, description, onConfirm, onCancel }) {
 return (
  <div className="fixed inset-0 bg-black bg-opacity-50 flex items-center justify-center z-50">
   <div className="bg-white rounded-lg p-8 max-w-md w-full">
    <h3 className="text-2xl font-bold mb-4">Confirm Payment</h3>
    <div className="mb-6 space-y-2">
     <p className="text-gray-600">{description}</p>
     <div className="flex items-center justify-between bg-purple-50 p-4 rounded">
      <span className="font-semibold">Amount:</span>
```

Step 4: Environment Configuration

bash

```
# .env (Backend)
PORT=3000
NODE_ENV=development

# Avalanche Fuji Testnet
AVALANCHE_RPC_URL=https://api.avax-test.network/ext/bc/C/rpc
CHAIN_ID=43113

# Smart Contracts (after deployment)
ESCROW_CONTRACT_ADDRESS=0x...
BADGE_CONTRACT_ADDRESS=0x...

# x402 Configuration
X402_PAYMENT_ADDRESS=0x...  # Your wallet address (receives USDC)
X402_FACILITATOR_URL=https://facilitator.ultravioletadao.xyz

# USDC Token on Fuji
USDC_CONTRACT_ADDRESS=0x5425890298aed601595a70AB815c96711a31Bc65

# Admin Wallet (for minting badges)
ADMIN_PRIVATE_KEY=0x...
ADMIN_ADDRESS=0x...

# Database
MONGODB_URI=mongodb://localhost:27017/projectx
```

bash

```
# .env (Frontend)
VITE_API_URL=http://localhost:3000
VITE_AVALANCHE_RPC_URL=https://api.avax-test.network/ext/bc/C/rpc
VITE_CHAIN_ID=43113
VITE_ESCROW_ADDRESS=0x...
VITE_BADGE_ADDRESS=0x...
```

Step 5: Test x402 Payment Flow

```bash
# Terminal 1: Start backend
npm run dev

# Terminal 2: Start frontend
cd frontend && npm run dev

# Terminal 3: Test with curl
curl -X POST http://localhost:3000/api/gigs/featured \
  -H "Content-Type: application/json" \
  -d '{"gigId":"123","title":"Test Gig"}'

# Should return:
# {
#   "error": "Payment required",
#   "price": "$0.50",
#   "paymentAddress": "0x...",
#   "facilitator": "https://facilitator.ultravioletadao.xyz"
# }
```

🎬 Complete Demo Walkthrough

Pre-Demo Setup (5 minutes before)

```bash
# 1. Check all services running
✓ Backend: http://localhost:3000/health
✓ Frontend: http://localhost:5173
✓ MongoDB: Connected
✓ Contracts deployed on Fuji

# 2. Prepare demo wallets
Employer Wallet:
- Address: 0xAAA...
- AVAX Balance: 2.0 AVAX (for escrow + gas)
- USDC Balance: 10.0 USDC (for x402 payments)

Worker Wallet:
- Address: 0xBBB...
- AVAX Balance: 0.5 AVAX (for gas)
- Has Badge: "Web Development" (token ID: 1)

# 3. Clear browser cache
# 4. Have backup video ready
```

Demo Script (90 seconds)

[0:00-0:15] Introduction

> "Traditional gig platforms like Upwork take 20% commissions and have fake reviews.
> ProjectX solves this with x402 micropayments and blockchain escrow on Avalanche."

[0:15-0:35] Show Dashboard

1. Connect wallet → MetaMask → Avalanche Fuji
2. Show wallet address: 0xAAA...
3. Show balances: 2.0 AVAX, 10 USDC
4. Show badge: "Web Development" badge NFT

[0:35-0:60] Create & Feature Gig

1. Click "Post a Gig"
2. Fill form:
   - Title: "Build Landing Page"
   - Payment: 0.5 AVAX
   - Required Badge: Web Development
3. Click "Create Gig" → MetaMask signs → Gig created
4. Click "Feature Gig ($0.50)"
5. Payment modal appears
6. Click "Pay with USDC" → MetaMask signs
7. Success notification → "Gig featured! ✨"
8. Show gig now at top of list with star

[0:60-0:75] Lock Escrow

1. Click "Lock Payment in Escrow"
2. MetaMask confirms: Send 0.5 AVAX to contract
3. Transaction confirmed
4. Show on Snowtrace: AVAX locked in contract

[0:75-0:90] Show Revenue

1. Switch to admin dashboard
2. Show revenue tracker: +$0.50 USDC earned
3. Show transaction on Snowtrace
4. "Platform earned $0.50 vs Upwork's $100 commission"

🏆 Competitive Advantages Summary

Feature	Traditional (Upwork)	ProjectX on Avalanche
Commission	20% ($100 on $500 gig)	$0.50 micropayment
Settlement	7-14 days (bank transfer)	2 seconds (blockchain)
Reviews	Text (fakeable)	NFT badges (verifiable)
Payment Method	Credit card (2.9% fee)	USDC via x402 (0% fee)
Escrow	Platform holds funds	Smart contract (trustless)
Skill Verification	None	On-chain NFT badges
Geographic Limits	Yes (payment rails)	No (global blockchain)
Platform Risk	High (can freeze funds)	Low (code is law)

✅ Final Checklist Before Submission

Technical Requirements

- [ ] Smart contracts deployed to Avalanche Fuji
- [ ] Contracts verified on Snowtrace
- [ ] x402 payments working with testnet USDC
- [ ] Frontend connects to Avalanche only
- [ ] Demo completes in under 2 minutes
- [ ] Backup video recorded
- [ ] All transactions visible on Snowtrace

Documentation

- [ ] README with setup instructions
- [ ] Contract addresses documented
- [ ] API endpoints documented
- [ ] Video demo uploaded
- [ ] Pitch deck completed (5 slides max)

Demo Preparation

- [ ] Wallets funded with AVAX + USDC
- [ ] Badge NFT minted to demo wallet
- [ ] Database seeded with sample gigs
- [ ] Team roles assigned for pitch
- [ ] Practice demo 10+ times

🚀 Deployment Instructions

Deploy Contracts to Fuji

bash

```
# Install Hardhat
npm install --save-dev hardhat @nomicfoundation/hardhat-toolbox

# Create deployment script
# scripts/deploy.js
const hre = require("hardhat");

async function main() {
  // Deploy Escrow
  const Escrow = await hre.ethers.getContractFactory("GigEscrow");
  const escrow = await Escrow.deploy();
  await escrow.waitForDeployment();
  console.log("Escrow deployed to:", await escrow.getAddress());

  // Deploy Badge NFT
  const Badge = await hre.ethers.getContractFactory("SkillBadge");
  const badge = await Badge.deploy();
  await badge.waitForDeployment();
  console.log("Badge deployed to:", await badge.getAddress());
}

main().catch((error) => {
  console.error(error);
  process.exitCode = 1;
});

# Run deployment
npx hardhat run scripts/deploy.js --network fuji

# Verify on Snowtrace
npx hardhat verify --network fuji <CONTRACT_ADDRESS>
```

Deploy Backend (Railway.app)

```bash
# Install Railway CLI
npm install -g @railway/cli

# Login
railway login

# Initialize project
railway init

# Add environment variables
railway variables set AVALANCHE_RPC_URL="https://api.avax-test.network/ext/bc/C/rpc"
railway variables set X402_PAYMENT_ADDRESS="0x..."

# Deploy
railway up
```

Deploy Frontend (Vercel)

```bash
# Install Vercel CLI
npm install -g vercel

# Deploy
cd frontend
vercel

# Set environment variables in Vercel dashboard
VITE_API_URL=https://your-backend.railway.app
VITE_AVALANCHE_RPC_URL=https://api.avax-test.network/ext/bc/C/rpc
```

🎓 Key Takeaways for Judges

1. **All-Avalanche Architecture**
   - "We chose Avalanche because it's the only chain that can handle both large escrow payments AND micropayments at scale"
2. **x402 Integration**
   - "x402 enables us to charge $0.50 instead of 20% commissions—that's 40x cheaper for users"
3. **Real Revenue Model**
   - "Most hackathon projects have no business model. We generated $0.50 in this demo. At scale, that's $34k/month"
4. **Technical Execution**
   - "Two smart contracts, x402 micropayments, NFT badges—all working live on Avalanche"
5. **Future Vision**
   - "We'll launch our own Avalanche L1 where badge holders become validators"

You're Ready. Go Build! 🚀

Last Updated: December 7, 2025 Submission Deadline: December 8, 2025, 12:00 PM EST