AI-Powered OSINT Platform Development Guide

Phase 1: Strategic Planning & Architecture (Use ChatGPT Plus + Claude)

Step 1: Technical Architecture Refinement

Tool: ChatGPT Plus (GPT-4) **Process**:

1. Feed your entire charter to ChatGPT
2. Ask: "Create a detailed technical architecture plan for this OSINT platform, focusing on the MVP features. Include database schemas, API endpoints, and component hierarchy."
3. Have it ask clarifying questions about your technical preferences
4. Ask it to critique its own plan and regenerate
5. Create `instructions.md` with the final plan

Sample Prompt:

```
You are a senior software architect. Based on this project charter, create a comprehensive technical plan for the MVP phase. Ask me clarifying questions about:
- Frontend framework preferences (React/Vue/Svelte)
- Backend technology (Node.js/Python/Go)
- Database choices (PostgreSQL + PostGIS vs MongoDB)
- Real-time solution (WebSockets vs Server-Sent Events)
- Authentication strategy
- File storage approach

After my answers, critique your own plan for potential issues and regenerate.
```

Step 2: Development Roadmap Creation

Tool: Claude Pro **Process**:

1. Take the technical plan from ChatGPT
2. Ask Claude to break it into 2-week sprints with specific deliverables
3. Create test-driven development approach for each feature
4. Generate user stories with acceptance criteria

Phase 2: Setup & Foundation (Use Cursor Pro + Claude)

Step 3: Project Initialization

Tool: Cursor Pro **Setup**:

1. **Create** `.cursorrules`:

```
# OSINT Platform Development Rules
- Always write tests first (TDD approach)
- Use TypeScript for type safety
- Follow the project plan in instructions.md
- Write code incrementally in small, testable chunks
- Run tests after each change
- Use semantic commit messages
- Keep components small and focused
- Prioritize accessibility (WCAG 2.1 AA)
- Implement security best practices from day one
- Use environment variables for all config
```

2. **Create** `instructions.md` with your technical plan from Step 1
3. **Create** `.cursorignore`:

```
node_modules/
.git/
dist/
build/
coverage/
.env*
logs/
*.log
.DS_Store
```

Step 4: Core Architecture Setup

Tool: Cursor Composer Agent **Process**:

1. **First Increment - Project Structure**:

```
Create the initial project structure for a OSINT platform with:
- Frontend: React + TypeScript + Vite
- Backend: Node.js + Express + TypeScript
- Database: PostgreSQL + PostGIS
- Real-time: Socket.io
- Authentication: JWT + bcrypt
- Testing: Vitest + Testing Library

Follow TDD. Create failing tests first, then implement.
```

2. **Second Increment - Database Schema**:

```
@instructions.md
Implement the database schema for users, posts, and basic geolocation.
Write migration scripts and seed data for testing.
Include tests for all database operations.
```

Phase 3: Feature Development (Cursor + Claude/ChatGPT Hybrid)

Step 5: Feature-by-Feature Development

For each feature:

1. **Planning with ChatGPT**:
   - "I need to implement [specific feature]. Create detailed implementation instructions for another AI coding assistant."
   - Get comprehensive technical specifications
2. **Implementation with Cursor**:
   - Paste ChatGPT's instructions into Cursor Composer
   - Use `@instructions.md` frequently for context
   - Keep iterations small (Edit → Test → Verify loop)

Example Feature Development Flow:

User Authentication System

1. **ChatGPT Planning Prompt**:

```
Create detailed implementation instructions for another AI to build a JWT-based
authentication system for a OSINT platform. Include:
- Registration/login endpoints
- Password hashing with bcrypt
- JWT token generation/validation
- Protected route middleware
- Rate limiting for security
- Input validation
- Error handling
- Complete test suite

Provide step-by-step instructions that another AI can follow.
```

2. **Cursor Implementation**:

```
@instructions.md
Implement the user authentication system following these detailed instructions:
[Paste ChatGPT's instructions]

Follow TDD: write failing tests first, then implement features to pass tests.
```

Step 6: 3D Globe Integration

Special Consideration: This is complex - break into micro-increments

1. **Research Phase** (Perplexity Pro):
   - "Latest Three.js best practices for geographic data visualization 2024"
   - "Globe.gl vs custom Three.js implementation performance comparison"
   - "WebGL performance optimization techniques for large datasets"
2. **Planning Phase** (Claude Pro):

```
Based on this research, create a step-by-step implementation plan for integrating a 3D globe
with real-time OSINT post visualization. Break this into the smallest possible testable
increments.
```

3. **Implementation Phase** (Cursor):
   - Start with static globe
   - Add single point plotting
   - Add point clustering

- Add real-time updates
- Add interaction handlers

Phase 4: Advanced Features & Optimization

Step 7: Community Notes System

High Priority - Core Differentiator

1. **Algorithm Design** (Claude Pro):

```
Design the community notes consensus algorithm similar to Twitter's Community Notes but
optimized for OSINT verification. Include:
- Voting mechanics
- Bias detection
- Reputation weighting
- Threshold calculations
- Abuse prevention
```

2. **Implementation** (Cursor + previous pattern)

Step 8: Real-time Features

Tools: Cursor + Claude for WebSocket management

1. **Performance Testing Setup**:

```
Create load testing scripts for WebSocket connections.
Test with 100+ concurrent users receiving real-time updates.
Implement performance monitoring.
```

Phase 5: Quality Assurance & Deployment

Step 9: Security Audit

Tool: Claude Pro for security review

```
Review this codebase for security vulnerabilities:
[Use gitingest.com to provide full codebase]

Focus on:
- Authentication/authorization flaws
- SQL injection possibilities
- XSS vulnerabilities
- Rate limiting effectiveness
- Input validation completeness
```

Step 10: Performance Optimization

Tool: Perplexity Pro for research + Cursor for implementation

- Research latest performance optimization techniques
- Implement code splitting, lazy loading
- Optimize database queries
- Set up CDN and caching

Ongoing Development Best Practices

Context Management

- **Start new Cursor chats** when context exceeds ~50 files
- **Use @ explicitly** to add specific files rather than relying on auto-context
- **Commit frequently** to git - don't accumulate large uncommitted changes
- **Resync/index** code in Cursor regularly

Problem-Solving Workflow

When stuck:

1. **Ask Cursor** to write a comprehensive report of all files and problems
2. **Use gitingest.com** to package the relevant code
3. **Ask Claude/ChatGPT** for solution recommendations
4. **Implement solutions** incrementally in Cursor

Prompt Strategies

- **Chain of thought**: "Explain your reasoning before implementing"
- **Incremental development**: "Implement only the user registration endpoint, write tests first"

- **Error-driven development**: "The test is failing because X, fix the minimal code needed"

Resource Management

Claude Pro Token Conservation

- Use for **high-level architecture** and **complex algorithm design**
- Save tokens by **preparing detailed prompts** offline first
- Use for **final code reviews** and **security audits**

ChatGPT Plus Usage

- Primary tool for **feature planning** and **instruction generation**
- Use for **breaking down complex requirements**
- Leverage for **documentation generation**

Cursor Pro Optimization

- Enable **YOLO mode** for automatic test writing
- Use **Composer Agent** for multi-file changes
- Utilize **Notepads** for frequently used prompts

Perplexity Pro for Research

- **Latest technology trends** and best practices
- **Security vulnerability research**
- **Performance optimization techniques**
- **Competitive analysis**

Sample Development Timeline

Week 1-2: Foundation

- Project setup and basic structure
- Database schema and migrations
- Basic authentication system

Week 3-4: Core Features

- Post creation and retrieval
- Basic map integration (2D first)

- User profiles and basic social features

Week 5-6: Geospatial Features

- PostGIS integration
- Location-based queries
- Basic globe visualization

Week 7-8: Real-time Features

- WebSocket implementation
- Live post updates
- Basic notification system

Week 9-10: Community Features

- Community notes system
- Basic reputation system
- Voting mechanisms

Week 11-12: Polish & Deploy

- Security hardening
- Performance optimization
- MVP deployment

Success Metrics for AI-Assisted Development

- **Development Speed**: Target 2x faster than traditional development
- **Code Quality**: 90%+ test coverage, minimal security issues
- **Bug Reduction**: TDD approach should reduce post-deployment bugs by 70%
- **Feature Completeness**: MVP features delivered within 12-week timeline

Emergency Protocols

When AI Gets Stuck

1. **Context Reset**: Start fresh Cursor chat with minimal context
2. **Problem Decomposition**: Break the stuck feature into smaller pieces
3. **Expert Consultation**: Use Claude Pro for high-level problem solving

4. **Alternative Approach**: Research different implementation strategies with Perplexity

When Performance Issues Arise

1. **Profiling First**: Use browser dev tools and backend profiling
2. **Research Solutions**: Perplexity Pro for optimization techniques
3. **Incremental Optimization**: Small, measurable improvements
4. **Load Testing**: Validate improvements under realistic conditions

This guide should maximize your AI tool efficiency while maintaining code quality and meeting your ambitious timeline goals.