**Samriddh AI: Project Charter**

**1. Project Title and Vision Statement**

**Title:** Samriddh AI - Personal Enterprise-Grade Development Companion

**Vision Statement:** To create an AI-powered development companion that enables individual developers to build production-ready, enterprise-grade applications through intelligent project planning, automated quality assurance, and cross-project learning, without the complexity of authentication or team collaboration overhead.

**2. Problem Statement**

The primary problems this platform solves are:

* **Development Velocity Bottleneck:** Individual developers spend 70-80% of their time on non-creative tasks like boilerplate code generation, debugging, testing setup, and configuration management, severely limiting their ability to focus on business logic and innovation.
* **Enterprise-Quality Standards Gap:** Personal and small-scale projects often lack enterprise-grade features (audit logging, comprehensive error handling, performance optimization, accessibility compliance) due to the complexity and time investment required to implement these standards manually.
* **Project Planning and Architecture Inefficiency:** Developers frequently dive into coding without proper planning, leading to architectural debt, scope creep, and project failures, with 60% of personal projects abandoned due to poor initial planning and overwhelming technical complexity.

**3. Target User Personas and Value Proposition**

| **Persona** | **Description** | **Primary Goal** | **Value Proposition** |
| --- | --- | --- | --- |
| **Solo Developer Sam** | An experienced developer working on personal projects, side businesses, or freelance work who values quality and wants to build professional-grade applications efficiently. | To rapidly create production-ready applications with enterprise-quality standards without spending weeks on setup and boilerplate code. | "An AI companion that handles project planning, architecture design, and code generation, allowing you to focus on business logic while ensuring enterprise-grade quality automatically." |
| **Technical Entrepreneur Emma** | A technical founder or entrepreneur who needs to build MVPs and business applications quickly while maintaining professional standards for potential scaling or investment. | To validate business ideas rapidly by building functional, scalable prototypes that can transition smoothly to production systems. | "Accelerate your product development from idea to market-ready application in days instead of months, with built-in scalability and professional architecture." |
| **Learning Developer Leo** | A developer seeking to improve their skills by studying enterprise-grade code patterns and architecture while building real projects. | To learn best practices and enterprise patterns through hands-on experience with AI-generated, high-quality code examples. | "Learn enterprise development patterns through practical implementation, with AI explanations and cross-project pattern recognition to accelerate your professional growth." |

**4. Core Features & Prioritization (MVP Scope)**

| **Feature** | **Description** | **Priority** |
| --- | --- | --- |
| **Structured Project Planning System** | Multi-phase planning workflow (Project Charter → Architecture → Implementation Plan) with AI-guided requirements gathering and validation. | P0 - Must-Have |
| **Dual AI Model Integration** | GLM 4.5 (thinking mode) and Claude Sonnet 4 Thinking with user-controlled provider selection for different development tasks. | P0 - Must-Have |
| **Production-Ready Code Generation** | Complete application generation with enterprise features (audit logging, error handling, testing, documentation) built-in by default. | P0 - Must-Have |
| **SQLite + Alembic Data Management** | Automatic database setup with migration management, data integrity constraints, and performance optimization for generated applications. | P0 - Must-Have |
| **VS Code Bridge Integration** | Lightweight extension providing seamless workspace integration, file operations, and terminal access for the standalone AI application. | P1 - Important |
| **Automated Quality Assurance** | Playwright-powered visual testing, accessibility compliance checking, and comprehensive test suite generation with validation loops. | P1 - Important |
| **Cross-Project Learning System** | Vector-based memory system that learns from successful patterns across projects and suggests reusable solutions and architectures. | P2 - Nice-to-Have |

**5. Key Success Metrics (KPIs)**

* **Development Velocity:** Reduce application development time from weeks/months to days (target: 80% time reduction for typical business applications).
* **Code Quality Standards:** Achieve >95% automated test coverage and enterprise-grade code quality metrics for all generated applications.
* **User Productivity:** Measure time-to-first-working-application (target: <4 hours from project idea to functional prototype).
* **Cross-Project Learning Effectiveness:** Track pattern reuse across projects and measure improvement in suggestion accuracy over time (target: 85% pattern reuse rate after 10 projects).

**6. Explicit Non-Goals (Out of Scope for MVP)**

* **Multi-User Authentication System:** The platform will have direct access without login requirements, user management, or permission systems.
* **Team Collaboration Features:** No real-time collaboration, shared workspaces, or multi-user project management capabilities.
* **Enterprise Security & Compliance:** No advanced security frameworks, role-based access control, or enterprise compliance certifications (SOC 2, HIPAA, etc.).
* **Template-Based Development:** No pre-built application templates; each project will go through the complete development planning and generation cycle.
* **Mobile Application Development:** The MVP will focus on web applications and desktop tools; mobile app generation is out of scope.
* **Subscription or Licensing Management:** No payment processing, subscription tiers, or license key management systems.

**7. Key Assumptions and Risks**

**Assumptions:**

* **AI Model Availability:** GLM 4.5 with thinking mode and Claude Sonnet 4 Thinking will remain accessible and provide consistent quality output for code generation and project planning tasks.
* **Individual Developer Adoption:** Solo developers and technical entrepreneurs will value AI-assisted development that maintains their complete control over the development process while accelerating productivity.

**Risks:**

* **Risk:** "AI-generated code quality inconsistency could lead to production issues and user trust erosion."  
  **Mitigation:** "Implement comprehensive automated testing, validation loops, and quality gates that ensure every generated application meets enterprise standards before delivery. Include rollback mechanisms and manual override options for critical decisions."
* **Risk:** "Over-reliance on AI could reduce developer learning and skill development."  
  **Mitigation:** "Design the system to be educational, providing detailed explanations for architectural decisions, highlighting best practices, and encouraging developers to understand and customize the generated code rather than treating it as a black box."

**Samriddh AI: Complete Product Features & Capabilities**

**1. Core AI Intelligence & Provider Management**

**Dual AI Model Architecture**

* **GLM 4.5 (Primary Model)**
  + Always-enabled thinking mode for enhanced reasoning
  + Specialized for enterprise application development and complex business logic
  + Optimized for structured project planning and architectural decisions
  + Superior performance in code generation with enterprise-grade patterns
* **Claude Sonnet 4 Thinking (Secondary Model)**
  + Advanced reasoning capabilities for complex problem-solving
  + Specialized in code analysis, debugging, and optimization
  + Excellent for detailed code reviews and architectural analysis
  + Enhanced capability for explaining complex technical concepts

**User-Controlled Provider Selection**

* Manual provider selection based on task requirements and user preference
* Seamless switching between models without losing conversation context
* Provider-specific optimization for different development phases
* Real-time provider performance tracking and comparison
* Intelligent suggestions for optimal provider selection based on task type

**Cross-Model Context Preservation**

* Seamless context transfer when switching between AI providers
* Unified conversation history across different models
* Consistent project understanding regardless of active provider
* Context-aware provider recommendations based on current development phase

**2. Structured Project Planning & Development Lifecycle**

**Phase-Based Development Methodology**

Based on the comprehensive prompt-driven planning system:

**Phase 0: Strategic Project Charter**

* **Interactive Requirements Gathering**
  + Guided questionnaire system for comprehensive requirement collection
  + Business logic definition and validation workflows
  + Stakeholder persona mapping and value proposition analysis
  + Success metrics definition (KPIs) and measurement frameworks
* **Project Scope Definition**
  + Feature prioritization with MVP scope determination
  + Explicit non-goals documentation to prevent scope creep
  + Risk assessment with mitigation strategy development
  + Timeline estimation based on complexity analysis

**Phase 1: Architectural Blueprint**

* **System Architecture Design**
  + Technology stack selection with detailed justification
  + Database schema design with relationship mapping
  + API structure planning with endpoint specification
  + Integration architecture for third-party services
* **Technical Decision Framework**
  + Architecture pattern selection (microservices, monolith, modular)
  + Performance requirements and scalability planning
  + Security considerations and implementation strategies
  + Deployment architecture and infrastructure planning

**Phase 2.0: Project Foundation & Core Contracts**

* **Foundational Type Definitions**
  + Comprehensive TypeScript type system establishment
  + Shared utility types and interfaces definition
  + API contract scaffolding with OpenAPI 3.0 specifications
  + Core environment variable manifest and configuration management
* **Coding Standards & Conventions**
  + Project-wide coding standards establishment
  + Error handling patterns and validation frameworks
  + Asynchronous programming patterns and best practices
  + Documentation standards and commenting conventions

**Phase 2.1+: Module Specifications**

* **Detailed Module Planning**
  + Module-specific type definitions and interfaces
  + Business logic documentation with validation rules
  + API endpoint specifications with full OpenAPI paths
  + Developer task checklists with granular implementation steps
* **Inter-Module Dependencies**
  + Dependency mapping and relationship analysis
  + Integration point definitions and contracts
  + Data flow analysis between modules
  + Testing strategy for module interactions

**Phase 3.0: Project Scaffolding & Infrastructure**

* **Complete Project Structure Generation**
  + Monorepo or multi-repo structure with proper organization
  + Configuration files (package.json, tsconfig.json, etc.)
  + Development environment setup with Docker support
  + CI/CD pipeline templates and deployment configurations
* **Core Infrastructure Components**
  + Database setup with SQLite and Alembic migrations
  + API server configuration with FastAPI/Express
  + Frontend application setup with React/Next.js
  + Testing framework integration and configuration

**Phase 3.1+: Implementation Execution**

* **File-by-File Generation**
  + Ordered implementation plan respecting dependencies
  + Context-aware code generation with cross-file consistency
  + Real-time validation and quality assurance
  + Progressive enhancement and iterative development

**3. Production-Ready Application Generation**

**Enterprise-Grade Application Features (Built-in)**

Every generated application automatically includes:

**Data Management & Persistence**

* **SQLite + Alembic Integration**
  + Automatic database schema generation with proper constraints
  + Migration management with version control and rollback capabilities
  + Data integrity enforcement with foreign keys and unique constraints
  + Performance optimization with automatic indexing strategies
  + Connection pooling and query optimization for scalability
* **Advanced Data Features**
  + Comprehensive input validation and sanitization
  + Audit trail system with automatic change tracking
  + Data backup automation with scheduled exports
  + Transaction management with rollback capabilities
  + Query performance monitoring and optimization suggestions

**API & Backend Architecture**

* **RESTful API Development**
  + Complete CRUD operations with proper HTTP methods
  + Input validation with detailed error responses
  + Rate limiting and request throttling
  + API versioning and backward compatibility
  + Comprehensive OpenAPI 3.0 documentation generation
* **Enterprise Backend Features**
  + Structured error handling with user-friendly messages
  + Comprehensive logging system with structured format
  + Performance monitoring with metrics collection
  + Health check endpoints with system status reporting
  + Configuration management with environment-specific settings

**Frontend & User Experience**

* **Modern Web Application Development**
  + Responsive design with mobile-first approach
  + Progressive Web App (PWA) capabilities
  + WCAG 2.1 AA accessibility compliance
  + Internationalization (i18n) support with multiple languages
  + SEO optimization with meta tags and structured data
* **Advanced UI Features**
  + Real-time data updates with WebSocket integration
  + Advanced form validation with custom rules
  + File upload/download with progress tracking
  + Data visualization with charts and graphs
  + Export capabilities (PDF, Excel, CSV)

**Quality Assurance & Testing**

* **Comprehensive Test Suite Generation**
  + Unit tests with >95% code coverage
  + Integration tests for API endpoints and database operations
  + End-to-end tests with Playwright automation
  + Performance tests with load testing scenarios
  + Accessibility tests with automated WCAG compliance checking
* **Code Quality Assurance**
  + Automated linting with ESLint/Prettier configuration
  + Type checking with strict TypeScript compilation
  + Security scanning with vulnerability detection
  + Code complexity analysis with maintainability metrics
  + Documentation coverage validation

**4. Advanced Visual Review & Quality Assurance**

**Playwright MCP Integration**

* **Automated Visual Testing**
  + Screenshot comparison with pixel-perfect accuracy
  + Visual regression testing across browser versions
  + Cross-device responsive design validation
  + UI component testing with interaction scenarios
  + Performance testing with Core Web Vitals monitoring
* **Accessibility & Compliance Testing**
  + WCAG 2.1 AA/AAA compliance validation
  + Keyboard navigation testing
  + Screen reader compatibility verification
  + Color contrast ratio validation
  + Focus management and tab order testing

**Automated Improvement System**

* **AI-Powered UX Analysis**
  + User interface usability assessment
  + Navigation flow optimization suggestions
  + Performance bottleneck identification
  + Accessibility improvement recommendations
  + Mobile optimization suggestions
* **Continuous Quality Monitoring**
  + Performance regression detection
  + Accessibility compliance monitoring
  + Visual consistency validation
  + User experience metrics tracking
  + Automated improvement suggestions

**5. Intelligent Debugging & Error Resolution**

**Automated Error Detection & Correction**

* **Comprehensive Validation Suite**
  + Real-time syntax and type error detection
  + Logic error identification with context analysis
  + Runtime error prediction and prevention
  + Performance bottleneck detection
  + Security vulnerability scanning
* **AI-Powered Error Resolution**
  + Automatic error fix generation with explanation
  + Multiple solution approaches with trade-off analysis
  + Context-aware debugging assistance
  + Root cause analysis with dependency tracking
  + Validation loops to ensure fix quality

**Quality Assurance Workflow**

* **Validate-Document-Correct Loop**
  + Systematic error documentation and categorization
  + AI-generated fix implementation with validation
  + Comprehensive testing after error resolution
  + Learning from successful corrections for future improvements
  + Quality gate enforcement before deployment
* **Development Workflow Integration**
  + Pre-commit hooks with automated validation
  + Continuous integration with quality checks
  + Automated testing pipeline with failure analysis
  + Code review assistance with improvement suggestions
  + Performance monitoring with optimization recommendations

**6. Cross-Project Learning & Knowledge Management**

**Vector-Based Memory System**

* **Project Pattern Recognition**
  + Successful architecture pattern identification and storage
  + Code solution reuse across similar problem domains
  + Best practice extraction from completed projects
  + Anti-pattern detection and avoidance suggestions
  + Performance optimization pattern learning
* **Intelligent Knowledge Transfer**
  + Cross-project solution suggestion based on similarity
  + Architectural decision learning and recommendation
  + Technology stack optimization based on project success
  + Development pattern evolution and improvement
  + Personal coding style learning and adaptation

**Dynamic Template Generation**

* **Project-Based Template Creation**
  + Successful project conversion to reusable patterns
  + Customizable project templates based on personal history
  + Architecture template generation from completed projects
  + Component library creation from successful implementations
  + Configuration template sharing across projects
* **Adaptive Learning System**
  + User preference learning and adaptation
  + Coding style pattern recognition and enforcement
  + Project success factor analysis and application
  + Technology stack preference learning
  + Development workflow optimization based on productivity metrics

**7. VS Code Integration & Development Environment**

**Seamless VS Code Bridge**

* **Native Workspace Integration**
  + Direct file system access without external APIs
  + Real-time workspace monitoring and context awareness
  + Git integration with change tracking and conflict resolution
  + Terminal access with command execution and output parsing
  + Project structure analysis and dependency mapping
* **Enhanced Development Experience**
  + Intelligent code completion with context awareness
  + Inline error detection and fix suggestions
  + Hover documentation with AI-generated explanations
  + Code lens actions for optimization and refactoring
  + Real-time code quality feedback and suggestions

**Development Workflow Automation**

* **Automated Development Tasks**
  + Dependency installation and management
  + Database migration execution and rollback
  + Test execution with result analysis
  + Build process automation with error handling
  + Deployment preparation and validation
* **Context-Aware Assistance**
  + Current file analysis with improvement suggestions
  + Project-wide impact analysis for code changes
  + Refactoring assistance with safety checks
  + Documentation generation and maintenance
  + Code review automation with quality assessments

**8. Data Management & Scalability**

**SQLite Production Optimization**

* **Advanced SQLite Configuration**
  + Write-Ahead Logging (WAL) mode for concurrent access
  + Memory-mapped I/O for improved performance
  + Connection pooling with optimal settings
  + Query optimization with automatic index suggestions
  + Backup strategies with incremental and full backups
* **Alembic Migration Management**
  + Automatic migration generation from model changes
  + Data migration support with validation
  + Rollback capabilities with data preservation
  + Migration dependency tracking and resolution
  + Schema versioning with conflict resolution

**Performance & Scalability Features**

* **Application Performance Optimization**
  + Database query optimization with execution plan analysis
  + Caching strategies with intelligent invalidation
  + Asset optimization with compression and minification
  + Lazy loading implementation for improved startup time
  + Background job processing with queue management
* **Monitoring & Analytics**
  + Application performance monitoring with metrics collection
  + Database performance analysis with slow query detection
  + User interaction tracking for UX optimization
  + Error rate monitoring with alerting
  + Resource usage tracking with optimization suggestions

**9. Security & Data Protection (Simplified)**

**Essential Security Features**

* **Basic Application Security**
  + Input validation and sanitization
  + SQL injection prevention with parameterized queries
  + XSS protection with content security policies
  + CSRF protection with token validation
  + Secure cookie handling and session management
* **Data Protection**
  + Local data encryption for sensitive information
  + Secure configuration management
  + Environment variable protection
  + File upload security with type validation
  + API rate limiting and request validation

**Privacy & Local-First Architecture**

* **Personal Data Control**
  + All data stored locally with user control
  + No external data transmission without explicit consent
  + Project data isolation and protection
  + Secure backup and restore capabilities
  + Configuration export and import with encryption

**10. User Experience & Interface Design**

**Direct Access Interface**

* **Streamlined User Experience**
  + Zero authentication barriers with immediate access
  + Intuitive project creation and management workflows
  + Real-time feedback and progress tracking
  + Context-sensitive help and documentation
  + Keyboard shortcuts and accessibility features
* **Modern Web Interface**
  + Responsive design optimized for all screen sizes
  + Dark/light mode with system preference detection
  + Customizable layout with drag-and-drop functionality
  + Real-time collaboration indicators and status updates
  + Progressive enhancement with offline capabilities

**Development-Focused UI**

* **Code-Centric Interface Design**
  + Syntax-highlighted code displays with copy functionality
  + Interactive code examples with execution capabilities
  + Side-by-side diff views for code comparisons
  + File tree navigation with project structure visualization
  + Integrated terminal with command history and autocomplete
* **Project Management Features**
  + Project dashboard with progress tracking
  + Task management with automated checklist generation
  + Timeline visualization with milestone tracking
  + Resource usage monitoring and optimization suggestions
  + Export capabilities for project documentation and reports

This comprehensive feature set positions Samriddh AI as a complete development companion that handles the entire software development lifecycle while maintaining the simplicity of direct access and personal control. Every feature is designed to enhance individual developer productivity while ensuring enterprise-grade quality in all generated applications.

**Samriddh AI: High-Level System Design (HLSD)**

**1. Executive Summary & Architectural Vision**

**Technical Architecture Summary**

The Samriddh AI architecture employs a **modular monolithic design** with specialized service modules, optimized for individual developer productivity and enterprise-grade application generation. The system features a dual AI provider architecture, structured project planning workflows, and comprehensive quality assurance automation, all built around a local-first, direct-access paradigm.

**Alignment with Vision Statement and Core Goals**

This architecture directly supports the **Vision Statement** of creating "an AI-powered development companion that enables individual developers to build production-ready, enterprise-grade applications" through:

* **Structured Planning Service**: Implements the comprehensive phase-based planning methodology to ensure proper project foundation and architecture
* **Dual AI Provider Management**: Leverages GLM 4.5 (thinking mode) and Claude Sonnet 4 Thinking for optimal task-specific AI assistance
* **Enterprise Code Generation**: Automatically produces production-ready applications with built-in enterprise features
* **Zero-Friction Access**: Eliminates authentication barriers while maintaining professional-grade development workflows
* **Cross-Project Intelligence**: Enables continuous learning and improvement across development projects

The architecture specifically addresses the **Core Goal** of "80% time reduction for typical business applications" through automated planning, code generation, and quality assurance pipelines.

**2. Core Architectural Components**

| **Component Name** | **Core Responsibilities** | **Corresponding Charter Feature(s)** |
| --- | --- | --- |
| **Project Planning Service** | Manages structured Phase 0-3.1+ planning workflows, user review cycles, and document generation with AI-guided requirements analysis | Structured Project Planning System |
| **AI Provider Management Service** | Handles dual AI model orchestration (GLM 4.5 + Claude Sonnet 4), user-controlled provider selection, and context preservation across switches | Dual AI Model Integration |
| **Code Generation Engine** | Produces production-ready applications with enterprise features, manages file generation workflows, and ensures code quality standards | Production-Ready Code Generation |
| **Data Management Service** | Manages SQLite databases with Alembic migrations, data integrity enforcement, and performance optimization for generated applications | SQLite + Alembic Data Management |
| **Quality Assurance Service** | Performs automated testing via Playwright MCP, accessibility compliance validation, and visual regression testing | Automated Quality Assurance |
| **Debugging & Validation Service** | Provides intelligent error detection, AI-powered correction suggestions, and validation loop management | VS Code Bridge Integration |
| **Knowledge & Learning Service** | Implements cross-project pattern recognition, solution reuse, and adaptive learning from successful implementations | Cross-Project Learning System |
| **VS Code Bridge Service** | Enables seamless IDE integration, workspace monitoring, and direct file system access without external APIs | VS Code Bridge Integration |
| **User Interface Service** | Provides direct-access web interface for project management, AI interaction, and development workflow orchestration | Direct access interface (derived from no-auth requirement) |

**3. High-Level Data Model**

**Core Entities and Relationships**

text

erDiagram

PROJECT ||--o{ PLANNING\_DOCUMENT : contains

PROJECT ||--o{ GENERATED\_FILE : produces

PROJECT ||--o{ PROJECT\_CONTEXT : maintains

PLANNING\_DOCUMENT {

string id PK

string project\_id FK

enum phase\_type

json content

enum status

datetime created\_at

datetime updated\_at

}

PROJECT {

string id PK

string title

text description

json requirements

enum status

string ai\_provider\_preference

datetime created\_at

datetime updated\_at

}

GENERATED\_FILE {

string id PK

string project\_id FK

string file\_path

text content

string content\_hash

enum generation\_method

json metadata

datetime created\_at

}

PROJECT\_CONTEXT {

string id PK

string project\_id FK

json workspace\_state

json ai\_conversation\_history

json learned\_patterns

datetime last\_updated

}

PATTERN\_LIBRARY ||--o{ LEARNED\_PATTERN : stores

LEARNED\_PATTERN {

string id PK

string pattern\_name

text pattern\_description

json pattern\_code

int usage\_count

float success\_rate

json metadata

datetime created\_at

}

QUALITY\_REPORT ||--|| PROJECT : validates

QUALITY\_REPORT {

string id PK

string project\_id FK

json test\_results

json performance\_metrics

json accessibility\_results

float overall\_score

datetime generated\_at

}

**Key Relationships:**

* **Projects** contain multiple **Planning Documents** representing different phases
* **Projects** generate multiple **Generated Files** with full traceability
* **Project Context** maintains state and conversation history for continuity
* **Pattern Library** stores reusable solutions across all projects for learning
* **Quality Reports** track application quality metrics and compliance

**4. Technology Stack & Strategic Rationale**

**Architectural Pattern: Modular Monolith**

**Justification:** Aligns perfectly with the **MVP scope** and **individual developer** target persona. A modular monolith provides:

* **Reduced Complexity**: Single deployment and database connection supports the "Zero-Friction Access" requirement
* **Development Velocity**: Faster iteration cycles support the **KPI** of "time-to-first-working-application < 4 hours"
* **Future Scalability**: Modular design allows evolution to microservices if needed without preventing growth
* **Risk Mitigation**: Addresses the **charter risk** of "over-engineering" while maintaining professional architecture standards

**Frontend: Next.js 14+ with React 18**

**Justification:** Directly supports the **User Productivity KPI** through:

* **Server-Side Rendering**: Improves initial load times for complex planning workflows
* **TypeScript Integration**: Ensures code quality aligned with enterprise-grade output requirements
* **Progressive Web App**: Supports offline development workflows for **Solo Developer Sam** persona
* **Component Architecture**: Modular design mirrors the backend architecture for consistency

**Backend: FastAPI with Python 3.11+**

**Justification:** Optimal for AI integration and supports **Core Features**:

* **Async Support**: Required for concurrent AI provider communication and real-time VS Code integration
* **Type Safety**: Python typing system ensures reliability needed for **Code Quality Standards KPI** (>95% automated test coverage)
* **AI Ecosystem**: Native compatibility with AI provider SDKs supports **Dual AI Model Integration** feature
* **Rapid Development**: Supports the **Development Velocity KPI** through automatic API documentation and validation

**Database: SQLite with Alembic Migrations**

**Justification:** Perfectly aligned with **charter requirements**:

* **Local-First Architecture**: No external database servers required, supporting the "personal use" and "direct access" requirements
* **Production Ready**: SQLite's ACID compliance and performance optimization support **enterprise-grade application generation**
* **Migration Management**: Alembic provides professional database versioning for generated applications
* **Zero Configuration**: Eliminates setup friction, directly supporting **User Productivity KPI**

**AI Integration: Native Provider SDKs**

**Justification:** Enables **Dual AI Model Integration** feature:

* **GLM 4.5 SDK**: Direct integration with thinking mode for enhanced reasoning
* **Anthropic Claude SDK**: Native access to Sonnet 4 thinking capabilities
* **Provider Abstraction**: Uniform interface supports seamless switching without context loss

**Testing Framework: Playwright + pytest**

**Justification:** Supports **Automated Quality Assurance** requirements:

* **Visual Testing**: Playwright MCP integration for automated UI validation
* **Cross-Browser**: Ensures generated applications work across all environments
* **Python Integration**: Seamless backend testing with pytest for comprehensive coverage

**Development Environment: VS Code Extension API**

**Justification:** Enables **VS Code Bridge Integration** feature:

* **Native File Access**: Direct workspace manipulation without external APIs
* **Real-time Integration**: Supports continuous development workflow
* **Developer Familiarity**: Leverages existing developer tools and habits

**5. API Design Philosophy**

**RESTful API with Real-time WebSocket Extensions**

**Justification:** Optimized for the **target personas** and **core features**:

* **RESTful for Planning Workflows**: The **Technical Entrepreneur Emma** persona requires structured, stateful planning processes. RESTful endpoints provide clear resource management for projects, planning phases, and generated artifacts
* **WebSocket for Real-time Feedback**: Supports **Solo Developer Sam's** need for immediate feedback during code generation and AI interactions
* **Simple Integration**: The **Learning Developer Leo** persona benefits from straightforward API patterns that are easy to understand and extend

**API Structure:**

text

/api/v1/projects/{id}/planning/{phase} # RESTful planning management

/api/v1/projects/{id}/generate # Code generation endpoints

/api/v1/ai/providers/{provider}/chat # AI provider communication

/ws/projects/{id}/live # Real-time updates via WebSocket

This design supports the **Cross-Project Learning KPI** by providing consistent data access patterns for pattern recognition across all project interactions.

**6. Authentication and Authorization Strategy**

**Direct Access Architecture (No Authentication)**

**Justification:** Explicitly aligned with **charter requirements** and **target personas**:

* **Zero Barriers**: Supports the charter's explicit non-goal of "no multi-user authentication system"
* **Individual Developer Focus**: **Solo Developer Sam** and **Technical Entrepreneur Emma** personas require immediate access without setup overhead
* **Local Security**: All data remains local to the developer's machine, eliminating external security concerns
* **Development Velocity**: Direct access supports the **User Productivity KPI** of "<4 hours from project idea to functional prototype"

**Implementation:**

python

*# No authentication middleware - direct access to all endpoints*

@app.middleware("http")

**async** **def** direct\_access\_middleware(request: Request, call\_next):

*# All requests automatically authorized for local development*

request.state.user\_id = "local\_developer"

request.state.permissions = ["all"]

response = **await** call\_next(request)

**return** response

**7. Critical Third-Party Integrations**

**Required External Services:**

| **Service** | **Purpose** | **Charter Alignment** |
| --- | --- | --- |
| **GLM 4.5 API** | Primary AI provider with thinking mode | Dual AI Model Integration - Default provider for enterprise development |
| **Anthropic Claude API** | Secondary AI provider for specialized tasks | Dual AI Model Integration - Code analysis and debugging specialization |
| **Playwright Runtime** | Automated testing and visual validation | Automated Quality Assurance - Visual regression and accessibility testing |
| **VS Code Extension API** | IDE integration and workspace access | VS Code Bridge Integration - Native file system and terminal access |

**Optional Integrations:**

* **File System Watchers**: For real-time workspace monitoring
* **Git Integration**: For version control awareness (local only)
* **Terminal Emulation**: For command execution within generated applications

**Integration Strategy:** All integrations designed for offline-first operation with graceful degradation to support the **individual developer** focus and **local-first architecture**.

**8. Architectural Plan for Scalability and KPIs**

**Performance Bottleneck Analysis and Mitigation:**

**Development Velocity KPI: 80% Time Reduction**

**Potential Bottleneck:** AI provider response times during code generation  
**Architectural Strategy:**

python

*# Concurrent AI processing with intelligent caching*

**class** AIProviderManager:

**async** **def** generate\_with\_caching(self, prompt: str, context: ProjectContext):

cache\_key = self.generate\_cache\_key(prompt, context)

**if** cached\_result := **await** self.cache.get(cache\_key):

**return** cached\_result

*# Parallel processing for multiple code sections*

results = **await** asyncio.gather(\*[

self.glm\_provider.generate(section)

**for** section **in** self.split\_prompt(prompt)

])

**return** self.merge\_results(results)

**Code Quality KPI: >95% Test Coverage**

**Potential Bottleneck:** Test generation and execution time  
**Architectural Strategy:**

python

*# Incremental testing with intelligent test selection*

**class** QualityAssuranceService:

**def** \_\_init\_\_(self):

self.test\_cache = TestCacheManager()

self.impact\_analyzer = CodeImpactAnalyzer()

**async** **def** run\_quality\_checks(self, changed\_files: List[str]):

*# Only run tests affected by changes*

affected\_tests = self.impact\_analyzer.get\_affected\_tests(changed\_files)

*# Parallel test execution*

results = **await** asyncio.gather(\*[

self.run\_test\_suite(suite) **for** suite **in** affected\_tests

])

**return** self.aggregate\_results(results)

**User Productivity KPI: <4 Hours to Functional Prototype**

**Potential Bottleneck:** Project planning and setup time  
**Architectural Strategy:**

python

*# Optimized planning workflow with intelligent defaults*

**class** ProjectPlanningService:

**def** \_\_init\_\_(self):

self.pattern\_matcher = PatternMatcher()

self.template\_generator = TemplateGenerator()

**async** **def** accelerated\_planning(self, requirements: str):

*# Use cross-project learning to suggest similar successful patterns*

similar\_projects = **await** self.pattern\_matcher.find\_similar(requirements)

*# Pre-populate planning documents with proven patterns*

planning\_template = self.template\_generator.create\_from\_patterns(

similar\_projects, requirements

)

**return** planning\_template

**Cross-Project Learning KPI: 85% Pattern Reuse Rate**

**Potential Bottleneck:** Pattern recognition and storage performance  
**Architectural Strategy:**

python

*# Efficient vector-based pattern storage with SQLite optimization*

**class** PatternLearningService:

**def** \_\_init\_\_(self):

self.vector\_store = LocalVectorStore() *# SQLite-based*

self.pattern\_extractor = PatternExtractor()

**async** **def** learn\_from\_project(self, project: Project):

*# Extract patterns in background*

patterns = **await** self.pattern\_extractor.extract(project)

*# Batch insert with SQLite optimization*

**async** **with** self.vector\_store.batch\_transaction():

**for** pattern **in** patterns:

**await** self.vector\_store.store\_pattern(pattern)

*# Update similarity indices for fast retrieval*

**await** self.vector\_store.rebuild\_indices()

**Scalability Architecture Principles:**

1. **Local-First Performance**: All core operations optimized for single-machine execution
2. **Intelligent Caching**: Multi-layer caching to minimize AI provider calls and database queries
3. **Asynchronous Processing**: Non-blocking operations for AI interactions and file generation
4. **Incremental Operations**: Only process changed components to minimize computational overhead
5. **Resource Optimization**: Memory-efficient algorithms suitable for individual developer workstations

This architecture ensures that as projects grow in complexity and the pattern library expands, the system maintains high performance while delivering on all defined KPIs for individual developer productivity.

Personal-Optimized System Architecture

┌─────────────────────────────────────────────────────────────────┐

│ Samriddh AI Personal Edition │

├─────────────────────────────────────────────────────────────────┤

│ │

│ ┌─────────────────┐ ┌─────────────────┐ │

│ │ VS Code │ │ Web Frontend │ │

│ │ Extension │◄──►│ (React/Next) │ │

│ │ (Bridge) │ │ Direct Access │ │

│ └─────────────────┘ └─────────────────┘ │

│ │ │ │

│ └───────────────────────┼────────────────────────────┤

│ │ │

│ ┌─────────────────────────────────┼────────────────────────────┐│

│ │ Core Application (FastAPI) ││

│ ├────────────────────────────────────────────────────────────┤│

│ │ ┌─────────────────┐ ┌─────────────────┐ ││

│ │ │ Project Planner │ │ Code Generator │ ││

│ │ │ Agent │ │ Agent │ ││

│ │ └─────────────────┘ └─────────────────┘ ││

│ │ ││

│ │ ┌─────────────────┐ ┌─────────────────┐ ││

│ │ │ Visual Review │ │ Quality │ ││

│ │ │ Agent │ │ Assurance Agent │ ││

│ │ └─────────────────┘ └─────────────────┘ ││

│ │ ││

│ │ ┌─────────────────┐ ┌─────────────────┐ ││

│ │ │ Debugging & │ │ Cross-Project │ ││

│ │ │ Validation │ │ Learning │ ││

│ │ │ Service │ │ Service │ ││

│ │ └─────────────────┘ └─────────────────┘ ││

│ │ ││

│ │ ┌─────────────────┐ ┌─────────────────┐ ││

│ │ │ SQLite + │ │ AI Provider │ ││

│ │ │ Alembic Manager │ │ Manager │ ││

│ │ └─────────────────┘ └─────────────────┘ ││

│ └────────────────────────────────────────────────────────────┤│

│ │

│ ┌──────────────────────────────────────────────────────────┐ │

│ │ Local Storage Layer │ │

│ ├──────────────────────────────────────────────────────────┤ │

│ │ ┌─────────────┐ ┌─────────────┐ ┌─────────────┐ │ │

│ │ │ SQLite │ │ Project │ │ Pattern │ │ │

│ │ │ Database │ │ Storage │ │ Library │ │ │

│ │ │ (Alembic) │ │ (Local) │ │ (Learning) │ │ │

│ │ └─────────────┘ └─────────────┘ └─────────────┘ │ │

│ │ │ │

│ │ ┌─────────────┐ ┌─────────────┐ │ │

│ │ │ GLM 4.5 │ │ Claude │ │ │

│ │ │ (Default │ │ Sonnet 4 │ │ │

│ │ │ Thinking) │ │ (Thinking) │ │ │

│ │ └─────────────┘ └─────────────┘ │ │

│ │ │ │

│ │ ┌─────────────┐ │ │

│ │ │ Playwright │ │ │

│ │ │ Testing │ │ │

│ │ │ (MCP) │ │ │

│ │ └─────────────┘ │ │

│ └──────────────────────────────────────────────────────────┘ │

└─────────────────────────────────────────────────────────────────┘

**Samriddh AI Personal Edition: Architecture Description**

**System Architecture Overview**

The Samriddh AI Personal Edition follows a three-layer architecture optimized for individual developers, emphasizing local-first operations, direct access, and enterprise-grade code generation capabilities.

**Architecture Layers**

**Layer 1: Client Interaction Layer**

**VS Code Extension (Bridge Component)**

* Lightweight plugin installed in Visual Studio Code
* Provides deep IDE integration with workspace access
* Enables file operations, directory navigation, and terminal control
* Serves as bridge between VS Code environment and core application

**Web Frontend Component**

* Responsive web application built with React and Next.js
* Implements direct access model without authentication requirements
* Provides primary user interface for AI interactions and project management
* Supports real-time communication with backend services

**Layer 2: Core Application Backend (FastAPI-based)**

**Project Planner Agent**

* Manages structured project planning workflows
* Orchestrates progression through defined phases: charter, architecture, foundation, module specification, scaffolding, and implementation
* Handles user review cycles and document approval processes

**Code Generator Agent**

* Generates production-ready application code
* Incorporates enterprise features: audit logging, error handling, data validation, performance optimization
* Ensures code quality standards and testing integration
* Supports multiple programming languages and frameworks

**Visual Review Agent**

* Integrates Playwright MCP for automated UI testing
* Performs visual regression testing and accessibility compliance checks
* Conducts cross-browser compatibility validation
* Generates improvement recommendations based on test results

**Quality Assurance Agent**

* Executes comprehensive test suites (unit, integration, end-to-end)
* Evaluates code quality metrics and performance benchmarks
* Monitors accessibility compliance (WCAG 2.1 standards)
* Provides quality scoring and improvement suggestions

**Debugging & Validation Service**

* Implements automated error detection and analysis
* Coordinates AI-powered error correction workflows
* Manages validate-document-correct feedback loops
* Provides intelligent debugging assistance with context awareness

**Cross-Project Learning Service**

* Maintains repository of successful architectural and coding patterns
* Enables knowledge transfer between projects
* Supports pattern recognition and recommendation systems
* Facilitates continuous improvement through machine learning

**SQLite + Alembic Manager**

* Manages local SQLite database operations
* Handles schema migrations using Alembic
* Ensures data integrity and performance optimization
* Provides backup and recovery capabilities

**AI Provider Manager**

* Manages communication with dual AI models:
  + GLM 4.5 (default provider with thinking mode enabled)
  + Claude Sonnet 4 (secondary provider with thinking mode enabled)
* Supports user-controlled provider selection and seamless switching
* Maintains conversation context across provider transitions
* Handles provider-specific optimizations and capabilities

**Layer 3: Local Storage Layer**

**SQLite Database**

* Primary persistent storage for all project data, metadata, and artifacts
* Stores user preferences, project configurations, and generated code
* Maintains conversation history and learning patterns

**Project Storage**

* Local file system storage for project files and documents
* Organizes project assets, templates, and generated artifacts
* Supports versioning and change tracking

**Pattern Library**

* Stores learned architectural and coding patterns
* Maintains success metrics and usage statistics for patterns
* Enables pattern matching and recommendation algorithms
* Supports cross-project knowledge sharing

**Additional Integration Components**

**Playwright Testing Runtime**

* Provides execution environment for automated testing workflows
* Supports visual regression testing and accessibility validation
* Enables cross-browser testing capabilities
* Integrates with quality assurance workflows

**Design Philosophy and Characteristics**

**Local-First Architecture**

* All core operations execute locally without external dependencies
* Minimizes latency and maximizes user control
* Ensures data privacy and security through local storage
* Supports offline development workflows

**Direct Access Model**

* Eliminates authentication barriers and user management complexity
* Provides immediate access to all features and capabilities
* Optimized for individual developer productivity
* Reduces setup friction and cognitive overhead

**Enterprise-Grade Output**

* Every generated application includes production-ready features
* Automatic incorporation of logging, validation, testing, and monitoring
* Adherence to enterprise coding standards and best practices
* Built-in scalability patterns and performance optimizations

**AI-First Development Workflow**

* Structured planning phases guided by AI assistance
* Intelligent code generation with quality validation
* Automated testing and improvement recommendations
* Continuous learning and pattern optimization

**Samriddh AI: Project Foundation & Core Contracts Document**

**1. Document Overview**

**Purpose:** This document establishes the foundational technical conventions for the entire Samriddh AI application. All subsequent module specifications and code generation must strictly adhere to the patterns and contracts defined herein. This ensures perfect consistency, maintainability, and developer efficiency from day one across all components of the AI-powered development companion.

**2. Foundational Type Definitions (types.ts)**

typescript

*/\*\**

*\* Core foundational types for Samriddh AI*

*\* Single source of truth for all shared data shapes*

*\*/*

*/\*\**

*\* UUID type for unique identifiers across the system*

*\*/*

**export** **type** UUID = string;

*/\*\**

*\* AI provider types supported by the system*

*\*/*

**export** **type** AIProviderType = 'GLM\_4\_5' | 'CLAUDE\_SONNET\_4';

*/\*\**

*\* Project planning phases following the structured methodology*

*\*/*

**export** **type** ProjectPhase = 'charter' | 'architecture' | 'foundation' | 'module\_spec' | 'scaffolding' | 'implementation';

*/\*\**

*\* Project status enumeration*

*\*/*

**export** **type** ProjectStatus = 'planning' | 'in\_progress' | 'completed' | 'archived';

*/\*\**

*\* Code generation status for tracking file generation progress*

*\*/*

**export** **type** GenerationStatus = 'pending' | 'generating' | 'completed' | 'failed';

*/\*\**

*\* Quality assurance test types*

*\*/*

**export** **type** TestType = 'unit' | 'integration' | 'e2e' | 'visual' | 'accessibility' | 'performance';

*/\*\**

*\* Core Project entity representing a software development project*

*\*/*

**export** **interface** Project {

*/\*\* Unique project identifier \*/*

id: UUID;

*/\*\* Project title \*/*

title: string;

*/\*\* Detailed project description \*/*

description: string;

*/\*\* Current project status \*/*

status: ProjectStatus;

*/\*\* Selected AI provider for this project \*/*

aiProvider: AIProviderType;

*/\*\* Project requirements and specifications \*/*

requirements?: ProjectRequirements;

*/\*\* Project creation timestamp \*/*

createdAt: string; *// ISO 8601 format*

*/\*\* Last update timestamp \*/*

updatedAt: string; *// ISO 8601 format*

}

*/\*\**

*\* Project requirements captured during planning phase*

*\*/*

**export** **interface** ProjectRequirements {

*/\*\* Core business requirements \*/*

businessLogic: string;

*/\*\* Technical constraints and preferences \*/*

technicalConstraints: string[];

*/\*\* Target user personas \*/*

targetUsers: string[];

*/\*\* Success criteria and KPIs \*/*

successCriteria: string[];

*/\*\* Explicit non-goals \*/*

nonGoals: string[];

}

*/\*\**

*\* Planning document for each phase of project development*

*\*/*

**export** **interface** ProjectDocument {

*/\*\* Unique document identifier \*/*

id: UUID;

*/\*\* Associated project ID \*/*

projectId: UUID;

*/\*\* Planning phase this document represents \*/*

phase: ProjectPhase;

*/\*\* Document content (markdown or structured data) \*/*

content: string;

*/\*\* User approval status \*/*

approved: boolean;

*/\*\* User feedback and modifications \*/*

feedback?: string;

*/\*\* Document creation timestamp \*/*

createdAt: string;

*/\*\* Last update timestamp \*/*

updatedAt: string;

}

*/\*\**

*\* Generated file entity tracking code generation*

*\*/*

**export** **interface** GeneratedFile {

*/\*\* Unique file identifier \*/*

id: UUID;

*/\*\* Associated project ID \*/*

projectId: UUID;

*/\*\* File path relative to project root \*/*

filePath: string;

*/\*\* File content \*/*

content: string;

*/\*\* Content hash for change detection \*/*

contentHash: string;

*/\*\* Generation status \*/*

status: GenerationStatus;

*/\*\* AI provider used for generation \*/*

aiProvider: AIProviderType;

*/\*\* Generation metadata \*/*

metadata: GenerationMetadata;

*/\*\* File creation timestamp \*/*

createdAt: string;

*/\*\* Last update timestamp \*/*

updatedAt: string;

}

*/\*\**

*\* Metadata for code generation tracking*

*\*/*

**export** **interface** GenerationMetadata {

*/\*\* Prompt used for generation \*/*

prompt: string;

*/\*\* Generation duration in milliseconds \*/*

duration: number;

*/\*\* Number of tokens used \*/*

tokensUsed?: number;

*/\*\* Generation quality score \*/*

qualityScore?: number;

}

*/\*\**

*\* AI conversation message for tracking interactions*

*\*/*

**export** **interface** AIMessage {

*/\*\* Unique message identifier \*/*

id: UUID;

*/\*\* Associated project ID \*/*

projectId: UUID;

*/\*\* Message role \*/*

role: 'user' | 'assistant' | 'system';

*/\*\* Message content \*/*

content: string;

*/\*\* AI provider used \*/*

aiProvider: AIProviderType;

*/\*\* Message timestamp \*/*

timestamp: string;

}

*/\*\**

*\* Cross-project learning pattern storage*

*\*/*

**export** **interface** LearnedPattern {

*/\*\* Unique pattern identifier \*/*

id: UUID;

*/\*\* Pattern name/title \*/*

name: string;

*/\*\* Pattern description \*/*

description: string;

*/\*\* Pattern category \*/*

category: string;

*/\*\* Code/configuration pattern \*/*

pattern: string;

*/\*\* Success rate across projects \*/*

successRate: number;

*/\*\* Usage count \*/*

usageCount: number;

*/\*\* Projects where this pattern was successful \*/*

successfulProjects: UUID[];

*/\*\* Pattern creation timestamp \*/*

createdAt: string;

*/\*\* Last update timestamp \*/*

updatedAt: string;

}

*/\*\**

*\* Quality assurance report for generated code*

*\*/*

**export** **interface** QualityReport {

*/\*\* Unique report identifier \*/*

id: UUID;

*/\*\* Associated project ID \*/*

projectId: UUID;

*/\*\* Test results by type \*/*

testResults: Record<TestType, TestResult>;

*/\*\* Overall quality score \*/*

overallScore: number;

*/\*\* Accessibility compliance level \*/*

accessibilityScore: number;

*/\*\* Performance metrics \*/*

performanceMetrics: PerformanceMetrics;

*/\*\* Report generation timestamp \*/*

generatedAt: string;

}

*/\*\**

*\* Individual test result*

*\*/*

**export** **interface** TestResult {

*/\*\* Test execution status \*/*

passed: boolean;

*/\*\* Test coverage percentage \*/*

coverage: number;

*/\*\* Number of tests run \*/*

testsRun: number;

*/\*\* Number of failures \*/*

failures: number;

*/\*\* Execution duration \*/*

duration: number;

*/\*\* Detailed results \*/*

details?: string;

}

*/\*\**

*\* Performance metrics from automated testing*

*\*/*

**export** **interface** PerformanceMetrics {

*/\*\* Load time in milliseconds \*/*

loadTime: number;

*/\*\* First contentful paint \*/*

firstContentfulPaint: number;

*/\*\* Largest contentful paint \*/*

largestContentfulPaint: number;

*/\*\* Cumulative layout shift \*/*

cumulativeLayoutShift: number;

*/\*\* Bundle size in bytes \*/*

bundleSize: number;

}

*/\*\**

*\* Generic API response wrapper for successful responses*

*\*/*

**export** **interface** ApiResponse<T> {

*/\*\* Success indicator \*/*

success: true;

*/\*\* Response data payload \*/*

data: T;

}

*/\*\**

*\* Standardized API error response*

*\*/*

**export** **interface** ApiErrorResponse {

*/\*\* Failure indicator \*/*

success: false;

*/\*\* Error details \*/*

error: {

*/\*\* Human-readable error message \*/*

message: string;

*/\*\* Optional error code for programmatic handling \*/*

code?: string;

};

}

*/\*\**

*\* Paginated response wrapper*

*\*/*

**export** **interface** PaginatedResponse<T> {

*/\*\* Array of items for current page \*/*

items: T[];

*/\*\* Total number of items across all pages \*/*

total: number;

*/\*\* Current page number (1-based) \*/*

page: number;

*/\*\* Number of items per page \*/*

pageSize: number;

*/\*\* Total number of pages \*/*

totalPages: number;

*/\*\* Whether there is a next page \*/*

hasNext: boolean;

*/\*\* Whether there is a previous page \*/*

hasPrev: boolean;

}

*/\*\**

*\* Validation error details*

*\*/*

**export** **interface** ValidationError {

*/\*\* Field name that failed validation \*/*

field: string;

*/\*\* Validation error message \*/*

message: string;

*/\*\* Invalid value that was provided \*/*

value?: any;

}

*/\*\**

*\* System configuration interface*

*\*/*

**export** **interface** AppConfig {

*/\*\* API base URL \*/*

apiBaseUrl: string;

*/\*\* Default AI provider \*/*

defaultAIProvider: AIProviderType;

*/\*\* Database connection string \*/*

databaseUrl: string;

*/\*\* Enable development mode features \*/*

developmentMode: boolean;

}

**3. Formal API Contract Scaffolding (OpenAPI 3.0 Specification)**

text

openapi: 3.0.0

info:

title: Samriddh AI API

description: >-

Complete API specification for Samriddh AI - Personal Enterprise-Grade Development Companion.

This API supports structured project planning, AI-powered code generation, quality assurance,

and cross-project learning capabilities.

version: 1.0.0

contact:

name: Samriddh AI Development

email: dev@samriddh.ai

servers:

- url: http://localhost:8000

description: Local development server

- url: https://api.samriddh.ai

description: Production server

components:

securitySchemes:

# Note: No authentication required per charter requirements

# Direct access architecture

directAccess:

type: apiKey

in: header

name: X-Direct-Access

description: Direct access key for local development

schemas:

UUID:

type: string

format: uuid

description: Universally unique identifier

AIProviderType:

type: string

enum:

- GLM\_4\_5

- CLAUDE\_SONNET\_4

description: Supported AI provider types

ProjectPhase:

type: string

enum:

- charter

- architecture

- foundation

- module\_spec

- scaffolding

- implementation

description: Project planning phases

ProjectStatus:

type: string

enum:

- planning

- in\_progress

- completed

- archived

description: Project status enumeration

Project:

type: object

required:

- id

- title

- description

- status

- aiProvider

- createdAt

- updatedAt

properties:

id:

$ref: '#/components/schemas/UUID'

title:

type: string

minLength: 1

maxLength: 200

description:

type: string

maxLength: 2000

status:

$ref: '#/components/schemas/ProjectStatus'

aiProvider:

$ref: '#/components/schemas/AIProviderType'

requirements:

$ref: '#/components/schemas/ProjectRequirements'

createdAt:

type: string

format: date-time

updatedAt:

type: string

format: date-time

ProjectRequirements:

type: object

properties:

businessLogic:

type: string

description: Core business requirements and logic

technicalConstraints:

type: array

items:

type: string

description: Technical constraints and preferences

targetUsers:

type: array

items:

type: string

description: Target user personas

successCriteria:

type: array

items:

type: string

description: Success criteria and KPIs

nonGoals:

type: array

items:

type: string

description: Explicit non-goals and out-of-scope items

ProjectDocument:

type: object

required:

- id

- projectId

- phase

- content

- approved

- createdAt

- updatedAt

properties:

id:

$ref: '#/components/schemas/UUID'

projectId:

$ref: '#/components/schemas/UUID'

phase:

$ref: '#/components/schemas/ProjectPhase'

content:

type: string

description: Document content in markdown or structured format

approved:

type: boolean

description: User approval status

feedback:

type: string

description: User feedback and modifications

createdAt:

type: string

format: date-time

updatedAt:

type: string

format: date-time

GeneratedFile:

type: object

required:

- id

- projectId

- filePath

- content

- contentHash

- status

- aiProvider

- createdAt

- updatedAt

properties:

id:

$ref: '#/components/schemas/UUID'

projectId:

$ref: '#/components/schemas/UUID'

filePath:

type: string

description: File path relative to project root

content:

type: string

description: Generated file content

contentHash:

type: string

description: SHA-256 hash of content for change detection

status:

type: string

enum: [pending, generating, completed, failed]

aiProvider:

$ref: '#/components/schemas/AIProviderType'

metadata:

$ref: '#/components/schemas/GenerationMetadata'

createdAt:

type: string

format: date-time

updatedAt:

type: string

format: date-time

GenerationMetadata:

type: object

properties:

prompt:

type: string

description: Prompt used for generation

duration:

type: integer

description: Generation duration in milliseconds

tokensUsed:

type: integer

description: Number of tokens consumed

qualityScore:

type: number

format: float

minimum: 0

maximum: 100

description: Generation quality score

AIMessage:

type: object

required:

- id

- projectId

- role

- content

- aiProvider

- timestamp

properties:

id:

$ref: '#/components/schemas/UUID'

projectId:

$ref: '#/components/schemas/UUID'

role:

type: string

enum: [user, assistant, system]

content:

type: string

aiProvider:

$ref: '#/components/schemas/AIProviderType'

timestamp:

type: string

format: date-time

LearnedPattern:

type: object

required:

- id

- name

- description

- category

- pattern

- successRate

- usageCount

- createdAt

- updatedAt

properties:

id:

$ref: '#/components/schemas/UUID'

name:

type: string

minLength: 1

maxLength: 100

description:

type: string

maxLength: 500

category:

type: string

maxLength: 50

pattern:

type: string

description: Code or configuration pattern

successRate:

type: number

format: float

minimum: 0

maximum: 1

usageCount:

type: integer

minimum: 0

successfulProjects:

type: array

items:

$ref: '#/components/schemas/UUID'

createdAt:

type: string

format: date-time

updatedAt:

type: string

format: date-time

QualityReport:

type: object

required:

- id

- projectId

- testResults

- overallScore

- accessibilityScore

- performanceMetrics

- generatedAt

properties:

id:

$ref: '#/components/schemas/UUID'

projectId:

$ref: '#/components/schemas/UUID'

testResults:

type: object

additionalProperties:

$ref: '#/components/schemas/TestResult'

overallScore:

type: number

format: float

minimum: 0

maximum: 100

accessibilityScore:

type: number

format: float

minimum: 0

maximum: 100

performanceMetrics:

$ref: '#/components/schemas/PerformanceMetrics'

generatedAt:

type: string

format: date-time

TestResult:

type: object

required:

- passed

- coverage

- testsRun

- failures

- duration

properties:

passed:

type: boolean

coverage:

type: number

format: float

minimum: 0

maximum: 100

testsRun:

type: integer

minimum: 0

failures:

type: integer

minimum: 0

duration:

type: integer

description: Duration in milliseconds

details:

type: string

description: Detailed test results

PerformanceMetrics:

type: object

required:

- loadTime

- firstContentfulPaint

- largestContentfulPaint

- cumulativeLayoutShift

- bundleSize

properties:

loadTime:

type: integer

description: Load time in milliseconds

firstContentfulPaint:

type: integer

description: First contentful paint in milliseconds

largestContentfulPaint:

type: integer

description: Largest contentful paint in milliseconds

cumulativeLayoutShift:

type: number

format: float

description: Cumulative layout shift score

bundleSize:

type: integer

description: Bundle size in bytes

ApiResponse:

type: object

required:

- success

- data

properties:

success:

type: boolean

enum: [true]

data:

type: object

description: Response payload

ApiErrorResponse:

type: object

required:

- success

- error

properties:

success:

type: boolean

enum: [false]

error:

type: object

required:

- message

properties:

message:

type: string

description: Human-readable error message

code:

type: string

description: Error code for programmatic handling

PaginatedResponse:

type: object

required:

- items

- total

- page

- pageSize

- totalPages

- hasNext

- hasPrev

properties:

items:

type: array

items:

type: object

total:

type: integer

minimum: 0

page:

type: integer

minimum: 1

pageSize:

type: integer

minimum: 1

maximum: 100

totalPages:

type: integer

minimum: 0

hasNext:

type: boolean

hasPrev:

type: boolean

tags:

- name: Projects

description: Project management and planning operations

- name: Planning

description: Structured project planning workflow

- name: AI

description: AI provider interactions and conversation management

- name: Code

description: Code generation and file management

- name: Quality

description: Quality assurance, testing, and validation

- name: Learning

description: Cross-project learning and pattern management

- name: Config

description: Configuration and system management

**4. Foundational Frontend State & Data-Fetching Strategy**

**Authentication & Session Management**

Since the application uses direct access without authentication, the global state focuses on:

* **AI Provider Selection**: Track user's preferred AI provider (GLM 4.5 or Claude Sonnet 4)
* **Project Context**: Maintain current project state and preferences
* **Configuration**: Store user preferences and system configuration

**React Context Structure:**

typescript

**interface** AppContextType {

*// AI Provider Management*

currentProvider: AIProviderType;

setProvider: (provider: AIProviderType) => **void**;

*// Project State*

currentProject: Project | **null**;

setCurrentProject: (project: Project | **null**) => **void**;

*// System Configuration*

config: AppConfig;

updateConfig: (updates: Partial<AppConfig>) => **void**;

}

**Server Data Caching**

**Mandate: React Query** for all server state management with the following configuration:

* **Stale Time**: 5 minutes for project data, 1 hour for learned patterns
* **Cache Time**: 30 minutes default
* **Retry Logic**: 3 attempts with exponential backoff

**Base API Client Wrapper**

**Pre-configured Axios instance** with the following features:

typescript

**import** axios **from** 'axios';

**import** { ApiResponse, ApiErrorResponse } **from** './types';

**const** apiClient = axios.create({

baseURL: process.env.NEXT\_PUBLIC\_API\_BASE\_URL || 'http://localhost:8000',

timeout: 30000,

headers: {

'Content-Type': 'application/json',

},

});

*// Request interceptor for direct access (no auth tokens needed)*

apiClient.interceptors.request.use((config) => {

config.headers['X-Direct-Access'] = 'local-development';

**return** config;

});

*// Response interceptor for consistent API response handling*

apiClient.interceptors.response.use(

(response) => {

*// Unwrap ApiResponse<T> wrapper*

**if** (response.data?.success === true) {

**return** response.data.data;

}

**return** response.data;

},

(error) => {

*// Transform error to ApiErrorResponse format*

**const** apiError: ApiErrorResponse = {

success: false,

error: {

message: error.response?.data?.error?.message || error.message || 'An unexpected error occurred',

code: error.response?.data?.error?.code || error.response?.status?.toString(),

},

};

**return** Promise.reject(apiError);

}

);

**export** **default** apiClient;

**5. Core Environment Variable Manifest (.env.template)**

| **Variable Name** | **Scope** | **Description & Example** |
| --- | --- | --- |
| PORT | Backend | The port the backend server will run on (e.g., 8000). |
| DATABASE\_URL | Backend | SQLite database connection string (e.g., sqlite:///./data/samriddh.db). |
| NODE\_ENV | Backend | Application environment (e.g., development, production). |
| LOG\_LEVEL | Backend | Logging level (e.g., info, debug, error). |
| NEXT\_PUBLIC\_API\_BASE\_URL | Frontend | The base URL for the backend API (e.g., http://localhost:8000). |
| GLM\_API\_KEY | Backend | API key for GLM 4.5 provider (e.g., glm-key-123456). |
| ANTHROPIC\_API\_KEY | Backend | API key for Claude Sonnet 4 provider (e.g., sk-ant-123456). |
| PLAYWRIGHT\_HEADLESS | Backend | Run Playwright tests in headless mode (e.g., true). |
| VECTOR\_STORE\_PATH | Backend | Path for vector embeddings storage (e.g., ./data/embeddings). |
| CACHE\_TTL | Backend | Default cache time-to-live in seconds (e.g., 3600). |

**6. Project-Wide Coding Conventions**

**Naming Convention**

* **API endpoints and JSON properties**: camelCase (e.g., projectId, createdAt)
* **Database tables and columns**: snake\_case (e.g., project\_documents, created\_at)
* **File and directory names**: kebab-case (e.g., project-planner.service.ts)
* **Component names**: PascalCase (e.g., ProjectPlannerComponent)

**Error Handling**

* **All API error responses** must strictly conform to the ApiErrorResponse interface
* **Database errors** must be caught and transformed to user-friendly messages
* **Validation errors** must include field-level details using ValidationError interface
* **Async operations** must include proper error boundaries and fallback mechanisms

**Asynchronous Code**

* **Exclusive use of async/await** for all asynchronous operations
* **Prohibition of .then() chains** in new code
* **Promise-based APIs** must be wrapped with async/await
* **Error handling** must use try/catch blocks, not Promise.catch()

**Code Quality Standards**

* **JSDoc comments** required for all public functions, interfaces, and classes
* **TypeScript strict mode** enabled for all modules
* **ESLint and Prettier** configuration enforced via pre-commit hooks
* **Test coverage** minimum 95% for critical paths, 80% overall
* **Performance budgets** enforced for bundle size and API response times

**Documentation Requirements**

* **API endpoints** documented with OpenAPI specifications
* **Database schemas** documented with inline comments
* **Business logic** explained with comprehensive JSDoc
* **Configuration options** documented with examples and default values

**This Project Foundation & Core Contracts Document serves as the definitive technical foundation for all Samriddh AI development. All subsequent modules, features, and code generation must adhere strictly to these established patterns and conventions.**

**Samriddh AI: Complete Module List & module Implementation order**

Based on the comprehensive documentation (Project Charter, Product Features & Capabilities, High-Level System Design, and Project Foundation & Core Contracts), here is the complete list of modules required for Samriddh AI implementation:

**Core Architecture Modules**

**1. Project Planning Module**

* **Responsibilities**: Manages structured Phase 0-3.1+ planning workflows, user review cycles, and document generation
* **Features**: Interactive requirements gathering, project charter generation, architectural blueprint creation, implementation planning
* **Priority**: P0 - Must-Have
* **Dependencies**: AI Provider Management, User Interface

**2. AI Provider Management Module**

* **Responsibilities**: Handles dual AI model orchestration (GLM 4.5 + Claude Sonnet 4), user-controlled provider selection, and context preservation
* **Features**: Provider switching, context preservation, performance tracking, intelligent routing suggestions
* **Priority**: P0 - Must-Have
* **Dependencies**: Configuration Management, Cache Management

**3. Code Generation Module**

* **Responsibilities**: Produces production-ready applications with enterprise features, manages file generation workflows
* **Features**: Complete application scaffolding, enterprise-grade code patterns, quality validation, file-by-file generation
* **Priority**: P0 - Must-Have
* **Dependencies**: AI Provider Management, Data Management, Quality Assurance

**4. Data Management Module**

* **Responsibilities**: Manages SQLite databases with data integrity enforcement and performance optimization
* **Features**: Database operations, connection pooling, query optimization, data validation
* **Priority**: P0 - Must-Have
* **Dependencies**: Database Migration, Configuration Management

**5. Database Migration Module**

* **Responsibilities**: Handles Alembic migrations, schema versioning, and data migration support
* **Features**: Automatic migration generation, rollback capabilities, dependency tracking, schema versioning
* **Priority**: P0 - Must-Have
* **Dependencies**: Data Management, Configuration Management

**Quality Assurance & Testing Modules**

**6. Quality Assurance Module**

* **Responsibilities**: Performs automated testing, accessibility compliance validation, and quality scoring
* **Features**: Test orchestration, quality metrics calculation, compliance checking, report generation
* **Priority**: P1 - Important
* **Dependencies**: Testing Framework, Visual Review, Performance Monitoring

**7. Testing Framework Module**

* **Responsibilities**: Provides comprehensive test suite generation and execution capabilities
* **Features**: Unit tests, integration tests, end-to-end tests, test data generation, coverage reporting
* **Priority**: P1 - Important
* **Dependencies**: Code Generation, Quality Assurance

**8. Visual Review Module**

* **Responsibilities**: Implements Playwright MCP integration for automated UI testing and visual validation
* **Features**: Screenshot comparison, visual regression testing, cross-browser validation, accessibility testing
* **Priority**: P1 - Important
* **Dependencies**: Testing Framework, Quality Assurance

**9. Debugging & Validation Module**

* **Responsibilities**: Provides intelligent error detection, AI-powered correction suggestions, and validation loops
* **Features**: Error analysis, automatic fix generation, validation workflows, quality gates
* **Priority**: P1 - Important
* **Dependencies**: AI Provider Management, Code Generation, Quality Assurance

**Learning & Intelligence Modules**

**10. Cross-Project Learning Module**

* **Responsibilities**: Implements pattern recognition, solution reuse, and adaptive learning from successful implementations
* **Features**: Pattern extraction, success rate tracking, knowledge transfer, recommendation engine
* **Priority**: P2 - Nice-to-Have
* **Dependencies**: Pattern Recognition, Knowledge Base Management, Vector Storage

**11. Pattern Recognition Module**

* **Responsibilities**: Identifies and stores successful architectural and coding patterns across projects
* **Features**: Pattern extraction algorithms, similarity matching, usage analytics, pattern categorization
* **Priority**: P2 - Nice-to-Have
* **Dependencies**: Vector Storage, Cross-Project Learning

**12. Knowledge Base Management Module**

* **Responsibilities**: Manages storage and retrieval of learned patterns, best practices, and project insights
* **Features**: Knowledge indexing, semantic search, pattern libraries, knowledge graph management
* **Priority**: P2 - Nice-to-Have
* **Dependencies**: Vector Storage, Pattern Recognition

**13. Vector Storage Module**

* **Responsibilities**: Provides vector-based storage for semantic search and pattern matching
* **Features**: Embedding generation, similarity search, vector indexing, performance optimization
* **Priority**: P2 - Nice-to-Have
* **Dependencies**: Data Management, Performance Optimization

**User Interface & Integration Modules**

**14. User Interface Module**

* **Responsibilities**: Provides direct-access web interface for project management and AI interaction
* **Features**: React-based UI, real-time updates, responsive design, project dashboards
* **Priority**: P0 - Must-Have
* **Dependencies**: Frontend State Management, Real-time Communication, API Gateway

**15. Frontend State Management Module**

* **Responsibilities**: Manages React application state, caching, and data synchronization
* **Features**: React Query integration, global state management, cache invalidation, offline support
* **Priority**: P0 - Must-Have
* **Dependencies**: API Gateway, Configuration Management

**16. VS Code Bridge Module**

* **Responsibilities**: Enables seamless IDE integration, workspace monitoring, and file system access
* **Features**: VS Code extension, file operations, terminal integration, workspace analysis
* **Priority**: P1 - Important
* **Dependencies**: File System Management, Real-time Communication

**17. Real-time Communication Module**

* **Responsibilities**: Handles WebSocket connections for live updates and AI interactions
* **Features**: WebSocket management, real-time messaging, connection pooling, reconnection logic
* **Priority**: P1 - Important
* **Dependencies**: API Gateway, User Interface

**Infrastructure & Support Modules**

**18. API Gateway Module**

* **Responsibilities**: Provides RESTful API endpoints and request routing for all system operations
* **Features**: Request routing, validation, error handling, API documentation, rate limiting
* **Priority**: P0 - Must-Have
* **Dependencies**: Configuration Management, Error Reporting

**19. Configuration Management Module**

* **Responsibilities**: Manages environment variables, system configuration, and user preferences
* **Features**: Environment configuration, validation, hot reloading, secure storage
* **Priority**: P0 - Must-Have
* **Dependencies**: None (foundational)

**20. Logging & Monitoring Module**

* **Responsibilities**: Provides comprehensive logging, performance monitoring, and system observability
* **Features**: Structured logging, performance metrics, error tracking, health monitoring
* **Priority**: P1 - Important
* **Dependencies**: Configuration Management, Performance Optimization

**21. Error Reporting Module**

* **Responsibilities**: Handles error capture, reporting, and user notification for system failures
* **Features**: Error aggregation, user-friendly messaging, diagnostic information, error analytics
* **Priority**: P1 - Important
* **Dependencies**: Logging & Monitoring, User Interface

**22. Cache Management Module**

* **Responsibilities**: Implements multi-layer caching for performance optimization and cost reduction
* **Features**: Request caching, response caching, cache invalidation, performance analytics
* **Priority**: P1 - Important
* **Dependencies**: Configuration Management, Performance Optimization

**23. Performance Optimization Module**

* **Responsibilities**: Monitors and optimizes system performance, resource usage, and response times
* **Features**: Performance profiling, optimization suggestions, resource monitoring, benchmarking
* **Priority**: P1 - Important
* **Dependencies**: Logging & Monitoring, Cache Management

**File System & Storage Modules**

**24. File System Management Module**

* **Responsibilities**: Handles local file operations, project storage, and workspace management
* **Features**: File CRUD operations, directory management, file watching, project organization
* **Priority**: P0 - Must-Have
* **Dependencies**: Configuration Management, Error Reporting

**25. Asset Management Module**

* **Responsibilities**: Manages generated files, project assets, and resource organization
* **Features**: Asset organization, version tracking, content hashing, asset optimization
* **Priority**: P1 - Important
* **Dependencies**: File System Management, Code Generation

**26. Backup & Recovery Module**

* **Responsibilities**: Provides automated backup and recovery capabilities for project data
* **Features**: Automated backups, recovery procedures, data integrity validation, export/import
* **Priority**: P1 - Important
* **Dependencies**: File System Management, Data Management

**Utility & Enhancement Modules**

**27. Environment Management Module**

* **Responsibilities**: Manages development environments, dependencies, and runtime configuration
* **Features**: Environment setup, dependency management, runtime configuration, environment switching
* **Priority**: P1 - Important
* **Dependencies**: Configuration Management, File System Management

**28. API Documentation Module**

* **Responsibilities**: Generates and maintains comprehensive API documentation
* **Features**: OpenAPI documentation, interactive API explorer, documentation versioning, examples
* **Priority**: P1 - Important
* **Dependencies**: API Gateway, Configuration Management

**29. CLI & Automation Module**

* **Responsibilities**: Provides command-line interface and automation capabilities
* **Features**: CLI commands, script execution, automation workflows, batch operations
* **Priority**: P2 - Nice-to-Have
* **Dependencies**: File System Management, Code Generation

**30. Terminal Integration Module**

* **Responsibilities**: Manages terminal operations, command execution, and output parsing
* **Features**: Command execution, output capture, terminal emulation, process management
* **Priority**: P1 - Important
* **Dependencies**: VS Code Bridge, Error Reporting

**Module Implementation Priority**

**Phase 1 (Weeks 1-8): Core Foundation**

1. Configuration Management Module
2. File System Management Module
3. Data Management Module
4. Database Migration Module
5. API Gateway Module
6. AI Provider Management Module
7. User Interface Module
8. Frontend State Management Module

**Phase 2 (Weeks 9-12): Core Functionality**

1. Project Planning Module
2. Code Generation Module
3. Quality Assurance Module
4. Testing Framework Module
5. Real-time Communication Module

**Phase 3 (Weeks 13-16): Enhanced Features**

1. Visual Review Module
2. Debugging & Validation Module
3. VS Code Bridge Module
4. Terminal Integration Module
5. Logging & Monitoring Module
6. Error Reporting Module

**Phase 4 (Weeks 17-20): Advanced Capabilities**

1. Cache Management Module
2. Performance Optimization Module
3. Asset Management Module
4. Backup & Recovery Module
5. Cross-Project Learning Module
6. Pattern Recognition Module

**Future Enhancements**

1. Knowledge Base Management Module
2. Vector Storage Module
3. Environment Management Module
4. API Documentation Module
5. CLI & Automation Module

**Module Dependencies Overview**

**No Dependencies (Foundational):**

* Configuration Management Module

**Minimal Dependencies:**

* File System Management Module
* Data Management Module
* API Gateway Module

**Core Dependencies:**

* All other modules depend on foundational and minimal dependency modules

**Advanced Dependencies:**

* Learning modules depend on core functionality being established
* Optimization modules depend on basic system operation

This modular architecture ensures clean separation of concerns, maintainable code organization, and systematic implementation following the established development phases.