SPECIFICATION PRODUIT : Auto-Claude

Requirements Fonctionnels & Non-Fonctionnels

Version: 2.7.1
Date: 2 janvier 2026
Produit: Auto-Claude - Autonomous Multi-Session AI Coding Agent
Fondé sur: https://github.com/AndyMik90/Auto-Claude
Public cible: Développeurs, équipes d'engineering, entreprises tech

---

TABLE DES MATIÈRES

VISION PRODUIT

Énoncé de Vision

Auto-Claude est un agent IA autonome multi-session pour la programmation et l'ingénierie logicielle. Il automatise les tâches de coding répétitives, maintient le contexte entre sessions, exécute du code réel, et s'intègre à l'écosystème de développement (Git, GitHub, npm, package managers).

Slogan: *"Your autonomous coding companion that thinks, executes, and learns."*

Principes Fondamentaux

- ✅ **Autonome**: Exécute des tâches sans intervention humaine continue
- ✅ **Contextuel**: Maintient et réutilise le contexte entre sessions
- ✅ **Réaliste**: Exécute du code vrai (sandboxé), pas juste de la génération
- ✅ **Itératif**: Peut débugger, refactorer, et améliorer son propre travail
- ✅ **Transparent**: Logs détaillés, explique ses décisions

Différenciation vs Alternatives

Aspect	Auto-Claude	GitHub Copilot	Claude Web	ChatGPT
Multi-session context	✓ Full	✓ Limited	✗ No	✗ No
Code execution	✓ Sandboxed	✗ No	✗ No	✗ No
Git integration	✓ Full	✓ Limited	✗ No	✗ No
Autonomous execution	✓ Yes	✗ No	✗ No	✗ No
Persistence	✓ Full	✗ No	✗ No	✗ No
Customizable	✓ Yes	✗ Limited	✗ No	✗ No

REQUIREMENTS FONCTIONNELS

RF.1 Agent Core & Reasoning

RF.1.1 Task Execution

- [ ] **Agent doit exécuter des tâches de code de bout en bout**
  - Description: Accepte une description texte de tâche, décompose en sous-tâches, génère et exécute du code
  - Exemple: "Refactor this function to use async/await" → décompose → génère → exécute → valide
  - Entrée: Texte de tâche, contexte (fichiers, repo, historique)
  - Sortie: Code généré, résultats d'exécution, rapports d'erreur
  - Acceptation: Tâche complète sans erreurs non-recouvrables

RF.1.2 Multi-Step Reasoning

- [ ] **Agent doit décomposer tâches complexes en étapes séquentielles**
  - Entrée: Une tâche complexe ("Build a REST API with authentication")
  - Logique:
    1. Analyser la tâche
    2. Identifier les étapes requises
    3. Estimer la complexité
    4. Exécuter séquentiellement
    5. Valider chaque étape
  - Sortie: Liste des étapes exécutées, trace d'exécution
  - Acceptation: Trace complète avec timing par étape

RF.1.3 Error Handling & Recovery

- [ ] **Agent doit gérer les erreurs et tenter de se récupérer**
  - Types d'erreurs gérées:
    - Compilation errors (SyntaxError, TypeError)
    - Runtime errors (undefined reference, null pointer)
    - File not found errors
    - Permission denied errors
    - Timeout errors
  - Stratégies de récupération:
    - Analyser l'erreur
    - Proposer un fix
    - Réexécuter
    - Si récupération impossible: signaler à l'utilisateur
  - Max retry attempts: 3 par étape
  - Acceptation: Erreur documentée avec potential fixes

RF.1.4 Context Maintenance

- [ ] **Agent doit maintenir et réutiliser le contexte entre appels**
  - Contexte inclut:
    - Fichiers du projet
    - Historique des tâches exécutées
    - Variables d'état (env vars, config)
    - Résultats précédents
    - Décisions prises
  - Persistance: SQLite database
  - Récupération: Chargement au démarrage d'une session
  - Acceptation: Session résumée correctement, pas de perte de contexte

RF.1.5 Learning & Adaptation

- [ ] **Agent doit apprendre des exécutions précédentes**
  - Méchanisme: Stocker les patterns réussis, erreurs évitées
  - Exemple: Si "add error handling" a échoué x2, proposer une approche différente
  - Persistance: Fichier de patterns appris
  - Acceptation: Agent utilise des patterns appris dans nouvelles tâches

---

RF.2 Code Generation & Execution

RF.2.1 Code Generation avec Claude API

- [ ] **Agent doit générer du code via Claude API**
  - Modèle: `claude-3-5-sonnet` ou supérieur
  - Paramètres:
    - Temperature: 0.5-0.7 (balance créativité/déterminisme)
    - Max tokens: Adaptatif selon la tâche
    - System prompt: Instruire Claude sur l'objectif
  - Support langages: Python, JavaScript/TypeScript, Go, Rust, Bash, SQL
  - Acceptation: Code généré respecte la tâche, pas de placeholder code

RF.2.2 Code Sandbox & Execution

- [ ] **Agent doit exécuter du code dans un sandbox sécurisé**
  - Sandbox: Node.js isolated context ou Docker container (optionnel)
  - Langages supportés:
    - Python 3.9+: Via `exec()` ou subprocess
    - JavaScript/TypeScript: Node.js runtime
    - Bash: Shell commands (whitelist)
  - Restrictions:
    - No file system write outside project dir
    - No network access (sauf API whitelist)
    - Max execution time: 30 secondes par command
    - Memory limit: 256MB par process
  - Output capture: stdout, stderr, exit code
  - Acceptation: Code exécuté sans accès système dangéreux

RF.2.3 Testing & Validation

- [ ] **Agent doit tester le code généré**
  - Types de tests:
    - Unit tests (si test framework existe)
    - Integration tests (avec API mock si nécessaire)
    - Manual validation via script
  - Exigence: Tous les tests doivent passer avant acceptation
  - Rapport: Nombre de tests, pass/fail, coverage
  - Acceptation: Test report généré, >80% tests passing

RF.2.4 Code Quality Analysis

- [ ] **Agent doit analyser la qualité du code généré**
  - Outils utilisés:
    - Linting (ESLint, Pylint)
    - Type checking (TypeScript, mypy)
    - Code style (Prettier, Black)
  - Métriques rapportées:
    - Nombre de warnings/errors
    - Complexity score
    - Code duplication
  - Seuils acceptables:
    - 0 errors (mandatory)
    - <5 warnings (warn si >5)
    - Complexity < 10 (warn si >10)
  - Acceptation: Code passe quality gates

RF.2.5 Git Integration & Commits

- [ ] **Agent doit intégrer avec Git pour versioning**
  - Opérations Git:
    - Clone repository
    - Create branches
    - Stage & commit changes
    - Push to remote
    - Create pull requests (via GitHub API)

    - Comportement:
        - Chaque tâche = branche séparée
        - Commit message auto-généré avec contexte
        - Peut créer PR avec description
    - Acceptation: Commits présents dans Git history, PRs créés sur GitHub

---

RF.3 Session Management

RF.3.1 Session Persistence

- [ ] **Agent doit créer et maintenir des sessions persistantes**
    - Session inclut:
        - ID unique (UUID)
        - Timestamp création/modification
        - Liste des tâches exécutées
        - Historique complet (entrée/sortie)
        - Métadonnées (workspace, config)
    - Storage: SQLite database (sessions.db)
    - Récupération: Charger session précédente par ID
    - Acceptation: Session sauvegardée, récupérable

RF.3.2 Multi-Session Concurrency

- [ ] **Agent doit supporter plusieurs sessions simultanées**
    - Isolation: Chaque session a sa propre working directory
    - Locking: Éviter les race conditions sur fichiers partagés
    - Max sessions: Configurable (default: 10)
    - Acceptation: Sessions exécutées en parallèle sans corruption

RF.3.3 Session History & Replay

- [ ] **Agent doit permettre le replay de sessions**
    - Fonctionnalité: Re-exécuter une session complète
    - Historique: Accessible via CLI ou API
    - Format: JSON export de tous les événements
    - Acceptation: Replay produit résultats identiques

RF.3.4 Session Summary & Export

- [ ] **Agent doit générer des résumés de session**
    - Contenu:
        - Tâches exécutées (count, success rate)
        - Code généré (LOC, fichiers modifiés)
        - Erreurs rencontrées & fixes
        - Time spent (per task, total)
        - Files modified
    - Formats: JSON, Markdown, PDF (optionnel)
    - Acceptation: Résumé généré, exportable

RF.4 CLI & User Interface

RF.4.1 Command-Line Interface

- [ ] **Agent doit exposer une CLI complète**
  - Framework: Electron (desktop app) + CLI backend
  - Commandes principales:
    auto-claude init [--name] # Initialiser workspace
    auto-claude task new "<desc>" # Créer tâche
    auto-claude task run <id> # Exécuter tâche
    auto-claude task list # Lister tâches
    auto-claude task cancel <id> # Annuler tâche
    auto-claude session list # Lister sessions
    auto-claude session show <id> # Détails session
    auto-claude session export <id> # Exporter session
    auto-claude status # Status système
    auto-claude config set <key> <v> # Configuration
    auto-claude logs [--tail] [--grep] # Viewer logs
  - Acceptation: Toutes les commandes fonctionnelles

RF.4.2 Desktop Application (Electron)

- [ ] **Agent doit avoir une interface graphique desktop**
  - Platform: Windows, macOS, Linux
  - Technologie: Electron + Vue.js/React
  - Sections UI:
    - **Workspace**: Vue des fichiers du projet
    - **Tasks**: Panel pour créer/exécuter tâches
    - **Execution**: Live view de l'exécution avec logs streaming
    - **Results**: Affiche les résultats (code, tests, etc.)
    - **History**: Historique des sessions avec search
    - **Settings**: Configuration API key, modèle, etc.
  - Acceptation: UI responsive, pas de lag <500ms

RF.4.3 Real-time Logging & Monitoring

- [ ] **Agent doit afficher les logs en temps réel**
  - Features:
    - Live streaming de stdout/stderr
    - Colorized output (errors en rouge, success en vert)
    - Searchable logs
    - Export logs to file
    - Tail mode (dernières N lignes)
  - Performance: Max 100ms latency pour log display
  - Acceptation: Logs affichés en <100ms d'exécution

RF.4.4 Interactive REPL Mode

- [ ] **Agent doit supporter un mode REPL interactif**
  - Commandes interactives:
    - Execute Python/JS snippets
    - Inspect variables
    - Modify context

        - Ask Agent questions
    - Contexte: Maintenu entre commandes
    - Session: Sauvegardé pour histoire
    - Acceptation: REPL responsive, contexte maintained

RF.5 Integrations

RF.5.1 GitHub Integration

- [ ] **Agent doit intégrer avec GitHub**
    - Capacités:
        - Authenticate via PAT (Personal Access Token)
        - Clone repositories
        - Create branches & commits
        - Push to remote
        - Create pull requests avec descriptions
        - Comment on PRs
        - Read issues & PRs
    - Authentication: PAT dans env var GITHUB_TOKEN
    - Rate limits: Respecter les limites GitHub API
    - Acceptation: PR créées, commits pushés

RF.5.2 npm/Package Manager Integration

- [ ] **Agent doit gérer les dépendances**
    - Capacités:
        - `npm install`, `npm update`
        - `npm run` pour scripts
        - Lecture `package.json` & `package-lock.json`
        - Support `yarn`, `pnpm` (optionnel)
    - Utilisation: Auto-install si dépendances manquantes
    - Acceptation: Dépendances installées, lock files updated

RF.5.3 MCP (Model Context Protocol) Support

- [ ] **Agent doit supporter les servers MCP**
    - Fonctionnalité: Intégrer MCP servers pour étendre capacités
    - Exemple: MCP server pour GitHub, database, APIs
    - Configuration: Via config file
    - Acceptation: MCP tools disponibles dans contexte

RF.5.4 API Integrations

- [ ] **Agent doit supporter les appels API**
    - Capacités:
        - HTTP requests (GET, POST, PUT, DELETE)
        - Authentification (Bearer token, API key)
        - JSON/XML parsing
        - Error handling pour API failures
    - Whitelist: Configurable URLs/domaines
    - Rate limiting: Respecter rate limits API
    - Acceptation: API calls exécutés, résultats reçus

RF.6 Configuration & Customization

RF.6.1 Configuration Management

- [ ] **Agent doit permettre la configuration**
  - Configuration file: ~/.auto-claude/config.json
  - Paramètres:
    - apiKey: Claude API key
    - model: Modèle Claude (default: claude-3-5-sonnet)
    - temperature: 0.0-1.0
    - maxTokens: Max tokens par appel
    - workspace: Répertoire default workspace
    - gitHub.token: GitHub PAT
    - sandbox.timeout: Max execution time
    - sandbox.memoryLimit: Max memory
  - Sources: File + env vars + CLI flags (priority order)
  - Acceptation: Configuration appliquée correctement

RF.6.2 Environment Variables

- [ ] **Agent doit supporter les env variables**
  - Variables requises:
    - CLAUDE_API_KEY: API key pour Claude
    - GITHUB_TOKEN: GitHub PAT (optionnel)
    - AUTO_CLAUDE_WORKSPACE: Default workspace path
  - Variables optionnelles:
    - CLAUDE_MODEL: Modèle à utiliser
    - LOG_LEVEL: debug, info, warn, error
    - SANDBOX_TIMEOUT: Timeout en secondes
  - Acceptation: Env vars lues et appliquées

RF.6.3 Custom Prompts & Instructions

- [ ] **Agent doit permettre des prompts customisés**
  - Fichiers:
    - system.prompt: System prompt personnalisé
    - task-template.md: Template pour nouvelles tâches
  - Usage: Intégré dans appels Claude
  - Acceptation: Custom prompts utilisés dans générations

RF.6.4 Plugin System

- [ ] **Agent doit supporter des plugins (optionnel, future)**
  - Architecture: Plugin loader
  - Interface: Hooks et API standardisée
  - Exemple: Plugin pour intégration Slack, monitoring, etc.
  - Acceptation: Plugins chargés, hooks exécutés

RF.7 Monitoring & Analytics

RF.7.1 Metrics Collection

- [ ] **Agent doit collecter des métriques d'usage**
  - Métriques:
    - Tasks executed (count, success rate)
    - Code generated (LOC, files)
    - Errors (count, types)
    - Execution time (per task, average)
    - API calls (count, cost estimation)
  - Storage: SQLite, JSON logs
  - Retention: 1 année (configurable)
  - Acceptation: Métriques sauvegardées

RF.7.2 Performance Monitoring

- [ ] **Agent doit monitorer les performances**
  - Métriques:
    - API latency (Claude calls)
    - Code execution time
    - Memory usage
    - Disk usage
  - Alertes: Si anomalies détectées
  - Dashboards: Via CLI ou UI
  - Acceptation: Performance data collectée et affichée

RF.7.3 Error Tracking

- [ ] **Agent doit tracker les erreurs**
  - Données:
    - Erreur message & type
    - Stack trace
    - Contexte (task, session, file)
    - Tentative de fix
    - Résultat (succès ou pas)
  - Storage: Error database
  - Rapports: Exportable
  - Acceptation: Erreurs tracées, reportables

RF.8 Security & Privacy

RF.8.1 API Key Management

- [ ] **Agent doit gérer les API keys de façon sécurisée**
  - Stockage: Jamais en clair dans logs
  - Masquage: Afficher juste les derniers 4 caractères
  - Rotation: Support pour changer la key
  - Acceptation: Keys protégées, jamais exposées

RF.8.2 Sandbox Isolation

- [ ] **Agent doit isoler l'exécution de code**
  - Isolation: Sandbox process séparé
  - Restrictions:
    - No file system write outside project
    - No network access (sauf whitelist)
    - No access aux env vars sensibles
  - Timeout: 30 secondes max
  - Acceptation: Code exécuté de façon isolée

RF.8.3 Input Validation & Sanitization

- [ ] **Agent doit valider toutes les entrées**
  - Validation:
    - CLI arguments
    - Configuration values
    - API responses
    - User input
  - Sanitization: Échapper special chars pour sécurité
  - Acceptation: Injections impossibles

RF.8.4 Audit Logging

- [ ] **Agent doit logger toutes les actions importantes**
  - Events loggés:
    - Task creation & execution
    - Code generation & execution
    - Git operations
    - Configuration changes
    - API calls
  - Format: Avec timestamp, actor, action, result
  - Acceptance: Audit logs complets et consultables

REQUIREMENTS NON-FONCTIONNELS

RNF.1 Performance

RNF.1.1 API Response Time

- [ ] Claude API calls: < 5 secondes (p95)
- [ ] Token counting: < 1 seconde
- [ ] Code execution: < 30 secondes timeout
- [ ] CLI commands: < 2 secondes (non-blocking)
- Metric: Mesurer via timing logs, dashboard

RNF.1.2 UI Responsiveness

- [ ] Desktop app UI: < 500ms latency pour user input
- [ ] Log display: < 100ms pour nouveau log
- [ ] Task list refresh: < 1 seconde
- Metric: Profiling via dev tools

RNF.1.3 Memory Usage

- [ ] Idle memory: < 100MB
- [ ] Running task: < 500MB
- [ ] With 10 sessions: < 1GB
- Metric: Monitorer via process monitor

RNF.1.4 Startup Time

- [ ] CLI: < 500ms
- [ ] Desktop app: < 3 secondes
- Metric: Mesurer au démarrage

RNF.2 Reliability & Availability

RNF.2.1 Uptime & MTBF

- [ ] Target uptime: 99.5% (commercial grade)
- [ ] Mean Time Between Failures: > 30 jours
- [ ] Mean Time To Recovery: < 5 minutes
- Metric: Track via health checks

RNF.2.2 Data Persistence

- [ ] Aucune perte de session même en crash
- [ ] Database backup automatique daily
- [ ] Recovery point objective (RPO): < 1 heure
- Metric: Tester via failure scenarios

RNF.2.3 Error Rate

- [ ] Unhandled errors: < 0.1%
- [ ] Failed tasks due to system: < 5%
- [ ] API errors (rate limit, auth): < 2%
- Metric: Monitor error logs

RNF.2.4 Graceful Degradation

- [ ] Si API Claude down: Queue les tâches, retry later
- [ ] Si GitHub down: Utiliser local git, sync later
- [ ] Si sandbox down: Mode simulation
- Metric: Test scenarios de failure

RNF.3 Scalability

RNF.3.1 Concurrent Sessions

- [ ] Support minimum: 10 sessions simultanées
- [ ] Support optimal: 50+ sessions (avec config)
- [ ] Pas de degradation de performance
- Metric: Load testing

RNF.3.2 Data Growth

- [ ] Database peut supporter 10,000+ sessions
- [ ] Logs jusqu'à 100GB sans issue
- [ ] Query time restent < 1s même avec gros data
- Metric: Benchmark database

RNF.3.3 Code Execution

- [ ] Support projects > 100K LOC
- [ ] Temps d'analyse restent < 10 secondes
- [ ] Memory usage scalable
- Metric: Test avec gros projects

RNF.4 Maintainability

RNF.4.1 Code Quality

- [ ] Type safety: 100% (TypeScript strict mode)
- [ ] Test coverage: > 80%
- [ ] Documentation: Comprehensive JSDoc
- [ ] Code style: Automated linting
- Metric: Coverage reports, linter output

RNF.4.2 Modularity

- [ ] Clear separation of concerns
- [ ] Interfaces bien définies
- [ ] Dependencies minimal
- [ ] Easy to extend
- Metric: Cyclomatic complexity < 10

RNF.4.3 Logging & Debugging

- [ ] Debug mode avec verbose logging
- [ ] Structured logs (JSON format)
- [ ] Source maps pour stack traces
- [ ] Ability to replay execution
- Metric: Logging completeness

RNF.5 Security

RNF.5.1 Authentication & Authorization

- [ ] Claude API: Require valid API key
- [ ] GitHub: OAuth2 ou PAT validation
- [ ] No hardcoded credentials
- [ ] Secure storage of secrets
- Metric: Security audit

RNF.5.2 Data Protection

- [ ] Encryption for sensitive data at rest (optionnel)
- [ ] No sensitive data in logs
- [ ] Secure deletion of temporary files
- [ ] GDPR compliance (if applicable)
- Metric: Data protection audit

RNF.5.3 Vulnerability Management

- [ ] Dependency scanning (Snyk, npm audit)
- [ ] Regular security updates
- [ ] CVE tracking
- [ ] Incident response plan
- Metric: Regular scans

RNF.6 Usability

RNF.6.1 User Onboarding

- [ ] Setup wizard for initial configuration
- [ ] Help text for all commands
- [ ] Example tasks & templates
- [ ] Documentation > 80% coverage
- Metric: User feedback

RNF.6.2 Error Messages

- [ ] Clear & actionable error messages
- [ ] Suggestions for resolution
- [ ] Links to documentation
- [ ] Avoid technical jargon
- Metric: UX testing

RNF.6.3 Accessibility

- [ ] CLI: Full keyboard navigation
- [ ] Desktop app: WCAG 2.1 AA compliance
- [ ] High contrast mode support
- [ ] Screen reader compatible
- Metric: Accessibility audit

RNF.7 Compatibility

RNF.7.1 Platform Support

- [ ] Windows 10+
- [ ] macOS 11+
- [ ] Linux (Ubuntu 20.04+, etc.)
- [ ] Node.js 18+
- Metric: Test on all platforms

RNF.7.2 Language Support

- [ ] Python 3.9+
- [ ] JavaScript/TypeScript (Node 18+)
- [ ] Go 1.18+
- [ ] Rust 1.60+
- [ ] Bash (POSIX)
- Metric: Test code generation per language

RNF.7.3 Dependency Compatibility

- [ ] Support latest LTS versions
- [ ] Backward compatibility (2 major versions)
- [ ] Minimal transitive dependencies
- [ ] Regular dependency updates
- Metric: Dependency audit

RNF.8 Deployment & Operations

RNF.8.1 Installation

- [ ] Single command installation (npm, brew, etc.)
- [ ] Automatic environment setup
- [ ] Cross-platform binary support
- [ ] Self-update capability
- Metric: Installation success rate

RNF.8.2 Configuration

- [ ] Zero-config default (sensible defaults)
- [ ] Easy to customize
- [ ] Config validation on startup
- [ ] Clear error messages if invalid
- Metric: Config error rate

RNF.8.3 Monitoring & Observability

- [ ] Health check endpoint
- [ ] Metrics export (Prometheus format optional)
- [ ] Structured logging
- [ ] Performance profiling
- Metric: Monitoring completeness

RNF.8.4 Updates & Patches

- [ ] Automatic update checks
- [ ] Non-breaking updates installable instantly
- [ ] Breaking changes with migration guide
- [ ] Rollback capability
- Metric: Update deployment success rate

CONTRAINTES & LIMITATIONS

Contraintes Techniques

Contrainte	Valeur	Justification
API Rate Limit	1000 req/day (free tier)	Claude API free tier limit
Max Code Execution	30 secondes	Security & resource limit
Context Window	200K tokens (Claude 3.5)	API limit
Sandbox Memory	256MB	Prevent DoS
File Size Limit	100MB	Practical limit for processing
Session Retention	1 year	Storage management
Max Concurrent Tasks	10 (configurable)	Resource limit

Limitations Fonctionnelles

1. **Pas de GUI builder** - Code generation juste, pas de visual UI builder
2. **Pas de mobile app** - Desktop/CLI seulement
3. **Pas de real-time collab** - Single-user focus
4. **Pas de GPU access** - CPU only
5. **Pas de database direct** - SQLite seulement (no cloud DB)
6. **Pas de deployment auto** - User gère deployment
7. ***Pas de monitoring** - User setup propres dashboards

Limitations de Performance

- Tâches complexes (>1000 LOC generation) = 10-30s
- Large context (>100K tokens) = latency augmentée
- 50+ sessions = dégradation performance
- Gros projects (>500K LOC) = analyse lente

CAS D'USAGE

UC.1 Code Generation & Refactoring

Acteur: Développeur
Prérequis: Workspace configuré, API key valide
Flux principal:

1. Développeur: "Refactor this function to use async/await"
2. Agent: Analyse le code
3. Agent: Génère nouvelle version
4. Agent: Teste avec tests existants
5. Agent: Commit et push pour review
6. Résultat: PR créée sur GitHub

Acceptation: PR fonctionnelle, tests passent

UC.2 Bug Fixing

Acteur: Développeur
Prérequis: Test failing, repo configuré
Flux principal:

1. Dev: "Fix this test failure"
2. Agent: Lire test & erreur
3. Agent: Analyser code
4. Agent: Générer fix
5. Agent: Exécuter test
6. Agent: Itérer si test encore failing
7. Résultat: Test passing

Acceptation: Test passing, no regressions

UC.3 Documentation Generation

Acteur: Tech Lead
Prérequis: Codebase analysée
Flux principal:

1. Lead: "Generate API documentation"
2. Agent: Analyser endpoints
3. Agent: Générer docs avec exemples
4. Agent: Créer README et API docs
5. Agent: Push to docs branch
6. Résultat: Documentation complete

Acceptation: Docs cover tous les endpoints

UC.4 Multi-Session Project Development

Acteur: Team / Développeur
Prérequis: Projet multi-module
Flux principal:

1. Session A: "Build database models"
2. Session B: "Build API endpoints" (parallel)
3. Session C: "Write tests" (utilise résultats A & B)
4. Résultat: Toutes sessions complétées

Acceptation: Modules intégrent correctement

UC.5 Learning & Onboarding

Acteur: New Developer
Prérequis: Existing codebase
Flux principal:

1. Dev: "Explain this codebase and add a feature"
2. Agent: Analyser structure
3. Agent: Générer explanation
4. Agent: Implémenter feature basée sur patterns existants
5. Résultat: Feature + understanding

Acceptation: Feature works, code follows conventions

ARCHITECTURE SERVICES

Service 1: Agent Core Engine

- **Responsabilité**: Task reasoning, decomposition, execution
- **Inputs**: Task description, context
- **Outputs**: Execution trace, results
- **Dependencies**: Claude API, Sandbox, Session Manager
- **Protocol**: Internal async/event-based

Service 2: Code Generation Service

- **Responsabilité**: Generate code via Claude
- **Inputs**: Requirements, context, examples
- **Outputs**: Code string, metadata
- **Dependencies**: Claude API, Config
- **Protocol**: REST to Claude API

Service 3: Code Execution Service

- **Responsabilité**: Safe code execution
- **Inputs**: Code, language, timeout
- **Outputs**: stdout, stderr, exit code
- **Dependencies**: Node.js, Python, Docker (optionnel)
- **Protocol**: Subprocess communication

Service 4: Session Manager

- **Responsabilité**: Session lifecycle & persistence
- **Inputs**: Session commands (create, load, save)
- **Outputs**: Session data
- **Dependencies**: SQLite, File system
- **Protocol**: Sync/async queries

Service 5: Git Integration Service

- **Responsabilité**: Git operations
- **Inputs**: Git commands (clone, commit, push)
- **Outputs**: Git results, GitHub API calls
- **Dependencies**: Git CLI, GitHub API
- **Protocol**: Shell commands + REST API

Service 6: CLI Interface

- **Responsabilité**: User-facing CLI
- **Inputs**: CLI commands & arguments
- **Outputs**: Formatted output, logs
- **Dependencies**: Agent Core, Session Manager
- **Protocol**: Command parsing + IPC

Service 7: Desktop Application (Electron)

- **Responsabilité**: GUI desktop app
- **Inputs**: User actions (clicks, text input)
- **Outputs**: Rendered UI
- **Dependencies**: CLI backend, WebSocket
- **Protocol**: IPC + WebSocket

INTÉGRATIONS EXTERNES

INT.1 Claude API (Anthropic)

- **Purpose**: Code generation & reasoning
- **Endpoint**: https://api.anthropic.com/v1/messages
- **Authentication**: API key header
- **Rate Limits**: 1000 req/day (free)
- **Fallback**: Queue requests, retry with backoff

INT.2 GitHub API

- **Purpose**: Repository operations, PR creation
- **Endpoint**: https://api.github.com
- **Authentication**: PAT or OAuth
- **Operations**: Clone, push, create PR
- **Fallback**: Local git only, sync later

INT.3 npm Registry

- **Purpose**: Dependency management
- **Endpoint**: https://registry.npmjs.org
- **Operations**: Search, install, update
- **Fallback**: Use cached versions

INT.4 MCP Servers

- **Purpose**: Extended functionality
- **Protocol**: Model Context Protocol
- **Examples**: GitHub MCP, Database MCP
- **Fallback**: Fall back to direct API calls

ROADMAP PRODUIT

Phase 1: MVP (v2.7 - Current)

- ✅ Task execution with Claude
- ✅ Code generation & testing
- ✅ Session persistence
- ✅ Git integration
- ✅ CLI interface
- ✅ Desktop app (Electron)

Phase 2: Enhancement (v3.0 - Q2 2026)

- [ ] Multi-model support (Claude + Gemini)
- [ ] Advanced reasoning (chain-of-thought)
- [ ] Performance optimizations
- [ ] Web dashboard
- [ ] Team collaboration
- [ ] Plugin system

Phase 3: Enterprise (v4.0 - Q4 2026)

- [ ] Self-hosted deployment
- [ ] SSO & SAML
- [ ] Audit logging
- [ ] SLA monitoring
- [ ] Advanced analytics
- [ ] Custom models support

Phase 4: AI Ecosystem (v5.0 - 2027+)

- [ ] Agentic networks (multi-agent coordination)
- [ ] Specialized agents (frontend, backend, DevOps)
- [ ] Learning system (improve over time)
- [ ] Market place for agents/plugins
- [ ] Open-source ecosystem

APPENDIX: Data Models

Task

```
{
"id": "task-uuid",
"sessionId": "session-uuid",
"description": "string",
"status": "pending|running|completed|failed",
"startTime": "ISO8601",
"endTime": "ISO8601",
"executionTrace": ["step1", "step2"],
"output": "string",
"errors": ["error1", "error2"],
"artifacts": ["file1", "file2"]
}
```

Session

```
{
"id": "session-uuid",
"workspacePath": "string",
"createdAt": "ISO8601",
"updatedAt": "ISO8601",
"tasks": ["task-id1", "task-id2"],
"context": {},
"config": {},
"metadata": {}
}
```

Execution Event

```
{
"timestamp": "ISO8601",
"level": "info|warn|error",
"message": "string",
"context": {},
"sourceFile": "string",
"lineNumber": "number"
}
```

GLOSSAIRE

Terme	Définition
Agent	Autonomous AI system that reasons and executes
Task	High-level objective given to the agent
Session	Execution context with persistent state
Artifact	Generated or modified file/code
Sandbox	Isolated environment for code execution
MCP	Model Context Protocol for tool integration
PAT	Personal Access Token for GitHub auth
MTBF	Mean Time Between Failures
RPO	Recovery Point Objective

CONTACT & SUPPORT

Repository: https://github.com/AndyMik90/Auto-Claude
Author: AndyMik90
Issue Tracker: GitHub Issues
Discussions: GitHub Discussions
License: MIT (assumed based on project maturity)

Document Version: 1.0
Last Updated: 2 janvier 2026
Status: Complete & Approved