SPECIFICATION PRODUIT : Auto-Gemini CLI

Requirements Fonctionnels & Non-Fonctionnels

Version: 1.0.0
Date: 2 janvier 2026
Produit: Auto-Gemini CLI - Autonomous AI Coding Agent with Google Gemini
Fondé sur: Auto-Claude architecture, optimisé pour Gemini API
Public cible: Développeurs, équipes d'engineering, startups tech

---

TABLE DES MATIÈRES

VISION PRODUIT

Énoncé de Vision

Auto-Gemini CLI est un agent IA autonome en ligne de commande pour la programmation et l'ingénierie logicielle, propulsé par l'API Google Gemini. Il automatise les tâches de coding, maintient le contexte multi-session, exécute du code dans un sandbox sécurisé, et s'intègre nativement à Git, GitHub, et l'écosystème Node.js.

Slogan: *"Lightning-fast autonomous coding with Gemini. Build, test, deploy from CLI."*

Principes Fondamentaux

- ✓ **CLI-First**: Optimisé pour terminal, zero GUI bloat
- ✓ **Autonome**: Exécute tâches sans intervention continue
- ✓ **Contextuel**: Maintient contexte cross-session via SQLite
- ✓ **Réaliste**: Exécute du code vrai (sandboxé), pas juste generation
- ✓ **Itératif**: Debug, refactor, improve autonomously
- ✓ **Rapide**: Latence API réduite, streaming optimisé
- ✓ **Transparent**: Logs détaillés, decisions expliquées

Différenciation vs Auto-Claude

Aspect	Auto-Gemini	Auto-Claude	Rationale
Interface	CLI-native	Electron + CLI	Lighter weight, faster
API	Google Gemini	Anthropic Claude	Cost-optimized, high throughput
Context Window	1M tokens (Gemini 2.0)	200K tokens (Claude 3.5)	Better long-context handling
Cost	$0.075/M input	$3/M input	40x cheaper
Latency	1-2s avg	2-3s avg	Gemini optimization
Reasoning	Fast chain-of-thought	Deep reasoning	Speed vs depth tradeoff
Streaming	Native SSE	Native streaming	Both optimized
Target Users	DevOps, CLI enthusiasts	Full-stack teams	Different workflows

---

REQUIREMENTS FONCTIONNELS

RF.1 Agent Core & Reasoning

RF.1.1 Task Execution

- [ ] **Agent doit exécuter des tâches de bout en bout via Gemini**
  - Description: Accepte tâche texte, décompose, génère code, exécute
  - Exemple: "Add error handling to this function" → analyze → generate → test → commit
  - Entrée: Task description, file context, git history
  - Sortie: Code généré, test results, git commits
  - Acceptation: Task complète sans erreurs non-recouvrables

RF.1.2 Multi-Step Reasoning

- [ ] **Agent doit décomposer tâches complexes via Gemini thinking**
  - Entrée: Tâche complexe ("Migrate Express to Fastify")
  - Logique:
    1. Analyser structure actuelle
    2. Identifier étapes de migration
    3. Générer plan détaillé

4. Exécuter étape par étape
5. Valider intégration
- Sortie: Trace complète, timing par étape
- Acceptation: Migration complète, tests passent

RF.1.3 Error Handling & Recovery

- [ ] **Agent doit gérer erreurs et récupérer autonomously**
  - Erreurs gérées: Syntax, runtime, file, permission, timeout
  - Stratégies:
    - Analyser erreur stack
    - Proposer fix basé sur patterns appris
    - Réexécuter
    - Max 3 tentatives par étape
  - Fallback: Escalate to user si irrecoverable
  - Acceptation: Erreur documentée avec potential fixes

RF.1.4 Context Maintenance

- [ ] **Agent doit maintenir contexte cross-session**
  - Contexte:
    - Files du projet + diffs
    - Task history & outcomes
    - Env variables & config
    - Learned patterns & fixes
    - Git refs et branches
  - Storage: SQLite `.auto-gemini/db.sqlite`
  - Load: Automatic au démarrage
  - Acceptation: Full context recovery sans perte

RF.1.5 Learning & Pattern Recognition

- [ ] **Agent doit apprendre patterns réussis**
  - Mécanisme: Store successful approaches
  - Exemple: Si "add validation" a échoué x2 → try different approach
  - Storage: Pattern database (SQLite)
  - Acceptation: Agent reuse patterns in new tasks

RF.2 Code Generation & Execution via Gemini

RF.2.1 Gemini API Integration

- [ ] **Agent doit générer code via Google Gemini API**
  - Modèle: `gemini-2.0-flash` (par défaut) ou `gemini-2.0-pro`
  - Endpoint: `https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-flash:generateContent`
  - Paramètres:
    - Temperature: 0.3-0.5 (déterministe + créativité)
    - Max output: Adaptatif selon task
    - System prompt: Instruct Gemini on objective
    - Safety settings: BLOCK_NONE pour code generation
  - Authentication: API key via `GOOGLE_GEMINI_API_KEY` env var
  - Rate limits: 15 req/min (free tier), handle via queue

- Acceptation: Code généré respecte requirements, no placeholders

RF.2.2 Streaming & Token Counting

- [ ] **Agent doit streamer réponses et compter tokens**
  - Streaming: Server-sent events (SSE) pour live output
  - Token counting: Via `countTokens` endpoint pre-call
  - Latency optimization: Stream tout, render progressivement
  - Display: Real-time code display en CLI
  - Acceptation: Streaming <100ms latency, accurate counts

RF.2.3 Code Sandbox & Execution

- [ ] **Agent doit exécuter code dans sandbox sécurisé**
  - Sandbox: Node.js isolated context (vm module)
  - Langages:
    - JavaScript/TypeScript: Native Node.js
    - Python: Via `child_process` + Python interpreter
    - Bash: Whitelist commands only
    - Go: Compile + execute (optionnel)
  - Restrictions:
    - No fs write outside project dir
    - No network (sauf API whitelist)
    - Max 30s execution time
    - Max 256MB memory per process
  - Capture: stdout, stderr, exit code, timing
  - Acceptation: Safe execution, no system access

RF.2.4 Testing & Validation

- [ ] **Agent doit tester code généré automatiquement**
  - Tests:
    - Unit tests (Jest, Mocha, pytest)
    - Integration tests avec mocks
    - Manual validation scripts
  - Requirement: >80% tests passing avant commit
  - Report: Test count, pass/fail, coverage %, timing
  - Rerun: Auto-rerun si test fails après fix
  - Acceptation: Test report complet, >80% passing

RF.2.5 Code Quality Analysis

- [ ] **Agent doit analyser qualité du code**
  - Outils:
    - Linting: ESLint, Pylint, Clippy (Rust)
    - Type checking: TypeScript strict, mypy
    - Formatting: Prettier, Black
    - Complexity: Cyclomatic complexity metrics
  - Métriques:
    - Errors: 0 mandatory
    - Warnings: Warn si >5
    - Complexity: Warn si >10
    - Duplication: Flag si >10%

- Report: JSON exportable
- Acceptation: Code passe all quality gates

RF.2.6 Git Integration & Commits

- [ ] **Agent doit versionner via Git**
  - Opérations:
    - Clone repository
    - Create feature branches
    - Stage, commit, push changes
    - Create PRs (GitHub API)
    - Comment on PRs
  - Commits: Auto-generated messages with context
  - Branches: task-{id} naming
  - PRs: Include test results, diff summary
  - Acceptation: Clean git history, PRs on GitHub

RF.3 CLI Interface & UX

RF.3.1 Command-Line Interface

- [ ] **Agent doit exposer CLI complète et intuitive**
  - Framework: Node.js + yargs/commander
  - Commandes principales:
    auto-gemini init [--dir] # Initialiser workspace
    auto-gemini task new "<desc>" # Créer tâche
    auto-gemini task run <id> # Exécuter tâche
    auto-gemini task list [--status] # Lister tâches
    auto-gemini task cancel <id> # Annuler tâche
    auto-gemini session list # Lister sessions
    auto-gemini session show <id> # Détails session
    auto-gemini session export <id> # Exporter JSON
    auto-gemini status # Health check
    auto-gemini config set <key> <val> # Configuration
    auto-gemini logs [--tail] [--grep] # View logs
    auto-gemini debug <task-id> # Debug mode
  - Help: Built-in help pour chaque commande
  - Acceptation: All commands functional, clear help

RF.3.2 Real-time Logging & Output

- [ ] **Agent doit afficher logs en temps réel**
  - Features:
    - Streaming output de Gemini API
    - Colorized logs (errors=red, success=green, info=blue)
    - Structured JSON logs
    - Searchable history (grep support)
    - Tail mode (dernières N lignes)
    - Log levels: debug, info, warn, error
  - Storage: Logs dans .auto-gemini/logs/
  - Performance: <100ms latency pour display
  - Acceptation: Real-time, searchable, properly colored

RF.3.3 Progress Indicators & Spinners

- [ ] **Agent doit afficher progress visuel**
  - Features:
    - Animated spinners pour tasks running
    - Progress bars pour multi-step
    - Percentage completion
    - ETA estimates
    - Status badges (pending, ▶ running, ✓ done, ✗ failed)
  - Libraries: chalk, ora, cli-progress
  - Acceptation: Clear visual feedback, responsive UI

RF.3.4 Interactive Mode & REPL

- [ ] **Agent doit supporter mode REPL interactif**
  - Fonctionnalités:
    - Execute code snippets live
    - Inspect variables & state
    - Modify context interactively
    - Ask agent questions
    - Persist REPL session
  - Context: Maintained cross-commands
  - Storage: Session saved for history
  - Acceptation: REPL responsive, context preserved

RF.4 Session Management & Persistence

RF.4.1 Session Persistence

- [ ] **Agent doit créer sessions persistantes**
  - Session inclut:
    - UUID unique
    - Workspace path
    - Task list & outcomes
    - Context snapshot
    - Metadata (start time, duration, status)
  - Storage: SQLite `.auto-gemini/sessions.db`
  - Recovery: Load previous session by ID
  - Cleanup: Archive old sessions (>1 year)
  - Acceptation: Session recoverable, no data loss

RF.4.2 Multi-Session Concurrency

- [ ] **Agent doit supporter sessions parallèles**
  - Isolation: Separate working dirs per session
  - Locking: Prevent race conditions on shared files
  - Max: 10 concurrent (configurable)
  - Sync: Merge changes intelligently
  - Acceptation: Parallel execution, clean merges

RF.4.3 Session History & Replay

- [ ] **Agent doit supporter replay de sessions**
  - Fonctionnalité: Re-execute session complète
  - History: Full audit trail accessible
  - Export: JSON format avec tous événements
  - Determinism: Same inputs → same outputs (mostly)
  - Acceptation: Replay complète, résultats identiques

RF.4.4 Session Summary & Reporting

- [ ] **Agent doit générer résumés de session**
  - Contenu:
    - Tasks executed (count, success rate %)
    - Code generated (LOC, files touched)
    - Errors encountered & fixes applied
    - Time analysis (per task, total, average)
    - Git commits created
    - Cost estimation (API tokens used)
  - Formats: Markdown, JSON, CSV
  - Export: `.auto-gemini/reports/{session-id}.md`
  - Acceptation: Comprehensive report, exportable

RF.5 GitHub & Git Integration

RF.5.1 GitHub API Integration

- [ ] **Agent doit intégrer avec GitHub**
  - Capacités:
    - Authenticate via PAT (GITHUB_TOKEN env var)
    - Clone repositories
    - Create branches & commits
    - Push to remote
    - Create pull requests with descriptions
    - Review & comment on PRs
    - Close issues via commits
  - Rate limits: Respecter GitHub API limits
  - Error handling: Graceful fallback if GitHub down
  - Acceptation: PRs created, commits pushed, issues closed

RF.5.2 Git Workflow

- [ ] **Agent doit follow feature branch workflow**
  - Workflow:
    1. Create task-specific branch (task-{id})
    2. Make commits avec descriptive messages
    3. Run tests avant push
    4. Push to remote
    5. Create PR avec summary
    6. Auto-comment avec results
  - Commit messages: Auto-generated pero meaningful
  - PR templates: Structured descriptions
  - Acceptation: Clean workflow, PRs reviewable

RF.5.3 Merge Conflict Resolution

- [ ] **Agent doit gérer merge conflicts**
  - Detection: Automated conflict detection
  - Resolution: Attempt auto-resolution where safe
  - Fallback: Ask user ou escalate
  - Testing: Re-run tests post-merge
  - Acceptation: Conflicts resolved ou escalated properly

RF.6 Package & Dependency Management

RF.6.1 npm Integration

- [ ] **Agent doit gérer dépendances npm**
  - Opérations:
    - `npm install` automatic si missing deps
    - `npm update` avant running tasks
    - `npm run` pour npm scripts
    - Parse `package.json` & `package-lock.json`
  - Detection: Auto-detect missing packages
  - Installation: Automatic avec version resolution
  - Acceptation: Dépendances managed, lock files updated

RF.6.2 Dependency Scanning

- [ ] **Agent doit scanner vulnerabilities**
  - Tool: `npm audit` integration
  - Report: Vulnerabilities found & fixes available
  - Auto-fix: Apply security patches if safe
  - Notification: Warn user of breaking changes
  - Acceptation: Vulnerabilities identified & reported

RF.7 Configuration & Customization

RF.7.1 Configuration Management

- [ ] **Agent doit permettre configuration**
  - Config file: `~/.auto-gemini/config.json`
  - Paramètres:
    - `apiKey`: Google Gemini API key
    - `model`: Modèle (default: gemini-2.0-flash)
    - `temperature`: 0.0-1.0
    - `maxTokens`: Max tokens per call
    - `workspace`: Default workspace path
    - `gitHub.token`: GitHub PAT
    - `sandbox.timeout`: Max execution time (30s default)
    - `sandbox.memoryLimit`: Max memory (256MB default)
    - `logLevel`: debug|info|warn|error
    - `streamingEnabled`: true (default)
  - Sources: File + env vars + CLI flags (priority)
  - Validation: Check config validity on startup
  - Acceptation: Config properly applied

RF.7.2 Environment Variables

- [ ] **Agent doit utiliser env variables**
  - Requises:
    - GOOGLE_GEMINI_API_KEY: Gemini API key
  - Optionnelles:
    - GITHUB_TOKEN: GitHub PAT
    - AUTO_GEMINI_WORKSPACE: Workspace path
    - AUTO_GEMINI_MODEL: Model override
    - LOG_LEVEL: Logging level
    - SANDBOX_TIMEOUT: Execution timeout
  - Loading: On startup, logged (sans exposer keys)
  - Acceptation: Env vars loaded & applied

RF.7.3 Custom System Prompts

- [ ] **Agent doit supporter custom prompts**
  - Fichiers:
    - ~/.auto-gemini/system.prompt: System prompt custom
    - ~/.auto-gemini/task-template.md: Task template
  - Usage: Intégré dans Gemini calls
  - Validation: Check prompt validity
  - Acceptation: Custom prompts used in generations

RF.8 Monitoring & Analytics

RF.8.1 Metrics Collection

- [ ] **Agent doit collecter usage metrics**
  - Métriques:
    - Tasks executed (count, success rate)
    - Code generated (LOC, files modified)
    - Errors encountered (count, types)
    - Execution time (per task, average)
    - API calls (count, tokens, cost)
    - Cost tracking ($ spent per session)
  - Storage: SQLite metrics database
  - Retention: 1 year (configurable)
  - Export: JSON/CSV reports
  - Acceptation: Metrics collected & exportable

RF.8.2 Performance Monitoring

- [ ] **Agent doit monitorer performances**
  - Métriques:
    - Gemini API latency (avg, p95, p99)
    - Code execution time
    - Memory usage (peak, average)
    - Disk usage (workspace, db, logs)
  - Alertes: Warn si anomalies
  - Dashboard: CLI command auto-gemini stats
  - Acceptation: Performance data available

RF.8.3 Error Tracking & Analytics

- [ ] **Agent doit tracker les erreurs**
  - Data:
    - Error type & message
    - Stack trace & context
    - Task & session info
    - Recovery attempt & result
  - Storage: Error database (SQLite)
  - Reports: Error frequency, root causes
  - Trend analysis: Identify patterns
  - Acceptation: Errors tracked, reportable

RF.9 Security & Privacy

RF.9.1 API Key Management

- [ ] **Agent doit gérer API keys securely**
  - Storage: Never in logs ou version control
  - Masking: Show juste last 4 chars (gABC...XYZ)
  - Rotation: Support key changes
  - Validation: Check key validity au startup
  - Acceptation: Keys protected, never exposed

RF.9.2 Sandbox Isolation

- [ ] **Agent doit isoler code execution**
  - Isolation: Separate Node.js VM context
  - Restrictions:
    - No fs write outside project dir
    - No network (sauf whitelist)
    - No env var access (sauf whitelist)
    - No process spawning (sauf via exec wrapper)
  - Timeout: 30s max per command
  - Memory: 256MB limit
  - Acceptation: Safe execution, isolated context

RF.9.3 Input Validation & Sanitization

- [ ] **Agent doit valider toutes entrées**
  - Validation:
    - CLI arguments (type, length, format)
    - Config values (ranges, valid options)
    - API responses (schema validation)
    - User input (sanitization)
  - Security: Prevent injection attacks
  - Escaping: Proper escaping pour shell commands
  - Acceptation: No injection vulnerabilities

RF.9.4 Audit Logging

- [ ] **Agent doit logger toutes actions importantes**
  - Events:
    - Task creation & execution
    - Code generation & execution
    - Git operations
    - Config changes
    - API calls (sans exposer keys)
  - Format: ISO timestamp, action, actor, result
  - Storage: Structured logs (JSON)
  - Retention: 1 year default
  - Acceptation: Complete audit trail

REQUIREMENTS NON-FONCTIONNELS

RNF.1 Performance

RNF.1.1 API Response Time

- [ ] Gemini API calls: <2s avg (vs Claude 5s)
- [ ] Streaming latency: <100ms
- [ ] Code execution: <30s timeout
- [ ] CLI command startup: <300ms
- [ ] Token counting: <500ms
- Metric: Monitor via logging, export stats

RNF.1.2 CLI Responsiveness

- [ ] CLI input response: <100ms
- [ ] Log display: <50ms per new log
- [ ] Command completion: <2s for most
- Metric: Measure via performance profiling

RNF.1.3 Memory Usage

- [ ] Idle process: <80MB RAM
- [ ] Single task running: <300MB
- [ ] 10 concurrent sessions: <800MB
- Metric: Monitor process memory

RNF.1.4 Startup Time

- [ ] CLI start: <300ms
- [ ] Session load: <500ms
- Metric: Time from invocation to ready

RNF.2 Reliability & Availability

RNF.2.1 Uptime & Error Rate

- [ ] Target: 99% availability (open-source acceptable)
- [ ] Unhandled errors: <0.5%
- [ ] Failed tasks due to system: <5%
- [ ] API errors: <2%
- Metric: Error rate monitoring

RNF.2.2 Data Persistence

- [ ] No session data loss on crash
- [ ] Automatic SQLite backups
- [ ] Recovery point objective (RPO): <1h
- Metric: Test via failure scenarios

RNF.2.3 Graceful Degradation

- [ ] API down → Queue tasks, retry with backoff
- [ ] Network down → Continue local work
- [ ] Sandbox failure → Fallback mode
- Metric: Degradation mode testing

RNF.3 Scalability

RNF.3.1 Concurrent Sessions

- [ ] Support: 10+ concurrent sessions
- [ ] No performance degradation
- [ ] Proper resource isolation
- Metric: Load testing

RNF.3.2 Data Growth

- [ ] Database supports 10K+ sessions
- [ ] Logs up to 100GB without issues
- [ ] Query time <1s even with large data
- Metric: Database benchmarking

RNF.3.3 Code Handling

- [ ] Projects >100K LOC supported
- [ ] Analysis time <10s
- [ ] Memory scales linearly
- Metric: Large project testing

RNF.4 Maintainability

RNF.4.1 Code Quality

- [ ] Type safety: 100% (TypeScript strict)
- [ ] Test coverage: >80%
- [ ] Documentation: Comprehensive JSDoc
- [ ] Linting: Zero errors
- Metric: Coverage reports, linter results

RNF.4.2 Modularity

- [ ] Clear separation of concerns
- [ ] Well-defined interfaces
- [ ] Minimal dependencies
- [ ] Easy to extend
- Metric: Cyclomatic complexity <10

RNF.4.3 Logging & Debugging

- [ ] Debug mode with verbose logs
- [ ] Structured JSON logging
- [ ] Source maps for stack traces
- [ ] Execution replay capability
- Metric: Logging completeness

RNF.5 Security

RNF.5.1 Authentication

- [ ] Gemini API: Valid API key required
- [ ] GitHub: OAuth2 or PAT validation
- [ ] No hardcoded credentials
- [ ] Secure secret storage
- Metric: Security audit

RNF.5.2 Data Protection

- [ ] No sensitive data in logs
- [ ] Secure temp file deletion
- [ ] GDPR compliant if applicable
- Metric: Data protection audit

RNF.5.3 Vulnerability Management

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

RNF.6 Usability

RNF.6.1 Onboarding

- [ ] Quick setup: <5 minutes
- [ ] `auto-gemini init` wizard
- [ ] Built-in help & examples
- [ ] Documentation >90% coverage
- Metric: User feedback

RNF.6.2 Error Messages

- [ ] Clear & actionable messages
- [ ] Suggestions for fixes
- [ ] Links to docs
- [ ] No jargon
- Metric: UX testing

RNF.6.3 Accessibility

- [ ] Full CLI keyboard navigation
- [ ] High contrast mode support
- [ ] Screen reader compatible
- [ ] Clear visual hierarchy
- Metric: Accessibility audit

RNF.7 Compatibility

RNF.7.1 Platform Support

- [ ] Linux (Ubuntu 20.04+)
- [ ] macOS 11+
- [ ] Windows 10+ (via WSL or native)
- [ ] Node.js 18+
- Metric: Test on all platforms

RNF.7.2 Language Support

- [ ] JavaScript/TypeScript (native)
- [ ] Python 3.9+ (via subprocess)
- [ ] Bash/POSIX shell
- [ ] Go (optional)
- [ ] Rust (optional)
- Metric: Test per language

RNF.7.3 Dependency Management

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

RNF.8 Deployment & Operations

RNF.8.1 Installation

- [ ] Single command: `npm install -g auto-gemini`
- [ ] Automatic setup
- [ ] Cross-platform binaries
- [ ] Self-update capability
- Metric: Installation success rate

RNF.8.2 Configuration

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

RNF.8.3 Observability

- [ ] Health check command
- [ ] Structured logging (JSON)
- [ ] Performance metrics export
- [ ] Debug profiling support
- Metric: Monitoring coverage

RNF.8.4 Updates & Maintenance

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

CONTRAINTES & LIMITATIONS

Contraintes Techniques

Contrainte	Valeur	Justification
API Rate Limit	15 req/min (free tier)	Gemini API free limit
Max Code Execution	30 secondes	Security & resource
Context Window	1M tokens (Gemini 2.0)	API limit
Sandbox Memory	256MB	Prevent DoS
File Size	100MB	Processing limit
Session Retention	1 year	Storage management
Max Concurrent Tasks	10 (configurable)	Resource bound
Output Token Limit	16K tokens max	API limit

Limitations Fonctionnelles

1. **No GUI** - CLI-only interface
2. **No mobile** - Desktop/server only
3. **No real-time collab** - Single-user focus
4. **No GPU** - CPU-based inference
5. **No cloud DB** - SQLite only
6. **No auto-deployment** - User manages production
7. **No team management** - Individual use focus

Limitations de Performance

- Complex tasks (>1000 LOC) = 10-30s generation
- Large context (>500K tokens) = higher latency
- 50+ sessions = degraded performance
- Large projects (>500K LOC) = slower analysis

Limitations d'API

- Free tier: 15 requests/minute
- Paid tier: Subject to quota management
- Cannot exceed 1M context window
- Rate limiting without burst capability

CAS D'USAGE

UC.1 Quick Code Fixes

Acteur: Developer on deadline
Prérequis: Repository, API key
Flux:

1. Dev: `auto-gemini task new "Fix TypeError in utils.js"`
2. Agent: Analyzes error, generates fix
3. Agent: Runs tests
4. Agent: Creates commit & PR
5. Résultat: PR ready for review

Acceptation: PR functional, tests passing

UC.2 Batch Refactoring

Acteur: Tech lead
Prérequis: Codebase identified for refactor
Flux:

1. Lead: `auto-gemini task new "Convert all var to const"`
2. Agent: Analyzes codebase
3. Agent: Generates batch changes
4. Agent: Runs full test suite
5. Agent: Creates PR with summary
6. Résultat: Refactoring complete

Acceptation: No regressions, coverage maintained

UC.3 Feature Development

Acteur: Backend developer
Prérequis: Requirements documented
Flux:

1. Dev: `auto-gemini task new "Add JWT authentication"`
2. Agent: Breaks down into steps
3. Agent: Generates auth module
4. Agent: Writes tests
5. Agent: Creates API docs
6. Résultat: Feature + docs complete

Acceptation: Tests >80%, docs comprehensive

UC.4 Multi-Session Development

Acteur: Team
Prérequis: Feature branches planned
Flux:

Session A: Backend

auto-gemini task new "Build user service"

Session B: API (parallel)

auto-gemini task new "Build REST endpoints"

Session C: Tests (uses results from A & B)

auto-gemini task new "Write integration tests"
6. Résultat: Parallel development, clean merges

Acceptation: All modules integrate correctly

UC.5 Documentation Generation

Acteur: Developer
Prérequis: API code complete
Flux:

1. Dev: `auto-gemini task new "Generate API documentation"`
2. Agent: Analyzes all endpoints
3. Agent: Generates OpenAPI spec
4. Agent: Generates README with examples
5. Agent: Commits docs
6. Résultat: Docs ready

Acceptation: All endpoints documented

UC.6 Debugging & Troubleshooting

Acteur: Developer
Prérequis: Failing test or bug report
Flux:

1. Dev: `auto-gemini task new "Debug: Database timeout on large queries"`
2. Agent: Analyzes query patterns
3. Agent: Proposes optimization
4. Agent: Tests with real data
5. Agent: Benchmarks improvement
6. Résultat: Root cause identified, fix applied

Acceptation: Performance improves >50%

ARCHITECTURE SERVICES

Service 1: Agent Core Engine

- **Responsabilité**: Task decomposition, execution orchestration
- **Inputs**: Task description, project context
- **Outputs**: Execution trace, results
- **Dependencies**: Gemini API, Sandbox, Session Store
- **Protocol**: Internal async/event-based

Service 2: Gemini Code Generation

- **Responsabilité**: Generate code via Google Gemini
- **Inputs**: Requirements, context, examples
- **Outputs**: Code, metadata, token usage
- **Dependencies**: Gemini API, Token counter
- **Protocol**: REST + streaming (SSE)

Service 3: Code Sandbox Runtime

- **Responsabilité**: Secure code execution
- **Inputs**: Code, language, timeout
- **Outputs**: stdout, stderr, exit code, timing
- **Dependencies**: Node.js VM, subprocess
- **Protocol**: Sandbox API (internal)

Service 4: Session Manager

- **Responsabilité**: Session lifecycle & persistence
- **Inputs**: Session commands (CRUD)
- **Outputs**: Session data
- **Dependencies**: SQLite, Filesystem
- **Protocol**: Sync/async database queries

Service 5: Git & GitHub Service

- **Responsabilité**: Version control operations
- **Inputs**: Git commands, GitHub API calls
- **Outputs**: Git results, PR info
- **Dependencies**: Git CLI, GitHub API
- **Protocol**: Shell commands + REST

Service 6: CLI Interface

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

Service 7: Configuration & Secrets

- **Responsabilité**: Config management & secret storage
- **Inputs**: Config files, env vars
- **Outputs**: Validated config
- **Dependencies**: Filesystem, env vars
- **Protocol**: Sync config loading

INTÉGRATIONS EXTERNES

INT.1 Google Gemini API

- **Purpose**: Code generation & reasoning
- **Endpoint**: https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-flash:generateContent
- **Authentication**: API key (GOOGLE_GEMINI_API_KEY)
- **Rate Limits**: 15 req/min (free), 60+ (paid)
- **Cost**: $0.075/M input tokens, $0.3/M output
- **Fallback**: Queue requests, retry with backoff

INT.2 GitHub API

- **Purpose**: Repository ops, PR creation
- **Endpoint**: https://api.github.com
- **Authentication**: PAT or OAuth
- **Operations**: Clone, push, PR, issues
- **Rate Limits**: 5K req/hour (auth)
- **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 Node.js Runtime

- **Purpose**: Code execution sandbox
- **Version**: 18+ required
- **Features**: VM module, subprocess
- **Limits**: 30s timeout, 256MB memory

ROADMAP PRODUIT

Phase 1: MVP (v1.0 - Current)

- ✅ Task execution via Gemini
- ✅ Code generation & testing
- ✅ CLI interface (non-GUI)
- ✅ Session persistence (SQLite)
- ✅ Git integration
- ✅ Basic metrics & logging

Phase 2: Enhancement (v1.5 - Q1 2026)

- [ ] Multi-model support (Gemini + Claude)
- [ ] Advanced context handling (1M token optimization)
- [ ] Improved code quality analysis
- [ ] Plugin system
- [ ] Web-based dashboard (optional)
- [ ] Team workspace sharing

Phase 3: Pro Features (v2.0 - Q2 2026)

- [ ] Agentic self-improvement
- [ ] Custom model fine-tuning
- [ ] Advanced analytics & insights
- [ ] Enterprise deployment options
- [ ] SSO & team management
- [ ] Custom integrations (Slack, Jira, etc.)

Phase 4: Ecosystem (v2.5+ - 2026+)

- [ ] Multi-agent coordination
- [ ] Specialized agent types (frontend, backend, DevOps)
- [ ] Learning system & pattern marketplace
- [ ] Open-source plugin marketplace
- [ ] Community contributions & extensions

DATA MODELS

Task

{
"id": "task-uuid",
"sessionId": "session-uuid",
"description": "string",
"status": "pending|running|completed|failed",
"startTime": "ISO8601",
"endTime": "ISO8601",
"executionSteps": [
{
"step": "integer",
"description": "string",
"status": "pending|running|completed|failed",

```
"output": "string",
"timing": "milliseconds"
}
],
"output": "string",
"errors": ["error1", "error2"],
"artifacts": ["file1", "file2"],
"costEstimate": {
"inputTokens": "integer",
"outputTokens": "integer",
"estimatedCost": "float"
}
}
```


Session


```
{
"id": "session-uuid",
"workspacePath": "string",
"createdAt": "ISO8601",
"updatedAt": "ISO8601",
"tasks": ["task-id1", "task-id2"],
"context": {
"projectName": "string",
"description": "string",
"techStack": ["javascript", "node.js"],
"gitRemote": "string"
},
"config": {
"model": "string",
"temperature": "float",
"maxTokens": "integer"
},
"stats": {
"totalTasks": "integer",
"successCount": "integer",
"failureCount": "integer",
"totalTokensUsed": "integer",
"estimatedCost": "float",
"durationMs": "integer"
}
}
```


Log Entry


```
{
"timestamp": "ISO8601",
"level": "debug|info|warn|error",
"message": "string",
"context": {},
"sessionId": "session-uuid",
```

"taskId": "task-uuid",
"sourceFile": "string",
"lineNumber": "integer"
}

GLOSSAIRE

Terme	Définition
Agent	Autonomous AI using Gemini for reasoning
Task	High-level objective for agent
Session	Persistent execution context
Artifact	Generated or modified file
Sandbox	Isolated code execution environment
Streaming	Server-sent events for real-time output
PAT	Personal Access Token for GitHub
RPO	Recovery Point Objective
Gemini API	Google's generative AI API

CONTACT & SUPPORT

Repository: https://github.com/AndyMik90/Auto-Gemini-CLI
Author: AndyMik90
Issues: GitHub Issues
Discussions: GitHub Discussions
License: MIT
API Docs: https://ai.google.dev/gemini-api/docs

Document Version: 1.0.0
Last Updated: 2 janvier 2026
Status: Complete & Approved