



DEVELOPMENT HANDBOOK

Guida Completa allo Sviluppo con Claude Code + Cowork
Dal Prototipo Rev 5.0 al Pilot 2.000 Studenti

Architettura OIOD Rev 2.0 • 4 Strati • 5 Sottosistemi MMS
Stack: Python/FastAPI • PostgreSQL+pgvector • Neo4j • Redis • Gemini (Vertex AI)

Documento Autosufficiente — Febbraio 2026

Synthetic Data S.r.l. — Confidenziale

1. CONTESTO: Da Dove Partiamo

Questo documento è la guida operativa completa per sviluppare QWIKKEN — dal prototipo già funzionante (Rev 5.0 su Google AI Studio) fino al pilot da 2.000 studenti con architettura enterprise. È pensato per essere autosufficiente: contiene tutto il necessario per capire il progetto, configurare gli strumenti, e iniziare a scrivere codice dal giorno uno.

1.1 Punto di Partenza: QWIKKEN BASE Rev 5.0

Non partiamo da zero. QWIKKEN BASE Rev 5.0 è un prototipo operativo con frontend React/TypeScript che gira su Google AI Studio. Include 12 moduli funzionanti: DOC (Syllabus), Lecture Weaver, Web Scout, Tutor, Socrates AI, Nexus Graph, Test con Quiz, Simulazione Esame, Life Planner, Tesi, Linguistic Gym, Panic Mode.

Il lavoro di sviluppo non è costruire QWIKKEN da zero.

È trasformare il prototipo in un sistema enterprise con: backend scalabile (FastAPI su Cloud Run), database relazionale multi-tenant (PostgreSQL + pgvector), Knowledge Graph (Neo4j), architettura OIOD a 4 strati per i costi, e compliance governance (Genoma + XAI Logs).

1.2 Obiettivo: Pilot-Ready in 16 Settimane

Parametro	Valore
Studenti target	2.000 iscritti a 5-6 corsi
Moduli prioritari (P0)	DOC + Tutor + Socrates + Test con Quiz
Moduli P1 (dalla sett. 6)	Simulazione Esame (ARENA)
Architettura	OIOD Rev 2.0 a 4 strati (Fabbrica / Bibliotecario / Assistente Senior / Professore)
Frontend	Evoluzione di QWIKKEN BASE Rev 5.0 (React/TypeScript)
Backend	Python 3.12, FastAPI, Cloud Run
Database	PostgreSQL 16 + pgvector + Neo4j 5.x + Redis 7
LLM	Gemini 2.0 Flash (Bibliotecario), Gemini 3 Flash (Ass. Senior), Gemini 3 Pro (Professore)
Budget costi vivi	€5.200 (infra + LLM + tooling)

2. TOOL DI SVILUPPO: Claude Code + Cowork

Il team di sviluppo usa due strumenti Anthropic complementari che riducono lo sforzo complessivo del 67%. Questo capitolo spiega cosa sono, come si configurano, e quando usare l'uno o l'altro.

2.1 Claude Code: Coding Agentico

Claude Code è un tool a riga di comando che permette a Claude di operare direttamente nel repository del progetto. Non è un autocompletamento: è un agente che legge l'intero codebase, comprende l'architettura, scrive codice, crea test, e gestisce file — tutto da terminale.

Caratteristica	Dettaglio
Installazione	npm install -g @anthropic-ai/clause-code (richiede Node.js 18+)
Lancio	cd qwikken/ && claude (legge automaticamente CLAUDE.md)
Modello	Claude Sonnet 4.5 (default) o Opus 4.6 su Max
Context window	Fino a 1M token (intero codebase in contesto)
Cosa fa	Scrive codice, crea file, esegue test, gestisce git, refactoring multi-file
CLAUDE.md	File nella root del progetto che Claude Code legge automaticamente ad ogni sessione. Contiene architettura, convenzioni, stack, sprint corrente
Agent Teams (beta)	Più istanze Claude Code lavorano in parallelo su task diversi

2.2 Claude Cowork: Automazione Desktop

Cowork è un agente AI desktop rilasciato a gennaio 2026 (macOS) e febbraio 2026 (Windows). Opera su file locali, esegue task multi-step, e si integra con servizi esterni via MCP (Model Context Protocol). È il complemento di Claude Code per tutto ciò che non è coding puro.

Caratteristica	Dettaglio
Disponibilità	macOS (12 gen 2026), Windows (10 feb 2026)
Requisito	Abbonamento Claude Max (\$100 o \$200/mese)
Modello	Claude Opus 4.6 (1M token context window)
Plugin	11 plugin open-source (sales, legal, finance, marketing, dev, data analysis...)
Cosa fa	Legge/scrive/crea file locali, organizza cartelle, genera report, analizza dati, crea documenti
MCP Connectors	Google Drive, Gmail, GitHub, Slack, Asana e altri via Model Context Protocol
Chrome Integration	Può navigare il web, compilare form, fare scraping (via Claude in Chrome)

2.3 Quando Usare Cosa

Task	Tool	Perché
Scrivere un endpoint FastAPI	Claude Code	Lavora direttamente nel repo, conosce lo stack e il codebase
Generare 36.000 prompt per la Fabbrica	Claude Code	Batch scripting nel terminale, accesso a file JSON/YAML
QA su 3.600 artefatti (consistenza, formato)	Cowork	Analisi batch su file locali, report strutturato
Scrivere test suite per il routing engine	Claude Code	Crea/esegue test nel progetto, vede i risultati
Creare documentazione API da codebase	Cowork	Legge il codice + genera doc formattati (DOCX, MD)
Configurare Cloud Monitoring dashboard	Cowork	Template YAML, organizzazione file configurazione
Debug di un errore in produzione	Claude Code	Legge log, stack trace, modifica codice, riesegue
Analisi costi GCP mensile	Cowork	Legge export billing CSV, genera report Excel
Refactoring multi-file	Claude Code	Rinomina, sposta, aggiorna import su tutto il progetto
Preparare slide per il CEO/ateneo	Cowork	Genera PPTX/DOCX da dati e template

Regola pratica: se il task coinvolge il repository Git e richiede esecuzione di codice, usa Claude Code. Se coinvolge file locali, documenti, report, o organizzazione, usa Cowork.

3. LICENZE E COSTI AI TOOLING

3.1 Piani Claude (Febbraio 2026)

Piano	Costo	Claude Code	Cowork	Per Chi
Pro	\$20/mese	Sì (45 msg/5h window)	Sì (limitato)	Sviluppo leggero, 2-3h/giorno
Max 5x	\$100/mese	Sì (5x capacità)	Sì (completo)	Sviluppo intensivo full-day
Max 20x	\$200/mese	Sì (20x capacità)	Sì (completo)	Maratone di coding, sprint critici
Team (Standard)	\$25/utente/mese	No	No	Chat e collaborazione
Team (Premium)	\$150/utente/mese	Sì	Sì	Team enterprise con audit

3.2 Configurazione Consigliata per il Pilot

Ruolo	Piano	Costo/Mese	Mesi	Totale	Uso Principale
CTO / Tech Lead	Max 5x (\$100)	€92	4	€370	Claude Code (architettura, routing, Fabbrica, infra)
Full Stack Developer	Max 5x (\$100)	€92	4	€370	Claude Code (API, frontend, integrazione, test)
ML / AI Engineer	Max 5x (\$100)	€92	4	€370	Claude Code (prompt eng.) + Cowork (QA artefatti)
Product / Customer Success	Pro (\$20)	€18.50	4	€74	Cowork (doc, report, onboarding)
CEO	— (usa claude.ai)	€0	4	€0	Già incluso nel piano esistente
TOTALE				€1.184	Budget AI tooling per 4 mesi

Nota: il costo del tooling AI è una frazione del risparmio generato. Senza Claude Code + Cowork, servirebbero ~2.85 FTE aggiuntivi (almeno 1 assunzione a €45K/anno). Il tooling AI costa €1.184 e risparmia €17.956 netti. **ROI: 15x.**

4. IL CLAUDE.md: Il Cervello del Progetto

Il file più importante del repository non è main.py. È CLAUDE.md. È il file che Claude Code legge automaticamente ad ogni sessione per capire il contesto del progetto. Più è ricco e preciso, più Claude Code produce codice di qualità.

4.1 Struttura del CLAUDE.md

Il CLAUDE.md per QWIKKEN è stato già preparato (18KB) e contiene:

Sezione	Contenuto	Perché Serve a Claude Code
Vision	2-3 paragrafi sulla missione di QWIKKEN	Oriente le decisioni architettoniche
Stack Tecnologico	Tabella completa: Python, FastAPI, PostgreSQL, Neo4j, Redis, Vertex AI, Next.js...	Claude Code sa quali librerie usare
Architettura 5 Sottosistemi	SENSORIUM, SYNAPSE, CALCULATOR, CORTEX, SENTINEL con responsabilità e cartelle	Sa dove mettere ogni file e come i pezzi si collegano
Genoma 5 Livelli	L1 Governance → L2 Domain → L3 User → L4 Contractual → L5 IP	Implementa la gerarchia di vincoli correttamente
Offline Intelligence (OIOD)	4 strati: Fabbrica, Bibliotecario, Assistente Senior, Professore	Sa come implementare il routing e il model cascading
Multi-Tenant Architecture	Row-Level Security, label isolation Neo4j, Redis prefix, tenant_id su ogni query	Mai scrive una query senza filtro tenant
Struttura Progetto	Albero completo delle cartelle e file con descrizione	Sa dove creare nuovi file
Convenzioni Codice	Python async, Pydantic v2, snake_case, structlog, Google docstrings	Produce codice consistente con il progetto
Variabili Ambiente	Template .env con tutte le variabili necessarie	Configura correttamente le connessioni
Sprint Corrente	Checklist del lavoro in corso	Sa cosa deve fare nella sessione
Contesto Business	Pricing €99.99, COGS target €20.62, margine 79%	Oriente le scelte di ottimizzazione costi

4.2 Aggiornamento del CLAUDE.md

Regola critica: aggiorna il CLAUDE.md alla fine di ogni sprint. Aggiungi cosa è stato costruito, cosa funziona, dove sono i file. Claude Code lo rilegge ad ogni sessione — se il file è obsoleto, genera codice obsoleto.

Sezioni da aggiornare dopo ogni sprint: “Sprint Corrente” (checklist della prossima fase), “Struttura Progetto” (nuovi file/cartelle), e qualsiasi decisione architettonica presa durante lo sprint.

5. STACK TECNOLOGICO COMPLETO

Layer	Tecnologia	Versione	Ruolo in QWIKKEN
Backend Framework	Python + FastAPI + Uvicorn	3.12 / 0.115+	API REST async, SSE streaming per chat
ORM + Migrations	SQLAlchemy 2.0 + Alembic	2.0+	Modelli multi-tenant, migration management
Database Relazionale	PostgreSQL + pgvector	16 + 0.7	Dati strutturati, embedding search, artefatti
Knowledge Graph	Neo4j	5.x	Grafo semantico concetti/documenti/utenti
Cache	Redis	7.x	Session cache, semantic cache, rate limiting
LLM — Runtime (Bibliotecario)	Gemini 2.0 Flash via Vertex AI	Feb 2026	77% interazioni, \$0.10/\$0.40 per 1M tok
LLM — Runtime (Ass. Senior)	Gemini 3 Flash via Vertex AI	Feb 2026	15% interazioni, \$0.50/\$3.00 per 1M tok
LLM — Runtime (Professore)	Gemini 3 Pro via Vertex AI	Feb 2026	8% interazioni, \$2.00/\$12.00 per 1M tok
LLM — Fabbrica (batch)	Gemini 2.5 Pro via Batch API	Feb 2026	Pre-generazione artefatti, -50% con Batch API
Embeddings	text-embedding-004	Via Vertex AI	768 dim, per chunking e semantic search
Trascrizione Audio/Video	Google Chirp 3 (STT V2)	Via Vertex AI	Trascrizione lezioni per Lecture Weaver
Auth	Firebase Authentication	Latest	SSO universitario SAML/OIDC
Object Storage	Google Cloud Storage	Standard	File raw, artefatti JSON, media
Task Queue	Google Pub/Sub o Celery+Redis		Task asincroni (ingestion, batch gen)
Frontend	React 18 + TypeScript + Tailwind	Basato su Rev 5.0	Evoluzione del prototipo esistente
Container / Deploy	Docker + Cloud Run		Dev locale; produzione su GCP
Monitoring	Cloud Monitoring + Logging		Metriche, alerting, tracing

6. ARCHITETTURA: 5 Sottosistemi + 4 Strati OIOD

6.1 | 5 Sottosistemi MMS

Ogni richiesta dello studente attraversa una pipeline a 5 sottosistemi:

Utente → CORTEX (Intent) → SYNPASE (Knowledge) + CALCULATOR (Logic) → LLM (Format)
 → SENTINEL (Verify) → Risposta

Sottosistema	Funzione	Tecnologia	Cartella Codice
SENSORIUM	Ingestion: upload → extraction → chunking → embedding → indexing	pdfplumber, python-docx, Chirp 3, text-embedding-004	src/subsystems/sensorium/
SYNAPSE	Knowledge Graph: grafo concetti, relazioni prerequisito, query multi-hop	Neo4j 5.x, Cypher queries, concept mapper	src/subsystems/synapse/
CALCULATOR	Kernel Assiomatico: rule engine, gap analysis, scoring, time estimation	Rule Engine JSON/YAML, NumPy, schema validation	src/subsystems/calculator/
CORTEX	Intent Engine + LLM Gateway: intent detection, routing SOCRATES, prompt assembly, streaming	Gemini Flash (intent), routing a 4 strati, SSE	src/subsystems/cortex/
SENTINEL	Compliance: genome check pre/post output, anti-ghostwriting, Genesis Certificate, XAI Logs	SHA-256, RSA, style analysis, audit trail	src/subsystems/sentinel/

6.2 | 4 Strati OIOD Rev 2.0

L'architettura Offline Intelligence, Online Delivery è il cuore economico di QWIKKEN. Trasforma l'80%+ del traffico in lookup di artefatti pre-generati a costo quasi zero.

Strato	Modello AI	% Traffico	Cosa Fa	File Chiave
1. Fabbrica (batch)	Gemini 2.5 Pro (Batch API, -50%)	0% runtime	Pre-genera 36.000 artefatti: spiegazioni L1-L4, quiz, FAQ, alberi socratici	src/services/artifact_factory.py
2. Bibliotecario	Gemini 2.0 Flash (\$0.10/\$0.40)	77%	Cerca, seleziona, riformula artefatti pre-generati. Context caching -90%	src/subsystems/cortex/bibliotecario.py
3. Assistente Senior	Gemini 3 Flash (\$0.50/\$3.00)	15%	Escalation media: domande fuori FAQ, riformulazione avanzata, SOCRATES livello 2-3	src/subsystems/cortex/assistente_senior.py
4. Professore	Gemini 3 Pro (\$2.00/\$12.00)	8%	Escalation alta: SOCRATES profondo, cross-concept, generazione esami	src/subsystems/cortex/professore.py

Ogni escalation al Professore che produce una risposta di alta qualità viene promossa ad artefatto dal Learning Loop. Così il sistema migliora autonomamente e le escalation calano nel tempo.

7. ROADMAP DI SVILUPPO: 16 Settimane, 8 Fasi

Sett.	Fase	Sottosistemi	Deliverable Chiave	Tool AI
1-2	FASE 0: Setup + Onboarding Corsi	Infra + SENSORIUM	Repo, docker-compose (PG+pgvector, Redis, Neo4j), schema DB multi-tenant, CLAUDE.md, Firebase Auth, onboarding 6 syllabus	Claude Code (scaffold, DDL, migration)
3-4	FASE 1: La Fabbrica	SENSORIUM + Fabbrica	Pipeline batch: genera 36.000 artefatti per 6 corsi. QA 10%. Docente valida 1 corso. Storage artefatti in pgvector + JSON	Claude Code (batch script), Cowork (QA batch)
5-6	FASE 2: Bibliotecario + Routing	CORTEX + CALCULATOR	API routing 4-strati funzionante. Intent classification. Bibliotecario serve artefatti. Context caching attivo. Test 50 beta	Claude Code (routing, endpoint, test)
7-8	FASE 3: Ass. Senior + Socrates	CORTEX + SYNAPSE	Escalation Gemini 3 Flash. SOCRATES MODE con alberi + reasoning adattivo. Micro-verifiche integrate nel flusso TUTOR	Claude Code (socrates.py, queries Cypher)
9-10	FASE 4: Soft Launch 500	Tutti	Apertura a 500 studenti. Monitoring real-time. Dashboard metriche base. Iterazione su prompt e artefatti	Cowork (dashboard config, analisi feedback)
11-12	FASE 5: ARENA + Scale 1.200	CALCULATOR + CORTEX	Simulazione Esame attiva. Scoring engine. Learning Loop operativo. Scale a 1.200 studenti	Claude Code (scoring, quiz engine)
13-14	FASE 6: Full Scale 2.000	Tutti + SENTINEL	Tutti i 2.000 studenti. Ottimizzazione caching su dati reali. XAI Logs per compliance. Anti-ghostwriting base	Cowork (analisi costi, report compliance)
15-16	FASE 7: Valutazione + Report	N/A	Metriche finali. Report per ateneo. Proposta commerciale. Demo per CEO/CdA	Cowork (genera report, slide, analisi dati)

8. SPRINT DETTAGLIATI: Cosa Dire a Claude Code

Questo capitolo fornisce i prompt precisi da dare a Claude Code per ogni sprint. Non "costruisci QWIKKEN" ma task specifici e delimitati.

Sprint 1-2: Fondamenta + Ingestion (Settimane 1-2)

Obiettivo: repository funzionante con database, auth, multi-tenant e upload base.

Task #	Prompt per Claude Code	Output Atteso
1.1	"Crea docker-compose.yml con PostgreSQL 16 + pgvector, Redis 7, Neo4j 5. Configura .env.example con tutte le variabili."	docker-compose.yml, .env.example
1.2	"Crea i modelli SQLAlchemy in src/models/: TenantBase con tenant_id obbligatorio, User con CognitiveLevel enum L1-L4, Course, Enrollment con mastery tracking, GenomeConfig con 5 livelli."	6 file in src/models/
1.3	"Crea src/core/database.py con async engine e session factory. Aggiungi middleware tenant injection in src/core/middleware.py."	database.py, middleware.py
1.4	"Configura Alembic per async e genera la prima migration con tutte le tabelle."	alembic/, versions/001_initial.py
1.5	"Crea src/core/security.py con verifica Firebase token e FastAPI dependency per auth."	security.py
1.6	"Crea SENSORIUM pipeline base in src/subsystems/sensorium/: upload handler per PDF/DOCX, chunking a 512 token con overlap 50, embedding con text-embedding-004 via Vertex AI, storage in pgvector."	pipeline.py, extractors.py, chunker.py, embedder.py
1.7	"Crea test fixtures in tests/conftest.py: test tenant, test user, mock LLM. Scrivi test per il chunker e per il middleware tenant."	conftest.py, test_chunker.py, test_middleware.py

Sprint 3-4: La Fabbrica (Settimane 3-4)

Obiettivo: generare 36.000 artefatti didattici in batch e renderli ricercabili.

Task #	Prompt per Claude Code	Output Atteso
3.1	"Crea src/services/artifact_factory.py: pipeline batch che prende un corso, i suoi chunk/concetti, e genera artefatti (spiegazioni L1-L4, analogie, esempi numerici, micro-verifiche, FAQ, alberi socratici). Usa Gemini 2.5 Pro via Batch API. Output in JSON strutturato."	artifact_factory.py con schema JSON artefatti
3.2	"Crea i template prompt per ogni tipo di artefatto: spiegazione multi-livello, analogia, esempio numerico, quiz a 4 opzioni con feedback per distrattore, FAQ strutturata, albero socratico a 3-4 livelli. Parametrizzali per corso/concetto/livello."	prompt_templates/ con 8+ template
3.3	"Crea un CLI command (click o typer) per lanciare la Fabbrica su un corso: python -m src.cli.factory --course-id UUID --dry-run. Con progress bar e logging."	src/cli/factory.py
3.4	"Crea tabella artifacts in PostgreSQL e modello SQLAlchemy: id, tenant_id, course_id, concept_id, artifact_type (enum), content (JSONB), level (L1-L4), embedding, quality_score, validated_by, created_at."	models/artifact.py, migration

Sprint 5-6: Bibliotecario + Routing (Settimane 5-6)

Task #	Prompt per Claude Code	Output Atteso
5.1	"Crea src/subsystems/cortex/intent_router.py: usa Gemini 2.0 Flash per classificare l'intent dello studente in 15 categorie (EXPLAIN_CONCEPT, QUIZ_ME, DEEP_DIVE, EXAM_SIMULATE...). Estrai entità: corso, concetto, livello."	intent_router.py con 15 intent
5.2	"Crea src/subsystems/cortex/bibliotecario.py: cerca artefatti pre-generati in pgvector (semantic search). Se trova match >0.85, riformula con Flash e serve. Se no, escalation all'Assistente Senior."	bibliotecario.py con search + riformulazione
5.3	"Crea src/subsystems/cortex/assistente_senior.py: gestisce escalation medie con Gemini 3 Flash. Include context dal Genoma + materiali corso + storico studente."	assistente_senior.py
5.4	"Crea src/subsystems/cortex/professore.py: escalation alta con Gemini 3 Pro + context caching. Learning Loop: se la risposta ha rating >4/5, crea un nuovo artefatto."	professore.py con learning loop
5.5	"Crea l'endpoint SSE /api/v1/chat con streaming. Il flusso: intent_router → bibliotecario (o escalation) → sentinel check → stream response. Includi request_id e XAI metadata in ogni chunk."	api/v1/chat.py con SSE streaming

Sprint 7-8: SOCRATES + SYNAPSE (Settimane 7-8)

Task #	Prompt per Claude Code	Output Atteso
7.1	"Crea src/subsystems/cortex/socrates.py con due modalità: TUTOR MODE (spiega + micro-verifica) e SOCRATES MODE (domande progressive da albero pre-generato + branching adattivo quando l'albero si esaurisce)."	socrates.py bipartito
7.2	"Implementa SYNAPSE in src/subsystems/synapse/: setup Neo4j driver, schema nodi (Concept, Document, Course, User, Assessment), query Cypher per prerequisiti, gap analysis, concept linking."	graph.py, queries.py, concept_mapper.py
7.3	"Crea il genome checker in src/subsystems/sentinel/genome_checker.py: prima dell'output, verifica che la risposta rispetti L1 (mai inventare fonti), L2 (solo contenuti del corso), L3 (livello dello studente)."	genome_checker.py
7.4	"Integra micro-verifiche nel flusso TUTOR: dopo ogni spiegazione, proponi un quiz a 4 opzioni pre-generato dalla Fabbrica. Se lo studente sbaglia, servi il feedback specifico per il distrattore scelto."	Logica in socrates.py + quiz_engine.py

Dalla settimana 9 in poi il sistema è in produzione con studenti reali. Gli sprint successivi sono di scaling, ottimizzazione e moduli aggiuntivi. Il lavoro pesante di Claude Code è concentrato nelle settimane 1-8.

9. STRUTTURA DEL PROGETTO

La struttura completa del repository, già predisposta nello scaffold iniziale:

```

qwikken/
├── CLAUDE.md                      # Cervello per Claude Code
├── docker-compose.yml               # PG+pgvector, Redis, Neo4j
├── Dockerfile
├── pyproject.toml                  # Deps Python (40+)
├── alembic.ini + alembic/          # Migration engine
└── src/
    ├── main.py                     # FastAPI entry point
    ├── config.py                  # Pydantic settings
    ├── core/                       # database, redis, neo4j, security, middleware
    ├── models/                     # SQLAlchemy: base, user, tenant, course, genome,
        artifact, audit
    ├── schemas/                   # Pydantic: user, course, chat, genome, assessment
    ├── api/v1/                     # Routers: auth, chat, courses, planner, arena,
        genome, admin
    └── subsystems/
        ├── sensorium/             # pipeline, extractors, chunker, embedder,
            transcriber
        ├── synapse/                # graph, concept_mapper, queries, prerequisite
        ├── calculator/             # rule_engine, gap_analyzer, planner_engine,
            scoring
        ├── cortex/                 # intent_router, bibliotecario, assistente_senior,
        ├── sentinel/                # professore, socrates, prompt_builder, streaming
            xai_logger
        └── services/               # onboarding, tenant_provisioning,
            artifact_factory
└── frontend/                      # React/TS (evoluzione Rev 5.0)
└── tests/                         # unit/, integration/, e2e/

```

10. CONSIGLI OPERATIVI PER CLAUDE CODE

10.1 Le 7 Regole d'Oro

#	Regola	Perché	Esempio
1	Un task alla volta, ben definito	Claude Code lavora meglio con scope chiaro. Non dire "costruisci il backend"	"Crea lo schema PostgreSQL per la tabella artifacts con indice pgvector"
2	Il CLAUDE.md è il tuo asset #1	Aggiornalo dopo ogni sprint. Claude Code lo rilegge ogni sessione	Aggiungi: "Sprint 3 completato: Fabbrica genera artefatti per 1 corso"
3	Chiedi sempre i test	Senza test, il codice non è verificato. Claude Code li scrive bene	"Crea l'endpoint X con test pytest che copra i casi Y e Z"
4	Iterazione incrementale	Fai funzionare ogni pezzo prima di aggiungere il successivo	Prima: SENSORIUM che ingesta 1 PDF. Poi: Fabbrica. Poi: Bibliotecario
5	Branch per feature, merge per sprint	Mantieni il repo pulito. Claude Code gestisce bene il contesto	feature/sprint-03-fabbrica → squash merge → main
6	Context window management	Se la sessione è lunga, il contesto si satura. Chiudi e riapri	Dopo 40+ messaggi, nuova sessione: Claude Code rilegge CLAUDE.md fresco
7	Usa /cost per monitorare	Il comando /cost mostra quanto stai consumando nella sessione	Se /cost supera \$5 in una sessione, valuta se il task è troppo ampio

10.2 Anti-Pattern da Evitare

Anti-Pattern	Problema	Soluzione
"Costruisci QWIKKEN"	Task troppo vago, output incoerente	Decomponilo in task da 30-60 minuti
Non aggiornare CLAUDE.md	Claude Code non sa cosa è stato già costruito	Aggiorna alla fine di ogni giornata di lavoro
Saltare i test	Bug scoperti troppo tardi, regressioni	Chiedi test insieme al codice, sempre
Sessioni infinite (100+ messaggi)	Context overflow, degradazione qualità	Chiudi e riapri ogni 30-40 messaggi
Hardcodare valori	Non funziona in ambienti diversi (dev/staging/prod)	Usa config.py + .env per tutto
Ignorare gli errori di Claude Code	A volte sbaglia: import errati, API deprecate	Rivedi sempre il codice generato. "Trust but verify"

11. BUDGET COMPLETO DI SVILUPPO

Voce	Costo Una Tantum	Costo Mensile	Costo 4 Mesi	Note
Claude Code + Cowork (3 Max + 1 Pro)	—	€295	€1.184	Tool AI per il team di sviluppo
GCP Sviluppo (Cloud Run, SQL, Redis)	—	€80	€320	Ambiente di dev/staging
GCP Produzione (pilot 2.000 studenti)	—	€150	€600	Dalla settimana 9 in poi (~2.5 mesi)
Vertex AI — LLM Inference Runtime	—	€355	€1.420	Bibliotecario + Ass. Senior + Professore
Vertex AI — Fabbrica (batch, una tantum)	€891	—	€891	36.000 artefatti per 6 corsi
Dominio, vari, contingency	€100	—	€100	DNS, certificati, imprevisti
TOTALE COSTI VIVI			€4.515	Budget tecnologico puro

Voce	Costo 4 Mesi	Note
Costi vivi (tech)	€4.515	Infra + LLM + tooling AI
Team (5 persone, 3.35 FTE, già in organico)	€74.588	RAL da Business Plan + 35% contributi
Contingency 10%	€7.910	Buffer imprevisti
TOTALE FULLY LOADED	€87.013	Budget completo del pilot

Il costo incrementale reale per lanciare lo sviluppo e il pilot è ~€4.500 in costi vivi. Il team è già in organico. Per un investitore seed: il pilot da 2.000 studenti costa meno del 2% di un round seed da €300K.

12. CHECKLIST PRIMO GIORNO

Cosa fare concretamente per partire oggi.

#	Azione	Tempo Stimato	Comando / Note
1	Installare Claude Code	5 min	npm install -g @anthropic-ai/claude-code (Node.js 18+ richiesto)
2	Attivare piano Claude Max 5x (\$100/mese)	5 min	claude.ai/settings → upgrade. Include Cowork + Claude Code full
3	Creare repository Git	5 min	git init qwikken && cd qwikken
4	Copiare CLAUDE.md (18KB) nella root	1 min	Il file è già pronto — generato nelle sessioni precedenti
5	Copiare lo scaffold completo (35 file)	1 min	Lo scaffold è già pronto — estrarre il tar.gz generato
6	Lanciare docker-compose up -d	2 min	Avvia PostgreSQL+pgvector, Redis, Neo4j
7	Copiare .env.example → .env e configurare	5 min	Database URL, GCP project, Firebase, modelli Gemini
8	Lanciare Claude Code: claude	1 min	Dalla root del progetto. Legge CLAUDE.md automaticamente
9	Primo task: “Crea database.py con async engine + session factory e genera la migration iniziale”	20 min	Claude Code genera il codice, tu lo rivedi
10	Eseguire pytest per verificare	2 min	uv run pytest tests/ -v

Lo sviluppo può partire adesso. Il CLAUDE.md è pronto. Lo scaffold è pronto. I prezzi Google sono confermati. L'architettura OIOD Rev 2.0 è progettata. Claude Code + Cowork riducono lo sforzo del 67%. Il costo del primo mese di sviluppo è ~€370 (1 licenza Max) + €80 (infra dev) = **€450**.