15 Progetti Full-Stack per la Padronanza del Backend Python

Piano di Sviluppo Architetturale e Ingegneristico

20 febbraio 2026

Sommario

Questo documento descrive 15 progetti full-stack, scalando da applicazioni standard fino a sistemi distribuiti ad altissime prestazioni. L'obiettivo è fornire un contesto reale per applicare l'intero spettro dell'ingegneria backend in Python: dall'ottimizzazione dell'Event Loop e del Garbage Collector, fino alle architetture a microservizi per simulazioni matematiche e real-time bidding. I progetti sono ideati come casi studio di livello enterprise per consulenze informatiche e portfolio.

Indice

1 Fase 1: Fondamenta e Applicativi Gestionali **2**
- 1.1 Progetto 1: Sistema di Tracciamento Flotte Real-time (Logistica) . . . 2
- 1.2 Progetto 2: Ottimizzatore di Rotte (Logistica Matematica) . . . 2
- 1.3 Progetto 3: Piattaforma KDS - Kitchen Display System (Ristorazione) . . . 2
- 1.4 Progetto 4: Gestione Inventario e Fornitori (Retail) . . . 2
- 1.5 Progetto 5: Aggregatore di Dati Finanziari (Finanza) . . . 3

2 Fase 2: Integrazioni, IoT e Strumenti Analitici **4**
- 2.1 Progetto 6: Portale di Videoconsulti (Sanità) . . . 4
- 2.2 Progetto 7: Motore di Raccomandazione Prodotti (E-commerce) . . . 4
- 2.3 Progetto 8: Hub di Gestione Sensori Domestici (IoT) . . . 4
- 2.4 Progetto 9: Piattaforma E-learning Aziendale con Code Evaluation (HR) . . . 4
- 2.5 Progetto 10: Crawler Analitico Real Estate (Immobiliare) . . . 5

3 Fase 3: Progetti Enterprise Avanzati (High-Performance & Architettura) **6**
- 3.1 Progetto 11: Motore di Simulazione Monte Carlo Distribuito (Ingegneria) . . . 6
- 3.2 Progetto 12: Pipeline RAG (Retrieval-Augmented Generation) con Memory Management Custom (AI/Consulenza) . . . 6
- 3.3 Progetto 13: Piattaforma di Real-Time Bidding (Aste ad Alta Frequenza) . . . 6
- 3.4 Progetto 14: API Gateway Custom con Rate Limiting Distribuito (Sicurezza) . . . 7
- 3.5 Progetto 15: Orchestratore ETL Serverless per Data Warehouse (Data Engineering) 7

1 Fase 1: Fondamenta e Applicativi Gestionali

1.1 Progetto 1: Sistema di Tracciamento Flotte Real-time (Logistica)

Descrizione: Piattaforma per monitorare la posizione GPS dei veicoli aziendali in tempo reale.

- **Frontend:** React con librerie di mapping (Leaflet/Mapbox).
- **Syllabus coperto:**
  - **Framework FastAPI & WebSockets:** Streaming delle coordinate in tempo reale senza polling.
  - **AsyncIO e Event Loop:** Gestione asincrona delle migliaia di connessioni dei dispositivi IoT.
  - **Containerizzazione:** Creazione di `docker-compose.yml` per lanciare db, backend e frontend.

1.2 Progetto 2: Ottimizzatore di Rotte (Logistica Matematica)

Descrizione: Sistema per calcolare i percorsi di consegna ottimali (Vehicle Routing Problem).

- **Frontend:** React (Dashboard operativa).
- **Syllabus coperto:**
  - **Django REST Framework (DRF):** Esposizione robusta delle API.
  - **Code di Processi (Celery e Redis):** Calcolo asincrono in background per l'algoritmo oneroso.
  - **Concorrenza (Multiprocessing):** Parallelizzazione del calcolo euristico su più core della CPU.

1.3 Progetto 3: Piattaforma KDS - Kitchen Display System (Ristorazione)

Descrizione: Sistema che sincronizza in tempo reale le comande tra sala e cucina.

- **Frontend:** React (App tablet e monitor).
- **Syllabus coperto:**
  - **Validazione Dati con Pydantic:** Modellazione rigorosa degli ordini.
  - **Dependency Injection in FastAPI:** Iniezione pulita del database nelle route.
  - **Server ASGI (Uvicorn):** Configurazione per reggere i WebSockets.

1.4 Progetto 4: Gestione Inventario e Fornitori (Retail)

Descrizione: Piattaforma per l'automazione dei riordini, importazione di listini e gestione del magazzino.

- **Frontend:** React e Django Templates per le viste back-office.
- **Syllabus coperto:**
  - **Framework Django (MVT):** Uso del potente admin panel e dell'ORM per modelli relazionali complessi.

    - **Gestione File e Pathlib:** Upload e validazione sicura dei percorsi per file CSV dei listini.
    - **Generatori e Iteratori:** Parsing di file enormi riga per riga senza saturare la RAM.
    - **Decoratori e Context Managers:** Creazione di decoratori per il controllo degli accessi basato su ruoli.

1.5 Progetto 5: Aggregatore di Dati Finanziari (Finanza)

Descrizione: Piattaforma che aggrega prezzi da decine di exchange e normalizza i dati.

- **Frontend:** React con librerie di charting avanzate.
- **Syllabus coperto:**
    - **Typing Avanzato:** Uso di `mypy`, `Generics` e `Protocols` per tolleranza zero agli errori.
    - **Concorrenza (Threading):** Esecuzione parallela di richieste I/O-bound verso API esterne legacy.
    - **Ambienti Virtuali:** Setup deterministico con `Poetry`.

2 Fase 2: Integrazioni, IoT e Strumenti Analitici

2.1 Progetto 6: Portale di Videoconsulti (Sanità)

Descrizione: Portale per prenotare visite e avviare videochiamate tramite WebRTC.

- **Frontend:** React.
- **Syllabus coperto:**
  - **Architettura a Microservizi:** Booking Service (Django) separato dal Signaling Service (FastAPI).
  - **Pipeline CI/CD (GitHub Actions):** Linting (`ruff`), testing (`pytest`) e build automatizzato.

2.2 Progetto 7: Motore di Raccomandazione Prodotti (E-commerce)

Descrizione: Backend che analizza i carrelli e suggerisce correlazioni tramite similarità.

- **Frontend:** React (Componenti e widget embeddabili).
- **Syllabus coperto:**
  - **Data Model Python:** Implementazione dei "dunder methods" (`__eq__`, `__lt__`, `__hash__`).
  - **Programmazione Funzionale:** Uso intensivo di `map`, `filter`, `reduce` e `itertools`.

2.3 Progetto 8: Hub di Gestione Sensori Domestici (IoT)

Descrizione: Server di edge computing per raccogliere telemetrie e attuare comandi hardware.

- **Frontend:** React (PWA per smartphone).
- **Syllabus coperto:**
  - **Context Managers (`with`):** Gestione sicura delle connessioni (es. MQTT) verso l'hardware.
  - **Testing con Pytest:** Integration testing con uso avanzato di fixture e mock per simulare l'hardware.

2.4 Progetto 9: Piattaforma E-learning Aziendale con Code Evaluation (HR)

Descrizione: Costruite un portale interno per l'azienda. Voi proponete gli esercizi di Python, voi ricevete il codice dai vostri candidati, e lo valutate in totale sicurezza lato server.

- **Frontend:** React (Monaco Editor integrato).
- **Syllabus coperto:**
  - **Interprete Python e Gestione Memoria:** Analisi dell'AST per sicurezza e limitazione del Garbage Collector.
  - **Server WSGI (Gunicorn):** Tuning dei worker per isolare i processi in esecuzione.

2.5 Progetto 10: Crawler Analitico Real Estate (Immobiliare)

Descrizione: Scraping giornaliero di annunci, aggregazione dei prezzi e calcolo di heatmap.

- **Frontend:** React (Dashboard con mappe termiche interattive).
- **Syllabus coperto:**
  - **Generatori e Iteratori:** Yielding di flussi HTTP enormi per limitare il consumo di RAM.
  - **Pipeline CI/CD:** Schedulazione Cron job serverless su GitHub Actions.

3 Fase 3: Progetti Enterprise Avanzati (High-Performance & Architettura)

3.1 Progetto 11: Motore di Simulazione Monte Carlo Distribuito (Ingegneria)

Descrizione: Una piattaforma per simulazioni stocastiche (es. pricing di derivati, o analisi di stress strutturale) che richiede enorme potenza di calcolo. Perfetto per applicare le conoscenze matematico-ingegneristiche in un ambiente distribuito.

- **Frontend:** React (Pannello di configurazione parametri e visualizzazione grafici 3D/2D dei risultati).
- **Syllabus coperto:**
  - **Architettura a Microservizi & Celery:** Il master node (FastAPI) accetta la configurazione della simulazione e la divide in migliaia di sub-task inviati a Redis.
  - **Concorrenza Estrema (Multiprocessing + AsyncIO):** I worker Celery utilizzano il multiprocessing puro per saturare le CPU dei server fisici, mentre restituiscono progressi parziali al client tramite WebSockets (AsyncIO).

3.2 Progetto 12: Pipeline RAG (Retrieval-Augmented Generation) con Memory Management Custom (AI/Consulenza)

Descrizione: Un sistema di knowledge management aziendale che ingerisce PDF e documenti tecnici immensi (centinaia di GB), li vettorializza e permette a un LLM di rispondere a domande specifiche.

- **Frontend:** React (Interfaccia Chatbot documentale).
- **Syllabus coperto:**
  - **Gestione della Memoria e Generatori:** Ottimizzazione vitale: scrivere script Python che leggono, chunkizzano e inviano al Vector Database i documenti usando generatori puri, disabilitando/gestendo manualmente il Garbage Collector (`gc`) per evitare picchi di RAM mortali durante l'ingestion dei tensori.
  - **Typing e Pydantic:** Validazione severa delle strutture dati vettoriali in ingresso e in uscita dai modelli AI.

3.3 Progetto 13: Piattaforma di Real-Time Bidding (Aste ad Alta Frequenza)

Descrizione: Un sistema per aste online stile eBay ma per transazioni ad altissima frequenza (B2B o AdTech). Le puntate (bid) devono essere processate in meno di 10 millisecondi.

- **Frontend:** React (Schermata d'asta con timer ultra-precisi).
- **Syllabus coperto:**
  - **Event Loop spremuto al massimo:** Utilizzo di librerie ultra-veloci (es. `uvloop`) al posto dell'Event Loop standard di Python.
  - **SQLAlchemy Core:** Abbandono dell'ORM tradizionale per questo specifico microservizio in favore di SQLAlchemy Core, scrivendo query raw compilate per azzerare l'overhead dell'ORM durante l'inserimento massivo delle puntate.

3.4 Progetto 14: API Gateway Custom con Rate Limiting Distribuito (Sicurezza)

Descrizione: Invece di usare un servizio esistente (come Kong o Nginx), costruire da zero un API Gateway in Python che fa da scudo per le altre vostre app. Gestisce autenticazione, blocco degli abusi (rate limiting) e routing.

- **Frontend:** Nessuno (Solo backend e infrastruttura).
- **Syllabus coperto:**
  - **Decoratori, Middleware e Server ASGI:** Scrittura di middleware ASGI di basso livello (intercettando direttamente i dizionari `scope`, `receive`, `send`) per iniettare logica prima che la richiesta tocchi FastAPI o Django.
  - **Redis e Concorrenza:** Implementazione dell'algoritmo "Token Bucket" su Redis usando script Lua per garantire rate limiting distribuito e thread-safe tra più server.

3.5 Progetto 15: Orchestratore ETL Serverless per Data Warehouse (Data Engineering)

Descrizione: Uno strumento per aziende in cui gli utenti (es. analisti finanziari) possono disegnare tramite UI un flusso di estrazione dati, pulizia e caricamento (ETL). Il backend compila questa UI in script Python e li esegue.

- **Frontend:** React (Canvas interattivo drag-and-drop stile React Flow).
- **Syllabus coperto:**
  - **Interprete Python e Data Model:** Costruzione dinamica del codice Python a runtime (metaprogrammazione) in base ai nodi connessi nel frontend.
  - **Django REST Framework & CI/CD:** DRF salva lo schema dell'ETL nel database; dopodiché, tramite le API di GitHub, triggera una `GitHub Action` (CI/CD) che esegue lo script isolato in un container effimero, garantendo sicurezza totale e scalabilità.