DIGITAL MARKET SYSTEM

SISTEMA MERCATO DIGITALE

progetto innovazione digitalizzazione commercio su area pubblica

BRIEF COMPLETO – ECOSISTEMA DMS HUB / MIO / MULTI-AGENTE

Versione: 30.11.2025 – Andrea

RUOLO DELL'AGENTE A CUI STO SCRIVENDO

Sei il mio assistente tecnico principale (architetto + sviluppatore).

Devi:

- capire lo stato attuale di tutto il sistema (frontend, backend, agenti, automazioni);

- proporre piani chiari, a step, con file e funzioni da toccare;

- scrivere patch di codice pronte da copiare in GitHub (o da far applicare a Manus/MIO-GPT);

- NON inventare credenziali: sono già configurate su Vercel / Hetzner / Neon / Zapier.

---

1. ATTORI E AGENTI NEL SISTEMA

Persone:

- Andrea: product owner / architetto di fatto. Decide priorità e approva i piani.

Agenti:

- MIO (coordinatore GPT-5):

- vive nella Dashboard PA, chat principale in alto;

- è il "cervello" che coordina gli altri agenti (Manus, Abacus, Zapier, GPT Dev);

- parla SOLO con me nella chat sopra e con gli altri agenti dietro le quinte / mini-chat.

- GPT Dev (sviluppatore AI "pulito"):

- vecchia chat GPT collegata a GitHub tramite "ChatGPT Codex Connector";

- ruolo: leggere repo, proporre refactoring, scrivere codice pulito;

- obiettivo: affiancare Manus e, in futuro, diventare uno dei 4 sviluppatori nel multi-agente.

- Manus:

- agente esecutivo / refactoring automatico;

- legge i file, li modifica, fa commit/deploy;

- ha già fatto vari refactoring su DashboardPA.tsx, useAgentLogs, ecc., ma a volte incasina il flusso.

- Abacus:

- agente analitico / dati (per ora quasi solo concettuale, pochi endpoint reali).

- Zapier:

- agente automazione: collega GitHub ↔ Manus ↔ altri sistemi (Webhook, MCP "DMS-Zapier").

- DMS-Zapier (MCP):

- connettore custom che permette agli agenti GPT di richiamare Zapier / automazioni DMS.

---

2. INFRA / DOVE VIVE TUTTO

Repo principali GitHub (account: Chcndr):

- `dms-hub-app-new` → frontend Dashboard PA + MIO Agent.

- `mihub-backend-rest` → backend orchestratore MIO.

- `dms-system-blueprint` → documentazione, report, guide operative, architetture.

Hosting / servizi:

- Frontend: Vercel

- dominio: https://dms-hub-app-new.vercel.app (Dashboard: `/dashboard-pa`)

- ogni push su `master` fa un deploy.

- Backend orchestratore:

- inizialmente: server Hetzner (IP 157.90.29.66);

- ora esposto come: `https://orchestratore.mio-hub.me`

- endpoint principale: `/api/mihub/orchestrator`.

- Database:

- Neon PostgreSQL (es. istanza `ep-bold-silence-...`);

- tabella chiave: `agent_logs` (salva messaggi di tutti gli agenti).

- Zapier:

- usato per automazioni legate a GitHub e flussi DMS;

- connettore MCP "DMS-Zapier" visibile anche dentro GPT.

- GitHub Connector:

- "ChatGPT Codex Connector" installato sul mio account GitHub, con:

- permessi READ/WRITE su azioni, codice, issue, PR e workflow;

- accesso a TUTTI i repository dell'account.

IMPORTANTE CREDENZIALI (non scriverle mai in chiaro in chat):

- Vercel: env vars (es. `MIHUB_BACKEND_URL`, `DATABASE_URL`, API key varie).

- Hetzner backend: file `.env` locale sul server + PM2.

- Neon: stringa `DATABASE_URL` solo nei secret.

- Zapier / DMS-Zapier: token e hook configurati nel dashboard Zapier.

- GitHub PAT (se esiste ancora): collegato al connettore / vecchio GPT MIO.

Se ti chiedo di usare queste cose, GUIDAMI sui file / pannelli, ma non chiedermi password in chiaro.

---

3. FRONTEND – STRUTTURA CHAT MIO / MULTI-AGENTE

File chiave frontend (repo `dms-hub-app-new`):

- `client/src/pages/DashboardPA.tsx`

→ contiene:

- Chat MIO principale (sopra)

- Sezione "Chat Multi-Agente" con:

- vista singola (4 chat isolate)

- vista 4 agenti (read-only)

- `client/src/components/multi-agent/MultiAgentChatView.tsx`

→ griglia 2x2 con i 4 agenti (GPT Dev, Manus, Abacus, Zapier), solo lettura.

- `client/src/hooks/useAgentLogs.ts`

→ hook per leggere log da `/api/mio/agent-logs`.

- `client/src/lib/mioOrchestratorClient.ts`

→ funzioni di chiamata all'orchestrator:

- `sendMioMessage(...)`

- `sendAgentMessage(...)`

- `client/src/components/ChatWidget.tsx`

→ widget flottante in basso a destra (per ora disattivato / rotto).

- `vercel.json`

→ rewrite `/api/mihub/orchestrator` → backend MIO.

ARCHITETTURA CHAT (come DEVE essere):

1) Chat MIO Principale (sopra)

- Unica chat tra me (utente) e MIO coordinatore.

- Usa STATO LOCALE React (non useAgentLogs) per mostrare subito:

- messaggio utente;

- risposta di MIO.

- Flusso:

- utente scrive → handleSendMio → `sendMioMessage`:

- prima chiamata può NON avere `conversationId` → backend lo crea;

- backend risponde con `{ message, conversationId, ... }`;

- stato locale si aggiorna con:

- messaggio utente,

- risposta MIO,

- eventuale nuovo conversationId.

- Obiettivo: veloce, stabile, niente polling.

2) Vista 4 Agenti (sotto, griglia READ-ONLY)

- Mostra cosa succede tra MIO e i 4 agenti.

- Nessun input. Solo lettura.

- Usa `useAgentLogs` con:

- `conversationId = mioMainConversationId` (quello della chat principale);

- `agentName` = `gptdev`, `manus`, `abacus`, `zapier`.

- Vede solo log relativi alla stessa conversazione di MIO → tipo "scatola nera".

3) Vista Singola (sotto, 4 chat isolate)

- Tabs / pulsanti: **GPT Dev**, **Manus**, **Abacus**, **Zapier**.

- Ogni agente ha una chat COMPLETAMENTE SEPARATA da MIO:

- `gptdevConversationId` (persistito in localStorage `gptdev-single`);

- `manusConversationId` (`manus-single`);

- `abacusConversationId` (`abacus-single`);

- `zapierConversationId` (`zapier-single`).

- Flusso:

- utente scrive a GPT Dev (o Manus ecc.) → `handleSendGptdev` → `sendAgentMessage('gptdev', ...)`;

- orchestrator parla direttamente con quell'agente;

- i messaggi di questa chat NON devono toccare la conversazione MIO sopra.

- Qui GPT Dev e Manus sono **developer diretti**: posso chiedere refactoring, codice, piani, ecc.

4) Chat Widget Flottante (DMS AI Assistant)

- Obiettivo futuro:

- agganciato a MIO coordinatore,

- conversationId proprio (es. `mio-widget`),

- usa backend orchestrator via proxy Next/Vercel.

- Al momento:

- precedenti problemi: HTTP 404, 429, mancata persistenza;

- è meglio considerarlo DISABILITATO finché non lo sistemiamo.

---

4. BACKEND – LOG E ORCHESTRATOR

Backend orchestratore (`mihub-backend-rest`):

- endpoint principale: `POST /api/mihub/orchestrator`

- mode: `auto` (MIO decide se delegare a Manus/Abacus/Zapier)

- mode: `manual` con `targetAgent` = gptdev/manus/abacus/zapier (per vista singola)

- log:

- salva ogni messaggio (user + agente) in tabella `agent_logs` (Neon);

- campi chiave: `conversation_id`, `agent_name`, `role`, `content`, `created_at`.

- endpoint log:

- `GET /api/mio/agent-logs?conversation_id=...&agent_name=...`

- usato da `useAgentLogs`.

Da NON fare finché non lo dico:

- non toccare routing di orchestrator;

- non cambiare schema `agent_logs`;

- non cambiare la logica di salvataggio se non strettamente necessario.

---

5. SITUAZIONE ATTUALE – COSA VA / COSA NO

Cosa FUNZIONA (in generale):

- Chat MIO sopra:

- primo messaggio → MIO risponde correttamente;

- stato locale ok, UI chiara;

- endpoint orchestrator raggiungibile (URL corretto).

- Backend:

- salvataggio su `agent_logs` funziona;

- query `GET /api/mio/agent-logs` restituisce i messaggi giusti;

- persistenza `conversationId` lato client via `useConversationPersistence` ok.

- Vista 4 agenti:

- lettura log in read-only da `useAgentLogs` funziona (in varie fasi di test è stata l'unica sempre stabile).

Cosa È FRAGILE / DA SISTEMARE:

- Chat MIO:

- a volte il PRIMO messaggio funziona, il SECONDO dà HTTP 404 o errori vari;

- problemi di allineamento tra `conversationId` locale (tipo `conv_...`) e ciò che il backend si aspetta;

- differenze di payload: backend a volte risponde con `data.message` invece di `data.reply`.

- Vista singola:

- in alcuni refactoring precedenti Manus ha rotto gestioni di stato (es. state non definiti, handler doppi);

- oggi l'obiettivo è che sia semplice: UN handler per agente, UN conversationId per agente, uso corretto di `sendAgentMessage`.

- Rate limiting (HTTP 429):

- `useAgentLogs` fa polling ogni 5s per più agenti → molte richieste/minuto;

- backend potrebbe rispondere 429 dopo troppi test ravvicinati.

- Chat widget:

- ancora non affidabile (404, 429, problemi CORS / proxy);

- da considerare "fase 2".

---

6. COME STO LAVORANDO ADESSO

- In questo momento sto usando principalmente:

- MANUS (agente nella tua interfaccia) per:

- leggere i file sorgente,

- proporre fix,

- generare patch,

- a volte fare refactoring grossi (ma spesso si incarta).

- Questo GPT / MIO-GPT (sviluppatore) per:

- pensare l'architettura,

- scrivere report tecnici e piani chiari,

- preparare blocchi di codice "puliti" da applicare poi a mano o con Manus.

- Ho collegato:

- ChatGPT Codex Connector su GitHub (tutti i repo, read/write);

- DMS-Zapier connector (MCP) per automatizzare in futuro build/deploy/log.

OBIETTIVO:

- "RIESUMARE" il vecchio GPT sviluppatore collegato a GitHub per farlo lavorare DIRETTAMENTE sul codice:

- audit dei repo,

- proposte di refactoring,

- creazione di file nuovi / README / blueprint.

- Manus viene usato più come:

- code-reviewer + esecutore di patch + deploy su Vercel/Hetzner,

- non come architetto che rivoluziona tutto da solo.

In prospettiva:

- nel multi-agente:

- GPT Dev e Manus saranno i due "developer" principali,

- MIO sopra coordina e decide chi attivare,

- Abacus e Zapier restano più verticali su analisi e automazioni.

---

7. COSA CHIEDO A TE (AGENTE DI QUESTA CHAT)

Quando inizi a lavorare su questo progetto, voglio che tu:

1. Confermi di aver capito questo contesto.

2. Mi chieda SUBITO:

- quali repo vuoi che usiamo (di solito `dms-hub-app-new` + `mihub-backend-rest` + `dms-system-blueprint`);

- se lavoriamo in sola lettura (analisi) o anche con patch di codice da applicare.

3. Per ogni task importante:

- fai prima un AUDIT rapido dei file coinvolti (es. DashboardPA.tsx, mioOrchestratorClient.ts, useAgentLogs.ts);

- proponi un PIANO in punti, file per file;

- solo dopo il mio "OK" scrivi il codice (diff completo oppure file intero).

4. Nel codice:

- niente cambi architetturali giganteschi senza motivo;

- commenti chiari dove tocchi il flusso di MIO/coordinatore e dei multi-agenti;

- se tocchi qualcosa che riguarda `conversationId` o `agent_logs`, spiegami benissimo la logica.

Scopo finale:

- MIO sopra stabile, senza 404;

- vista 4 agenti che mostra bene cosa combinano GPT Dev / Manus / Abacus / Zapier;

- vista singola con 4 chat isolate usabili per lavorare direttamente con gli agenti sviluppatori;

- in una seconda fase, widget globale che porta MIO in tutte le pagine del DMS Hub.

FINE BRIEF

BRIEF SEZIONI APPLICAZIONE – DMS HUB / DASHBOARD PA

Versione: 30.11.2025 – Andrea

Scopo di questo documento

- Dare una mappa chiara di TUTTE le sezioni principali della Dashboard PA.

- Spiegare cosa fa ogni sezione (business) e come è collegata (tecnico).

- Descrivere la nuova macro-sezione "Gestione Imprese" con Qualificazioni, Wallet, KPI.

- Riassumere log, debug, endpoint e infrastruttura (Hetzner, Neon, Zapier, GitHub, ecc.).

---

0. VISIONE DI SISTEMA – COSA SERVE TUTTO

DMS Hub è la "console" per la Pubblica Amministrazione e per i gestori dei mercati.

Serve per:

- gestire mercati, posteggi, operatori/aziende;

- avere una mappa GIS di quello che succede sul territorio;

- tenere traccia di pagamenti, qualificazioni, KPI;

- usare un sistema multi-agente (MIO + GPT Dev + Manus + Abacus + Zapier) per:

- leggere i dati,

- proporre azioni,

- generare codice e automazioni,

- fare deployment delle modifiche.

Strati:

- UI / Dashboard PA (Vercel).

- API / orchestrator MIO (Hetzner, dominio orchestratore.mio-hub.me).

- Database (Neon PostgreSQL, tabella chiave `agent_logs` + tabelle dominio mercati/imprese).

- Automazioni (Zapier, DMS-Zapier, GitHub connector).

- Blueprint (repo di documentazione con guide operative, credenziali e architetture).

---

1. SHELL E NAVIGAZIONE DASHBOARD PA

Vista principale: `/dashboard-pa` (repo `dms-hub-app-new`).

La home della Dashboard PA ha:

- barra in alto con periodo, filtri, export report;

- griglia di “tile”/moduli, tipicamente:

- Gestione Mercati

- Gestione Negozi / Operatori

- HUB Operatore

- BUS HUB

- Core Map

- Sito pubblico

- DMS News

- DMS Carbon Credit / Sostenibilità

- TPAS / Sicurezza

- Debug

- Report

- MIO Agent

- Integrazioni

- Impostazioni, ecc.

Ogni tile apre un pannello specifico o una pagina figlia.

È la “plancia di comando” per la PA.

---

2. GESTIONE MERCATI / NEGOZI (STATO ATTUALE)

Componenti tipici nel client:

- `MercatiPanel` → gestione mercati e posteggi.

- `NegoziPanel` → gestione operatori/negozi/aziende legati ai mercati.

- Tabelle, filtri, dettagli schede.

Funzioni:

- elencare mercati attivi, date, località;

- gestire concessioni/posteggi (chi occupa cosa, quando);

- vedere anagrafiche di operatori collegati (Negozi/Imprese);

- integrare in futuro con Qualificazioni, Wallet e KPI.

Backend:

- endpoint REST per CRUD mercati/posteggi/operatori (prefisso `/api/mihub/...` o simili).

- i dettagli concreti (nomi tabelle, route precise) sono nel codice / blueprint,

l'importante è che DASHBOARD PA è il front per la PA, non per il singolo esercente.

---

3. MAPPA GIS – "CORE MAP / GIS VIEWER"

Sezione GIS (es. `GisViewer.tsx`):

- mappa interattiva del territorio (Grosseto, altri comuni).

- visualizza:

- mercati,

- posteggi,

- eventuali POI (negozi, servizi, logistica).

- usa un provider mappe (Leaflet, Mapbox o simili – dipende dal codice effettivo).

Funzioni attese:

- layer differenziati (mercati, traffico, sensori, ecc. – dove disponibili).

- click su un punto → dettaglio del mercato/posteggio.

- in prospettiva: overlay KPI (ad es. volumi vendita, affluenza, ecc. via Abacus).

Backend:

- endpoint tipo:

- `GET /api/mihub/mercati/geo`

- `GET /api/mihub/posteggi/geo`

- dinamica: la mappa pesca dalla stessa base dati dei pannelli mercati/negozi,

non da un database separato.

---

4. NUOVA MACRO-SEZIONE “GESTIONE IMPRESE”

Questa è la parte NUOVA da standardizzare e sviluppare bene.

Obiettivo:

- avere una vista unica per ogni impresa/operatore,

con collegati:

- dati anagrafici,

- qualificazioni / titoli,

- stato pagamenti (Wallet),

- KPI operativi,
- documenti e log di sistema legati a quell'impresa.

Struttura logica (macro-moduli):

4.1 Anagrafiche Imprese

- elenco imprese / operatori con:
  - dati base (nome, P.IVA, settore, contatti);
  - collegamento a mercati/posteggi;
  - stato attivo/sospeso/cessato.
- UI:
  - tabella + dettaglio a destra o pagina dedicata.
- backend (esempi):
  - `GET /api/mihub/imprese`
  - `GET /api/mihub/imprese/:id`
  - `POST /api/mihub/imprese` (crea/aggiorna).

4.2 Qualificazioni (licenze, DURC, concessioni, ISO…)

- "cartella tecnica" dell'impresa.
- per ogni impresa:
  - licenze attive,

- concessioni mercato/posteggio,
- scadenze (DURC, HACCP, certificazioni),
- eventuali allegati.

- UI:
- tab “Qualificazioni” dentro la scheda impresa.

- backend (esempi):
- `GET /api/mihub/imprese/:id/qualificazioni`
- `POST /api/mihub/imprese/:id/qualificazioni`.

4.3 Wallet (pagamenti, crediti/debiti)

- portafoglio digitale per tasse, canoni mercati, servizi.

- per ogni impresa:
- saldo,
- movimenti (addebiti, pagamenti, note di credito),
- integrazione futura con sistemi di pagamento.

- UI:
- tab “Wallet” nella scheda impresa,
- vista riepilogo “Wallet PA” per la tesoreria.

- backend (esempi):
- `GET /api/mihub/imprese/:id/wallet`
- `POST /api/mihub/imprese/:id/wallet/movimento`.

4.4 KPI (indicatori performance)

- strato di analytics (Abacus).

- per ogni impresa:

** - presenze a mercato,**

** - storico pagamenti / insoluti,**

** - partecipazione a iniziative,**

** - indicatori di rischio.**

- UI:

** - grafici, semafori, punteggi.**

- backend (esempi):

** - `GET /api/mihub/imprese/:id/kpi`**

** - `GET /api/mihub/kpi?periodo=...`.**

4.5 Integrazione con MIO / Multi-agente

- MIO sopra:

** - riceve richieste “alto livello” (es. “dimmi le imprese a rischio X”, “preparami un piano per regolarizzare questi operatori”);**

** - delega Abacus per analisi KPI,**

** - fora Zapier per trigger esterni (email, notifiche, automazioni).**

- Vista singola GPT Dev/Manus:

- usati per scrivere codice per questi moduli

(nuovi endpoint, pannelli React, script di migrazione DB).

- Log:

- ogni azione importante su Qualificazioni/Wallet/KPI loggata in `agent_logs` o in una tabella log di dominio.

---

5. SEZIONI “TESTI” / DOCUMENTAZIONE / NEWS

Nel DMS Hub ci sono varie sezioni “testuali” o informative:

- Sito Pubblico:

- vetrina pubblica per cittadini/operatori,

- potrebbe essere servito da un altro repo statico (es. `dms-site` o simili).

- DMS News:

- sezione per comunicazioni ufficiali, bandi, scadenze, notizie sui mercati.

- Documentazione:

- link o pannelli interni che rimandano al blueprint, manuali, guide,

- l’obiettivo è centralizzare qui anche la documentazione MIO/multi-agente.

Dal punto di vista tecnico:

- possono essere pannelli React che leggono da:

- file Markdown nel repo,

- API CMS,

- o semplice contenuto hardcoded da sistemare più avanti.

---

6. LOG, DEBUG, ENDPOINT – OSSERVABILITÀ

Questa è la parte “sala macchine”, fondamentale per capire cosa succede.

6.1 Log di Dominio / Applicativi

- Pannello “Log” nella Dashboard PA (es. `LogPanel.tsx`).

- Possibili fonti:

- log di operazioni mercati/imprese,

- log di sistema (errori, warning, job schedulati).

- Lato backend:

- API di lettura log (es. `/api/logs`, `/api/mihub/logs`),

- un servizio `apiLogsService.ts` che salva/legge record da DB (tipo tabella `api_logs`).

6.2 Log Multi-Agente (MIO / Manus / Abacus / Zapier / GPT Dev)

- Tabella `agent_logs` in Neon:

- `conversation_id`

- `agent_name`

- `role` (user/assistant/system)

- `content`

- `created_at`

- Endpoint:

- `POST /api/mio/agent-logs` (scrittura)

- `GET /api/mio/agent-logs?conversation_id=...&agent_name=...` (lettura).

- Frontend:

- `useAgentLogs` per leggere periodicamente questi log,

- MultiAgentChatView in READ-ONLY per vista 4 agenti,

- eventuali altre viste di ispezione.

6.3 Debug Orchestrator / API

- Sezione “Debug” nella dashboard (tile dedicato).

- Serve a:

  - testare endpoint specifici (API_TEST, TRPC, ecc.),

  - vedere errori HTTP degli orchestrator (404, 429, 500),

  - monitorare stato collegamenti a Guardian, Gemini, ecc.

- Backend:

  - `orchestrator.js` con logging dettagliato;

  - `api/github/logs.ts` per tracciare interazioni con GitHub;

  - `guardianRouter.ts` e affini per log di sicurezza (quando il servizio è attivo).

6.4 Endpoints principali da considerare

- Orchestrator:

  - `POST /api/mihub/orchestrator`

    - mode: `auto` (MIO decide e può coinvolgere i 4 agenti),

    - mode: `manual` + `targetAgent` (per vista singola).

- Logs agenti:

  - `GET/POST /api/mio/agent-logs`.

- API dominio:

  - mercati, posteggi, imprese, qualificazioni, wallet, kpi...

  - sempre con prefisso `/api/mihub/...`.

- Utility / debug:

- endpoint di health check,

- eventuali `API_TEST` / `TRPC` per compatibilità.

---

7. INFRASTRUTTURA E ACCESSI (ALTO LIVELLO)

Non metto qui password o token, ma solo la mappa:

- GitHub (account: Chcndr)

  - repo app: `dms-hub-app-new`, `mihub-backend-rest`, `dms-system-blueprint`, altri repo pubblici.

- connettore “ChatGPT Codex Connector” installato con:

  - permessi read/write su codice, issue, PR, workflow;

  - accesso a tutti i repository.

- Vercel:

- proj: `dms-hub-app-new`;

- URL: `https://dms-hub-app-new.vercel.app`;

- deploy automatico su push a `master`;

- env vars con URL backend, DB, ecc.

- Backend Hetzner:

- server Node/Express (o simile) con orchestrator;

- dominio: `https://orchestratore.mio-hub.me`;

- gestito con PM2;

- `.env` locale con secret (DB, API key LLM, ecc.).

- Neon PostgreSQL:

- DB gestito come servizio;

- tabelle dominio + `agent_logs` + `api_logs` (a seconda delle migrazioni).

- Zapier:

- automazioni per integrare GitHub, email, altri servizi;

- MCP “DMS-Zapier” per far parlare GPT con Zapier.

- Blueprint:

- repo: `dms-system-blueprint`;

- contiene:

- report tecnici,

- guide operative (Hetzner, Neon, Vercel),

- architetture mermaid,

- istruzioni per accedere alle console con le credenziali.

---

8. STRATEGIA ATTUALE DI LAVORO

- Uso MANUS come braccio operativo:

- gli faccio aprire file,

- gli do patch specifiche,

- lo faccio deployare (Vercel / Hetzner) quando il codice è pronto.

- Uso GPT Dev (MIO-GPT con GitHub connector) come sviluppatore "pulito":

- audit repo,

- piani di refactoring,

- file `.md` di documentazione,

- codice "pulito" da applicare.

- Uso MIO coordinatore (nella Dashboard) per:

- ragionare ad alto livello sui flussi,

- orchestrare il lavoro tra i 4 agenti (in prospettiva).

- Obiettivo:

- fissare in modo definitivo:

- chat MIO stabile,

- vista 4 agenti in lettura,

- vista singola 4 agenti sviluppatori,

- macro-sezione Gestione Imprese (Anagrafiche + Qualificazioni + Wallet + KPI),

- e poi portare MIO anche nel widget globale che segue Andrea in tutte le pagine.

FINE BRIEF SEZIONI

flowchart LR

%% ==========================

%% LIVELLO UTENTE / PA

%% ==========================

subgraph UI["Dashboard PA (Vercel)"]

A1[📋 Lista Imprese\n/gestione-imprese]

A2[🏷️ Qualificazioni\nTab nella scheda impresa]

A3[💰 Wallet\nPagamenti e movimenti]

A4[📈 KPI\nIndicatori e grafici]

A5[🧠 MIO Chat Principale\nGPT-5 Coordinatore]

A6[👨‍💻 GPT Dev – Chat Singola\nSviluppo codice]

A7[🛠️ Manus – Chat Singola\nRefactor & Deploy]

end

```
%% ========================
%% LIVELLO API / BACKEND
%% ========================
subgraph BE["Backend Hetzner\norchestratore.mio-hub.me"]
  direction LR

  subgraph API["API dominio /api/mihub/*"]
    B1[GET/POST /imprese\nanagrafiche]
    B2[GET/POST /imprese/:id/qualificazioni]
    B3[GET/POST /imprese/:id/wallet]
    B4[GET /imprese/:id/kpi]
  end

      subgraph ORCH["MIO Orchestrator\nPOST /api/mihub/orchestrator"]
    O1[MIO Coordinator\n(decide cosa fare)]
    O2[Router verso agenti\n(targetAgent / mode)]
  end

  subgraph LOGS["Servizio Log Agenti\n/api/mio/agent-logs"]
```

```
        L1[(agent_logs DB)]
    end
end

%% ==========================
%% LIVELLO AGENTI
%% ==========================
subgraph AGENTS["Agenti Specializzati"]
        G1[🤖 GPT Dev\nSviluppatore codice\n(legge/scrive su GitHub)]
    M1[🚢 Manus\nCode review + deploy\n(Hetzner/Vercel)]
    AB1[📊 Abacus\nAnalytics & KPI]
    Z1[⚡ Zapier\nAutomazioni esterne\n(email, notifiche, ecc.)]
end

%% ==========================
%% LIVELLO DATI
%% ==========================
subgraph DATA["Dati & Integrazioni"]
        D1[(Neon PostgreSQL\nTabelle imprese,\nqualificazioni, wallet, kpi)]
```

```
    D2[(agent_logs\nconversazioni agenti)]

    GH[(GitHub\nrepo dms-hub-app-new,\nmihub-backend-rest,
\nblueprint)]

    ZAP[(Zapier Flows\nGitHub ↔ Notifiche ↔ Altro)]

  end


  %% UI -> API dominio

  A1 -->|lista / dettagli imprese| B1

  A2 -->|legge/salva qualificazioni| B2

  A3 -->|legge/salva movimenti| B3

  A4 -->|legge KPI| B4


  %% API dominio -> DB

  B1 --- D1

  B2 --- D1

  B3 --- D1

  B4 --- D1


  %% UI -> MIO

  A5 -->|messaggi utente| ORCH

  ORCH -->|risposta aggregata| A5
```

```
%% MIO -> Agenti

O1 -->|task analisi KPI\n(es. imprese a rischio)| AB1

O1 -->|task automazione esterna| Z1

O1 -->|task sviluppo codice\nnuovi moduli imprese| G1

O1 -->|task deploy / refactor| M1


%% Agenti -> DB / GitHub / Zapier

AB1 -->|query KPI| D1

AB1 -->|scrive risultati\n(elaborazioni)| D1

G1 -->|legge/scrive codice| GH

M1 -->|deploy backend/frontend\nconfigurazione| GH

Z1 -->|esegue scenari\nnotifiche / sync| ZAP


%% Logging conversazioni agenti

ORCH -->|scrive log\n(user/assistant)| L1

G1 -->|scrive log| L1

M1 -->|scrive log| L1

AB1 -->|scrive log| L1

Z1 -->|scrive log| L1

L1 --- D2
```

%% Viste multi-agente in UI

A5 -->|mostra\nconversazione MIO| D2

A6 -->|chat singola GPT Dev| D2

A7 -->|chat singola Manus| D2

Uso dell’idea:

- Per GPT Dev / MIO vecchio: gli incolli prima il brief precedente, poi questo schema mermaid e gli dici “il tuo pezzo è questa parte qui: GPT Dev, Manus, Abacus, Zapier, GitHub, orchestrator”.
- Per Manus: gli fai vedere la mappa e gli dici esattamente quali blocchi deve toccare (es. “solo API dominio per Gestione Imprese + hook frontend, non toccare orchestrator né MIO principale”).
- Per chiunque entri dopo (sviluppatore umano): guarda questo schema + il brief, capisce subito dove si incastra la nuova sezione imprese dentro tutto il sistema.

Mappa visuale – Gestione Imprese ↔ MIO ↔ Abacus/Zapier

1. Flusso architetturale alto livello

```mermaid
flowchart LR
  subgraph PA["PA / Ufficio Commercio"]
    UI["Dashboard PA\nModulo Gestione Imprese"]
  end

  subgraph Backend["Backend MIO-hub (Hetzner)"]
    API["API REST\n/api/mihub/*"]
    MIO["MIO Orchestrator"]
    Abacus["Abacus\n(analytics / KPI)"]
    ZapierAgent["Zapier Agent\n(bridge verso Zapier)"]
  end

  subgraph DB["Neon PostgreSQL"]
    imprese[(imprese)]
    qualificazioni[(qualificazioni)]
    wallet[(wallet_movimenti)]
```

```
    kpi[(kpi_cache)]
    agentLogs[(agent_logs)]
  end

  subgraph Esterno["Servizi esterni"]
    Zapier["Zapier Flows"]
    Notifiche["Email / PEC / altri canali"]
  end

  %% UI ↔ backend dominio
  UI -->|CRUD imprese / qualifiche / wallet / KPI| API
  API --> imprese
  API --> qualificazioni
  API --> wallet
  API --> kpi

  %% Chat MIO
  UI <--> |POST /api/mihub/orchestrator\n(mode: auto)| MIO

  %% MIO ↔ Abacus / Zapier
  MIO --> |richiesta analisi| Abacus
```

```
    Abacus --> |query KPI| kpi

    MIO --> |task esterni| ZapierAgent
    ZapierAgent --> Zapier --> Notifiche

    %% Logging conversazioni agenti
    MIO <--> agentLogs
    Abacus <--> agentLogs
    ZapierAgent <--> agentLogs
```

```
sequenceDiagram
    actor PA as Ufficio PA
    participant UI as Gestione Imprese (UI)
    participant API as API /api/mihub/*
    participant DB as Neon DB
    participant MIO as MIO Orchestrator
    participant Abacus as Abacus (KPI)
    participant ZAgent as Zapier Agent
```

participant Zap as Zapier

PA->>UI: 1) Apri scheda impresa X

UI->>API: GET /imprese/X + /qualificazioni + /wallet + /kpi

API->>DB: query tabelle dominio

DB-->>API: dati impresa X

API-->>UI: anagrafica + qualifiche + wallet + KPI

Note over UI: la PA vede situazione completa dell'impresa

PA->>MIO: 2) "Analizza impresa X\n e proponi un piano"

UI->>MIO: POST /api/mihub/orchestrator\n(mode: auto, domanda PA)

MIO->>Abacus: 3) Calcola KPI / rischio per impresa X

Abacus->>DB: query storico KPI / pagamenti / presenze

DB-->>Abacus: risultati analitici

Abacus-->>MIO: KPI sintetici + alert

MIO->>ZAgent: 4) Prepara azioni esterne\n(es. avvisi, promemoria)

ZAgent->>Zap: trigger flow "Regolarizzazione impresa X"

Zap-->>ZAgent: OK (email/PEC/task creati)

MIO-->>UI: 5) Risposta coordinata\n(testuale + elenco azioni proposte)

Note over PA,MIO: la PA vede suggerimenti e cosa è stato automatizzato

MIO->>DB: 6) Scrive tutta la conversazione in agent_logs

Ok, rifaccio da zero come “documento di progetto” per la nuova chat.

Puoi incollarlo così com’è in un file .md tipo:

02_Mappa_Dashboard_PA_Sezioni_Dati_Agenti.md.

Mappa Funzionale Dashboard PA – Sezioni, Dati, Agenti

Sistema: DMS HUB – Dashboard PA

Frontend: dms-hub-app-new (Vercel)

Backend orchestratore: mihub-backend-rest (Hetzner – orchestratore.mio-hub.me)

DB principale: Neon PostgreSQL

Agenti: MIO (coordinatore), GPT Dev, Manus, Abacus, Zapier

1. Strato tecnico di base (contesto per tutti)

- Frontend (Vercel)
    - App React/Next: dms-hub-app-new
    - Usa:

        - chiamate REST al backend orchestratore (/api/mihub/*)
        - endpoint GET/POST /api/mio/agent-logs per log agenti (vista 4 agenti / singole)
    -
    - Chat MIO principale: ora funziona con stato locale, non più con polling.
-
- Backend (Hetzner)
    - Repo: mihub-backend-rest
    - Espone:
        - POST /api/mihub/orchestrator → MIO coordinatore + multi-agente
        - GET/POST /api/mio/agent-logs → lettura/scrittura log agenti
        - (da estendere) /api/mihub/mercati, /stalls, /vendors, /concessions, /qualificazioni, /wallet, /kpi, ecc.
    -
    - Secrets gestiti nel “cassetto chiavi” (niente più credenziali hardcodate).
-
- DB (Neon)
    - Tabelle rilevanti:
        - markets, stalls, vendors, concessions
        - agent_logs
        - (nuove) tabelle per qualificazioni, wallet, kpi ecc.
    -
-
- Agenti
    - MIO: orchestratore, parla solo nella chat principale con l’utente, e nelle viste multi-agente come log.
    - GPT Dev: scrive codice / fa PR su GitHub.
    - Manus: esegue script, fa deploy, sistemazione errori.
    - Abacus: query e analisi dati (KPI, indicatori, controlli incrociati).

    - Zapier: notifiche, automazioni esterne, integrazioni.
-

2. Sezioni UI – cosa sono e cosa toccano

2.1 Home / Overview

Scopo

Vista “stato del sistema” a colpo d’occhio.

Cosa deve mostrare

- Box KPI:
    - numero mercati attivi
    - numero posteggi totali / attivi / occupati
    - numero imprese
    - numero concessioni attive
    - % occupazione
-
- Ultimi errori gravi:
    - da Guardian / log backend

-
- Ultime attività agenti:
    - MIO, Manus, Abacus, Zapier (estratto da agent_logs)
-

Dati / sistemi

- Neon: markets, stalls, vendors, concessions
- agent_logs (per attività agenti)
- Log di backend/Guardian

Stato attuale

- Non ancora consolidata come “cruscotto” centrale. Alcune info sparse in altre sezioni.

TODO

- Una pagina unica che tira dentro:
    - conteggi da Neon
    - ultimi 10 errori
    - ultimi 10 eventi agenti (da agent_logs)
-
- Filtri base: Comune, Regione, periodo.

2.2 Gestione Mercati – Lista Mercati

Scopo

Gestire l’elenco dei mercati (fino a 8.000), con stato, tipologia, integrazioni.

Cosa deve mostrare per ogni mercato

- Codice gestionale DMS (5 cifre DMS)
- market.code interno (es. GR001)
- Nome mercato
- Tipologia:
    - giornaliero, settimanale, mensile, stagionale, rotazione, coperto, ecc.
-
- Comune, Provincia, Regione, eventuale Unione Comuni
- Posteggi:
    - totali
    - attivi
    - occupati
    - mq superficie vendita
-
- Giorni di svolgimento + orari operativi:
    - inizio occupazione
    - ultimo orario per marcare presenza
    - inizio spunta

    - consegna rifiuti
    - uscita
-
- Stato integrazione:
    - PDND / SSU / SSO: OK / warning / non configurato
-
- Flag:
    - “GIS caricato”
    - “Allineamento posteggi completato”
-

Dati / sistemi

- Neon: markets
- stalls per conteggi
- integrazioni (flag) SSU / SSO / PDND

Stato attuale

- Pannello mercati esiste già nel frontend, con dati di base.

TODO

- Allineare la lista mercati al modello completo sopra.
- Aggiungere filtri per Comune / Regione / tipo mercato.

- Collegare ogni riga a:
    - scheda mercato (tab Posteggi + GIS)
    - scheda Imprese / Concessioni del mercato
-

2.3 Gestione Mercati – Posteggi + GIS

Scopo

Allineare mappa GIS e lista posteggi con lo standard DMS, e lavorare sui posteggi in modo operativo.

Cosa deve mostrare (lista)

- stall_code_dms → codice univoco: XXXXX + lettera + NNN
- Numero visibile su GIS (1…185, con buchi)
- Stato posteggio:
    - libero / occupato / assegnato / sospeso / revocato
-
- Intestatario:
    - denominazione impresa attuale
-
- Metri e tipologia:
    - box, banco, obbligo mezzo dentro, obbligo scaricare, ecc.

-

Cosa deve mostrare (mappa GIS)

- Poligoni colorati per stato (verde / rosso / arancione / grigio)
- Click su poligono:
    - highlight riga corrispondente nella lista
-
- Click su riga:
    - centra/zoom sulla geometria in mappa
-

Azioni

- Selezionare uno o più posteggi:
    - per assegnazione / revoca / sospensione
-
- Link diretto:
    - scheda impresa
    - scheda concessione
-

Dati / sistemi

- Neon: stalls (con stall_code_dms)

- GIS: layer poligoni (editor v3 / Pepe GIS)
- Neon: concessions per legare posteggi ↔ concessioni ↔ impresa

Stato attuale

- Vista GIS + lista posteggi esiste ma va allineata allo standard codici e flussi operativi.

TODO

- Forzare uso stall_code_dms come chiave di allineamento.
- Implementare colore/stato coerente fra lista e GIS.
- Collegare alle azioni di creazione/modifica concessione.

2.4 Gestione Mercati – Imprese / Concessioni

Scopo

Gestire imprese e concessioni relative a un singolo mercato.

Cosa deve mostrare – lista imprese

- ID impresa DMS

- Denominazione
- P.IVA / CF
- Indirizzo sede
- Stato:
    - attiva / sospesa / revocata
-
- Numero concessioni collegate

Cosa deve mostrare – lista concessioni

- Codice concessione (del SUAP / Comune)
- Posteggio collegato (stall_code_dms)
- Data inizio / fine
- Stato:
    - attiva / subentro / revocata / restituita
-
- Collegamento a SCIA di subingresso
- Storico subentri / variazioni:
    - da chi, quando, protocollo
-

Azioni

- Crea / modifica impresa
- Crea concessione:

    - scegli impresa
    - scegli posteggio dal mercato (dal tab Posteggi + GIS)
-
- Vedi storico variazioni e stato concessione.

Dati / sistemi

- Neon: vendors (imprese)
- Neon: concessions
- Neon: stalls (lista posteggi del mercato)
- In futuro: SCIA da SSU/PDND

Stato attuale

- Tab esiste ma “mezzo funzionante” (modale impresa + concessione non ancora completa).

TODO

- Finire i flussi di creazione / modifica concessione.
- Incrociare con GIS (posteggi disponibili/occupati).
- Collegare questa sezione a Qualificazioni e Wallet.

2.5 Hub / Negozi (da ripristinare)

Scopo

Vista centrata su impresa/negozio, non sul mercato.

Cosa deve mostrare

Per ogni impresa:

- Codice impresa DMS
- Denominazione + logo (se presente)
- Elenco concessioni:
    - mercato, stall_code_dms, stato (attiva/sospesa/revocata)
-
- Eventuali punti vendita fissi (negozi)
- Indicatori:
    - giornate effettive di presenza
    - storicità concessioni
    - SCIA aperte/chiuse
-
- Eventuale profilo pubblico:
    - link a hub digitale, e-commerce, social
-

Azioni

- Apri scheda completa impresa

- Vedi mappa mercati in cui è presente
- Collegare in futuro a:
    - Hub e-commerce
    - Iniziative, bandi, rating impresa
-

Dati / sistemi

- Neon: vendors, concessions, stalls
- Eventuali tabelle di presenza giornaliera / transazioni (in futuro)

Stato attuale

- Sezione esisteva, persa col rollback. Va ripristinata.

TODO

- Ripristinare componente NegoziPanel / Hub.
- Collegare a Qualificazioni Impresa e Wallet della singola impresa.

2.6 Qualificazioni Impresa (NUOVA, ma già progettata)

Scopo

Avere la fotografia totale della compliance di un’impresa: formazione, requisiti sanitari, requisiti somministrazione, DURC, DVR, deleghe per presenza in mercato, ecc.

Cosa deve contenere per ogni impresa

- Attestati/certificati:
    - primo soccorso
    - antincendio (basso/medio rischio)
    - sicurezza sul lavoro (DLgs 81/08)
    - DVR
    - Haccp / requisiti sanitari
    - requisiti per somministrazione alimenti e bevande
-
- Deleghe:
    - elenco dipendenti / incaricati autorizzati a fare la presenza in app DMS al posto del titolare
-
- Scadenze:
    - ogni attestato con data rilascio, scadenza, ente rilascio
-
- DURC:
    - stato INPS/INAIL via PDND
    - data ultimo controllo
-

Funzioni

- Segnalare all’impresa:
    - cosa manca
    - cosa sta scadendo
-
- Segnalare alla PA:
    - chi è non in regola su requisiti minimi
-
- Futuro:
    - collegare a automazione SCIA / subingressi / bandi Bolkestein
    - contributo al rating impresa
-

Dati / sistemi

- Nuove tabelle Neon (già definite negli SQL che hai allegato)
- PDND / INPS / INAIL per DURC
- App DMS (per le deleghe operatori che marcano presenza)

Stato attuale

- Schema dati definito nei file SQL e nel documento “Qualificazioni Impresa – Schema Dati”.
- UI da sviluppare come tab dentro la scheda impresa + vista riepilogativa.

TODO

- Implementare endpoint /api/mihub/qualificazioni.
- Implementare UI tab “Qualificazioni” in Gestione Imprese / Hub Negozi.
- Prevedere un playground per enti formatori per caricare attestati rilasciati.

2.7 Wallet (NUOVO)

Scopo

Tenere sotto controllo pagamenti, tributi, canone unico, saldi e pendenze per ogni impresa.

Cosa deve mostrare

Per ogni impresa:

- Saldo complessivo
- Canone unico dovuto/pagato per posteggi
- Altri tributi collegati ai mercati / occupazione suolo
- Eventuali sanzioni / more
- Storico movimenti:
    - data, importo, descrizione, riferimento concessione/mercato
-

Azioni

- Visualizzare movimenti
- (Futuro) Innescare processi:
    - sollecito
    - rateizzazione
    - pagamento online
-

Dati / sistemi

- Nuove tabelle Neon (wallet / movimenti)
- Collegamento a:
    - concessioni
    - integrazioni eventuali con contabilità esterna
-

Stato attuale

- Concetto definito, UI non ancora realizzata.

TODO

- Endpoint /api/mihub/wallet (GET per impresa / GET aggregati / POST movimenti)

- Tab “Wallet” in scheda impresa + eventuale vista riassuntiva in KPI.

2.8 KPI / Indicatori

Scopo

Cruscotto numerico serio per Comune/Regione/periodo.

Cosa deve mostrare

- Numero mercati attivi per Comune
- % occupazione posteggi:
    - occupati vs totali
-
- Volume di SCIA:
    - nuovi titolari
    - subingressi, cessazioni
-
- Distribuzione:
    - tipologie merceologiche
    - giorni di svolgimento
-
- Indicatori fiscali e di rischio (in futuro con Wallet + Qualificazioni + DURC).

Azioni

- Filtri:
    - periodo (da / a)
    - Comune / Regione
    - settore merceologico
-
- Esportazione:
    - CSV / PDF
-
- Link:
    - “Apri dettaglio” → porta alla sezione di origine (mercati, imprese, ecc.)
-

Dati / sistemi

- Neon, interrogato soprattutto da Abacus
- SSU/PDND per SCIA (in futuro)
- Wallet per la parte economica

Stato attuale

- Non ancora implementato come sezione autonoma.

TODO

- Endpoint /api/mihub/kpi
- Componenti KpiPanel + grafici con libreria charts già presente.
- Collegamento con Abacus per calcoli più complessi.

2.9 Integrazioni & API

Scopo

Console tecnica: panoramica di tutte le API reali del sistema e stato integrazioni.

Cosa deve mostrare

- Tabella endpoint:
    - path, metodo, categoria
    - stato: OK / mock / non implementato
-
- Collegamento ai log Guardian:
    - ultimi errori per endpoint
    - medie di latenza
-
- Link diretto a:
    - API Playground (chiamate reali, JSON vero)

-

Azioni

- Testare endpoint dal Playground
- Filtrare gli endpoint per:
    - dominio (mercati, imprese, qualificazioni, wallet…)
    - stato implementazione
-

Dati / sistemi

- realEndpoints.ts + api/index.json nel blueprint
- Log Guardian e log backend MIO

Stato attuale

- Sezione c’è, ma va ripulita:
    - togliere roba finta
    - mostrare solo endpoint reali e utili.
-

TODO

- Allineare blueprint a implementazione reale.
- Collegare gli endpoint nuovi (qualificazioni, wallet, kpi).

2.10 Log & Debug

Scopo

Capire cosa sta succedendo nel sistema quando qualcosa non va.

Struttura ideale

- Tab System Logs:
    - log backend mihub-backend-rest (errori, warn, info)
-
- Tab Guardian Logs:
    - richieste API
    - errori MIO / Manus / Abacus / Zapier
-
- Tab Agent Traces:
    - agent_logs dettagliati per ogni conversazione
    - vista tipo “trace” con passaggi fra agenti
-

Azioni

- Filtri:
    - endpoint
    - agente
    - livello errore
    - conversation_id
-
- Drill-down:
    - entra nel singolo log:
        - request
        - response
        - stacktrace
    -
-
- Esportazione log per assistenza esterna.

Dati / sistemi

- Neon: agent_logs
- Guardian log (quando attivo)
- Log PM2/Node sul server Hetzner

Stato attuale

- C’è una parte di log, ma non tutto centralizzato.

TODO

- Portare tutto in una sezione unica Log & Debug.
- Collegare con vista multi-agente per tracciare una richiesta end-to-end.

2.11 Agenti AI (MIO + multi-agente)

Scopo

Zona in cui vedi e controlli MIO e i 4 agenti (GPT Dev, Manus, Abacus, Zapier).

Struttura attuale (dopo gli ultimi fix)

- Chat MIO principale (sopra):
  - SOLO user ↔ MIO
  - funziona con stato locale + chiamate a /api/mihub/orchestrator
-
- Vista 4 agenti (sotto, mini-chat):
  - read-only
  - mostra dialoghi MIO ↔ ogni agente, da agent_logs
-
- Vista singola agenti:

    - 4 chat isolate:
        - GPT Dev
        - Manus
        - Abacus
        - Zapier
    -
    - ognuna con proprio conversation_id separato
    - usano sendAgentMessage con mode: manual/auto a seconda dell’agente
-

Dati / sistemi

- Backend orchestratore
- agent_logs su Neon
- GitHub (GPT Dev / Manus)
- Zapier (automazioni)
- Neon (Abacus per query)

Stato attuale

- Chat MIO principale: OK.
- Vista 4 agenti: OK in sola lettura.
- Viste singole: implementate, da rifinire/testare end-to-end.

TODO

- Collegare GPT Dev in modo che:
    - legga/scriva su GitHub in modo sicuro (solo branch/PR).
-
- Far sì che Manus si usi come:
    - esecutore dei piani di MIO / GPT Dev (deploy, fix, script).
-

2.12 Impostazioni / Sicurezza

Scopo

Gestire utenti, ruoli, chiavi e ambienti.

Cosa deve mostrare

- Utenti della dashboard:
    - Comune
    - Regione
    - Super admin
-
- Ruoli:
    - chi può fare cosa (es. modificare mercati, solo vedere KPI, ecc.)

-
- Config integrazioni:
    - GitHub, Abacus, Zapier, PDND, ecc.
-
- Selezione ambiente:
    - backend attivo (prod/test)
    - configurazioni per simulazioni vs produzione
-

Dati / sistemi

- Tabelle users / roles (da creare/estendere)
- Cassetto chiavi (segreti, in futuro via UI in sola lettura)

Stato attuale

- Configurazioni esistono “a pezzi”, non ancora un pannello unico.

TODO

- Pannello chiaro:
    - elenco utenti
    - ruoli e permessi per sezione
    - test connessioni (ping Neon, orchestratore, GitHub, Zapier, PDND…).
-

MIO GPT-Dev, tipo:

“Questo è il sistema DMS Hub, queste sono le sezioni, questi sono i repo, questo è il tuo ruolo come GPT Dev…”

Così lo incolli come primo messaggio nella nuova chat e lui parte già “in contesto” senza che tu debba rispiegare tutto ogni volta.

Ok, andiamo diritti al punto: mettiamo in riga GPT-Dev e Manus una volta per tutte.

Ti preparo due prompt pronti da incollare + una mini check-list operativa per te.

1️⃣ Prompt per nuova chat “MIO GPT-Dev” (sviluppatore su GitHub)

Usalo in una nuova chat su ChatGPT, scegliendo il modello con connettore GitHub abilitato (quello che hai già usato ieri).

RUOLO E CONTESTO

Ti chiamo **MIO GPT-Dev**.

Sei lo sviluppatore principale del frontend **DMS Hub App New**.

Il progetto è questo:

- Repo principale frontend: `Chcndr/dms-hub-app-new` (branch: `master`)

- Dashboard PA: `https://dms-hub-app-new.vercel.app/dashboard-pa`

- Backend orchestratore: `https://orchestratore.mio-hub.me` (orchestratore MIO multi-agente)

- Backend principale: repo `Chcndr/mihub-backend-rest` (deploy su Hetzner, orchestrato da Manus)

- Database: Neon PostgreSQL

- Secrets / chiavi: gestiti tramite “Cassetto Chiavi” (NON devi toccare credenziali o variabili di ambiente)

ALTRI SISTEMI

- Manus AI: agente esterno che esegue script, fix, deploy su Hetzner e Vercel (NON sei tu).

- Zapier: usato per automazioni e webhook (non lo tocchi tu direttamente, a meno che non ti chieda espressamente di preparare file di config).

- Abacus: agente dati che parla con Neon ed altri dati (tu gli prepari le API/UI giuste).

IL TUO RUOLO (VINCOLI CHIARI)

1. Usi il **connettore GitHub** SOLO per:
   - leggere file, struttura e commit dei repo;
   - proporre modifiche con patch/PR, mai modifiche “magiche” non spiegate.
2. Tu NON fai:
   - deploy su Hetzner;
   - modifiche a segreti/env;
   - comandi da terminale;
   - cambi su Docker, PM2, server, ecc.
3. Produci sempre:

- piano di modifica in Markdown;

- elenco file da toccare;

- patch di codice complete (pronte da incollare in GitHub o da usare per una PR).

STATO ATTUALE (RIASSUNTO)

- La **chat MIO principale** (sopra) ora funziona: invia messaggi a `/api/mihub/orchestrator` su `orchestratore.mio-hub.me` e mostra la risposta subito (stato locale React).

- Il backend salva i log degli agenti in `agent_logs` su Neon, accessibili via `/api/mio/agent-logs`.

- La vista **4 agenti** usa `useAgentLogs` ed è READ-ONLY (MIO ↔ GPT-Dev / Manus / Abacus / Zapier).

- La vista **singola agenti** ha 4 chat isolate (GPT-Dev, Manus, Abacus, Zapier), ognuna con il proprio `conversationId`.

- Tutti i casini grossi su persistenza cronologia sono stati sistemati da Manus, ma il codice ora è delicato: VA tenuto coerente con questa architettura.

COSA TI CHIEDO ORA

1. Conferma lo stato attuale che riesci a leggere dal repo `Chcndr/dms-hub-app-new` usando il connettore GitHub:

   - Struttura intorno a:

     - `client/src/pages/DashboardPA.tsx`

     - `client/src/components/multi-agent/MultiAgentChatView.tsx`

     - `client/src/hooks/useAgentLogs.ts`

     - `client/src/lib/mioOrchestratorClient.ts`

     - `vercel.json`

   - Come sono implementati:

     - chat MIO principale;

     - vista 4 agenti (read-only);

     - vista singola per GPT-Dev / Manus / Abacus / Zapier.

2. Poi comportati così, SEMPRE:

- Quando ti chiedo una modifica:

- prima fai una **breve analisi** (file coinvolti, rischi);

- poi proponi **patch di codice** chiare e complete (```diff o blocchi completi di file/pezzi di file);

- separa sempre i passi: “Cosa devo cambiare io in GitHub” vs “Cosa deve fare Manus (deploy / script)”.

3. Non fare commit/push autonomi senza che te lo dica esplicitamente.

Il flusso deve essere:

- TU → proponi patch / PR;

- IO → applico le patch o le faccio applicare a Manus;

- Manus → fa i deploy.

PRIMO TASK

Per iniziare, fai questo:

1. Controlla in sola lettura il repo `Chcndr/dms-hub-app-new` (branch `master`).

2. Dimmi:

- quali file usano ancora `useAgentLogs` per MIO principale;

- come sono definite la vista 4 agenti (read-only) e la vista singola;

- se vedi ancora codice vecchio inutile o punti deboli nella gestione dei `conversationId`.

Rispondi con:

- una sezione “MAPPA FILE”;

- una sezione “STATO CHAT MIO / MULTI-AGENTE”;

- una sezione “PUNTI DEBOLI / COSA METTEREI A POSTO PRIMA”.

Poi ti assegnerò il primo task di codice vero (feature o refactor mirato).

2️⃣ Prompt per Manus GitHub (esecutore / deploy)

Questo lo incolli nella chat di Manus GitHub (quella che già usi per fare fix & deploy).

OGGETTO: Nuova organizzazione lavoro tra MIO GPT-Dev e Manus

Da ora in poi organizziamo il lavoro così:

RUOLI

- **MIO GPT-Dev (ChatGPT con connettore GitHub)**:
  - legge il codice dei repo (soprattutto `Chcndr/dms-hub-app-new` e `Chcndr/mihub-backend-rest`);
  - prepara patch / PR con il codice nuovo;
  - NON fa deploy, NON tocca server Hetzner, NON tocca segreti.

- **TU (Manus)**:
  - applichi le patch su GitHub (merge PR, edit file, commit);
  - esegui script di deploy su Hetzner per il backend, e ti assicuri che tutto riparta;
  - controlli i log, sistemi errori di build, config, CORS, Vercel, ecc.;
  - NON reinventi la logica di MIO se non ti chiedo esplicitamente di farlo.

INFRASTRUTTURA (PROMEMORIA)

- Frontend: `Chcndr/dms-hub-app-new` → deploy automatico su Vercel (`dms-hub-app-new.vercel.app`).

- Backend orchestratore: `Chcndr/mihub-backend-rest` → deploy su Hetzner (`https://orchestratore.mio-hub.me`).

- DB: Neon PostgreSQL.

- Chiavi / secrets: gestiti dal “Cassetto Chiavi” e variabili d’ambiente, già sistemati (OPENAI, GEMINI, NEON, HETZNER_SSH_KEY, ecc.).

- Deploy backend: SOLO tramite lo script che hai documentato nel report “Procedura Deploy Definitiva – Solo Manus, Mai l’Utente”.

MIO ORA È COSI:

- Chat principale MIO (sopra):

  - usa `sendMioMessage` su `https://orchestratore.mio-hub.me/api/mihub/orchestrator`;

  - mostra SUBITO il messaggio utente (stato locale React);

  - mostra dopo la risposta backend (`data.message` / `data.reply`).

- Vista 4 agenti:

  - READ-ONLY, usa `useAgentLogs` con `mioMainConversationId`;

  - mostra MIO, GPT-Dev, Manus, Abacus, Zapier.

- Vista singola:

  - 4 chat isolate, ognuna con `conversationId` separato;

  - handler `sendAgentMessage` per ogni agente.

COSA TI CHIEDO PRATICAMENTE

1. Quando ti mando un messaggio del tipo:

   - “Prendi la patch / PR di MIO GPT-Dev per [feature X] e applicala”

- tu:

  - apri il repo indicato;

  - applichi la patch esattamente come scritto;

  - fai commit chiaro (es: `feat: aggiungi vista KPI mercati`);

  - se serve, lanci lo script di deploy Hetzner per il backend.

2. Quando MIO GPT-Dev propone modifiche a:

- `DashboardPA.tsx`

- `MultiAgentChatView.tsx`

- `useAgentLogs.ts`

- `mioOrchestratorClient.ts`

- `vercel.json`

verifichi che:

  - non rompa la chat MIO principale;

  - non reintroduca chiamate sbagliate (niente `/api/orchestrator` vecchio, solo `/api/mihub/orchestrator`);

  - l'URL backend resti `https://orchestratore.mio-hub.me`.

3. Se devo fare un nuovo deploy backend:

- usi SEMPRE lo script che hai documentato tu nel report "Procedura Deploy Definitiva – Solo Manus, Mai l'Utente" (ssh → git pull → npm install → pm2 restart → health check).

In sintesi:

- MIO GPT-Dev scrive il codice,

- TU lo applichi e fai i deploy,

- Io non devo più toccare SSH, PM2, o comandi a mano.

Conferma che hai capito e riassumi tu in 5 bullet cosa farai quando dico:

"Usa il lavoro di MIO GPT-Dev per implementare [X]".

3️⃣ Come li usi nella pratica (flow operativo)

Per non perderti:

1. Nuova chat con GPT-Dev
    - Apri ChatGPT → nuova chat → scegli modello con GitHub connector.
    - Incolla il prompt del punto 1️⃣.
    - Dagli eventualmente conferma che il repo è privato e che userai tu le patch.
2.
3. Allinea Manus
    - Vai sulla chat abituale di Manus GitHub.
    - Incolla il prompt del punto 2️⃣.
    - Fatti dare i suoi “5 bullet” così sei sicuro che ha capito il ruolo.
4.
5. Per ogni nuova cosa da sviluppare:
    - Step A – GPT-Dev:
      “Implementa [feature X] su dms-hub-app-new: analizza repo, proponi patch, non fare deploy.”
    - Lui ti restituisce: file da modificare + patch.
    - Step B – Manus:
      “Applica le patch di MIO GPT-Dev per [feature X] nel repo [link], poi fai deploy se serve.”
    - Manus applica e deploya.
    - Tu testi sulla Dashboard PA in produzione.
6.

Perfetto, ti preparo il Task #1 pronto da incollare a MIO GPT-Dev.

Lo puoi usare così com'è in una nuova risposta nella sua chat.

TASK #1 – Implementare sezione "Qualificazioni Impresa" nella Dashboard PA

RUOLO

Tu sei MIO GPT-Dev, sviluppatore principale del frontend `Chcndr/dms-hub-app-new`.

Manus si occupa solo di applicare le patch e fare i deploy.

NON devi toccare:

- logica della chat MIO principale (che ora funziona),

- orchestratore backend già in produzione,

- configurazioni sensibili / secrets / deploy.

🎯 OBIETTIVO

Implementare una nuova sezione **"Qualificazioni Impresa"** nella **Dashboard PA**, riusando quanto possibile di:

- Gestione Mercati / Negozi (UI, tabelle, layout),

- infrastruttura multi-agente già esistente (MIO, Manus, Abacus, Zapier),

- schema dati che poi ti fornirò (SQL + MD già definiti).

Questa sezione serve per:

- vedere l'elenco delle imprese,

- vedere lo stato delle loro **qualificazioni** (licenze, concessioni, DURC, ISO, ecc.),

- preparare il terreno per far lavorare gli agenti (Abacus/MIO) su queste informazioni.

In questo primo task ti chiedo principalmente:

- **struttura frontend + API client**,

- **placeholders seri**, NON finti lorem ipsum, ma campi coerenti (codice impresa, CF/P.IVA, tipo qualificazione, scadenza, stato, note).

🏗️ CONTESTO ARCHITETTURA (DA RISPETTARE)

Repo: `Chcndr/dms-hub-app-new` (branch `master`)

Parti rilevanti:

- `client/src/pages/dashboard-pa/index.tsx`

- `client/src/pages/DashboardPA.tsx` (o struttura equivalente, a seconda del refactor attuale)

- `client/src/components/...` (MercatiPanel, NegoziPanel, LogPanel, MIOAgent, ecc.)

- `client/src/components/multi-agent/MultiAgentChatView.tsx`

- `client/src/lib/api.ts` o client HTTP equivalente

- `client/src/lib/mioOrchestratorClient.ts` (non toccare per questo task)

- `client/src/hooks/useLogs.ts`, `useAgents.ts` (da guardare per stile)

Backend: esiste già `mihub-backend-rest` con orchestratore e API varie.

Per questo task tu:

- puoi proporre gli endpoint REST da aggiungere (firma + payload),

- puoi proporre patch per `mihub-backend-rest`,

- ma NON fai deploy: quello lo farà Manus.

📚 DATI “QUALIFICAZIONI IMPRESA”

Il dettaglio di tabelle e SQL lo ha già definito Andrea in:

- `Modulo “Qualificazioni Impresa” – Schema Dati Definitivo.md`
- `001_qualificazioni_impresa.sql`
- `002_fix_compliance_view.sql`

Per ora assumiamo una struttura logica tipo:

- Tabella IMPRESE (semplificata):
  - `id_impresa`
  - `ragione_sociale`
  - `piva`
  - `codice_fiscale`
  - `comune`
  - `settore`

- Tabella QUALIFICAZIONI:
  - `id_qualificazione`
  - `id_impresa`
  - `tipo` (es. “CONCESSIONE MERCATO”, “DURC”, “ISO9001”)
  - `ente_rilascio`
  - `data_rilascio`
  - `data_scadenza`

- `stato` (es. ATTIVA / SCADUTA / IN_VERIFICA)

- `note`

Useremo questa logica come guida per UI e API, anche se la struttura reale nel DB è più ricca.

🖥️ REQUISITI FRONTEND

1. **Nuovo tab / sezione “Imprese & Qualificazioni” nella Dashboard PA**

- Aggiungi una voce nel menu/tabs principale, tipo:

  - “Mercati”

  - “Negozi”

  - “GIS”

  - “Log”

  - “MIO”

  - **“Imprese & Qualificazioni”** (nome da confermare, ma usalo come base)

- Deve aprire un nuovo pannello dedicato.

2. **Nuovo componente principale**

Crea un componente React:

- `client/src/components/imprese/ImpreseQualificazioniPanel.tsx`

  (se preferisci una struttura tipo `qualificazioni/QualificazioniPanel.tsx` va bene, ma scegli una convenzione pulita)

All’interno:

- layout a 2 colonne:

  - sinistra: elenco imprese (tabella),

  - destra: dettaglio qualificazioni della impresa selezionata (tabella + meta).

3. **Tabella imprese (colonna sinistra)**

Colonne minime:

- Ragione sociale

- P.IVA / CF

- Comune

- Settore (se disponibile / placeholder)

- Numero qualificazioni attive (anche come placeholder calcolato lato frontend se non hai ancora API pronte)

Comportamento:

- click su una riga → seleziona impresa e carica (o simula) le qualificazioni sulla destra;

- evidenzia la riga selezionata.

4. **Tabella qualificazioni (colonna destra)**

Colonne:

- Tipo qualificazione (CONCESSIONE, DURC, ISO…)

- Ente rilascio

- Data rilascio

- Data scadenza

- Stato (pill/label colorata: verde = ATTIVA, arancione = IN VERIFICA, rossa = SCADUTA)

- Note (troncate, con tooltip su hover)

Sopra la tabella:

- nome impresa

- P.IVA / CF

- un badge “X qualificazioni totali – Y attive – Z in scadenza (< 60 gg)”

5. **Stile/UI**

- Riusare componenti UI che già usate in Mercati/Negozi (tabella, card, badge).

- Niente cose pesanti: pannello deve essere veloce da caricare.

- Niente logica con agenti qui dentro per ora: questa sezione è “dati e visualizzazione”.

🌐 API / CLIENT HTTP

Voglio che tu imposti già il **contratto API** per Manus / backend, anche se i dati ora possono essere mock / fake.

1. Definizione client-side (in TypeScript)

In `client/src/lib/api.ts` (o dove è centralizzata la chiamata), definisci tipi e funzioni:

```ts
export interface ImpresaDTO {
  id: string;
  ragioneSociale: string;
  piva: string;
```

```
  codiceFiscale?: string;
  comune?: string;
  settore?: string;
  qualificazioniAttiveCount?: number;
}

export interface QualificazioneDTO {
  id: string;
  impresaId: string;
  tipo: string;
  enteRilascio?: string;
  dataRilascio?: string;   // ISO string
  dataScadenza?: string;   // ISO string
  stato: 'ATTIVA' | 'SCADUTA' | 'IN_VERIFICA';
  note?: string;
}

export async function fetchImprese(): Promise<ImpresaDTO[]> { ... }

export async function fetchQualificazioniByImpresa(impresaId: string):
Promise<QualificazioneDTO[]> { ... }
```

2. Endpoint REST che proponi per il backend (solo contratto)
    - GET /api/mihub/imprese
        - Query opzionali:

        - q (search per testo)
        - comune
        - settore
    -
  - GET /api/mihub/imprese/:id/qualificazioni
    - Ritorna array di QualificazioneDTO.
  -

3. Tu nel codice frontend puoi:
   - usare dummy endpoint tipo /api/mock/imprese OR
   - già puntare a /api/mihub/imprese sapendo che per ora potrebbe rispondere 404 in dev.
4. Ma nel report che generi devi scrivere CHIARAMENTE:
   - quali endpoint servono,
   - che cosa devono restituire.
5.

⚙️ COSA NON DEVI TOCCARE IN QUESTO TASK

- Tutta la parte chat MIO / multi-agente / useAgentLogs / conversationId.
- Orchestratore mioOrchestratorClient.ts se non strettamente necessario.
- Config Vercel e rewrites esistenti per /api/mihub/orchestrator.
- Qualsiasi file usato solo da Manus per deploy.

📦 OUTPUT CHE VOGLIO DA TE

Rispondi con:

1. ANALISI (breve ma concreta)
    - quali file toccherai,
    - quali componenti riuserai (tabelle, layout, ecc.).
2.
3. PIANO TECNICO PASSO-PASSO
    - Step 1: aggiungere tab/menu
    - Step 2: creare ImpreseQualificazioniPanel.tsx
    - Step 3: definire tipi e client API
    - Step 4: integrazione nella Dashboard
    - ecc.
4.
5. PATCH DI CODICE
    - blocchi completi per:
        - nuovo componente panel,
        - modifiche a tabs/menu,
        - nuove funzioni in api.ts,
    -
    - tutto in TypeScript/React, compatibile con lo stile esistente del repo.
6.
7. NOTE PER MANUS
    - se per farlo funzionare davvero servono:

        - nuovi endpoint backend,
        - migrazioni DB,
        - config Vercel,
    -
    - scrivi una sezione “TODO per Manus” con elenco chiaro.

8.

Quando hai finito questo task e le patch sono pronte, ti assegnerò il task successivo (integrazione agenti: es. far lavorare Abacus sulle qualificazioni per calcolare stati di rischio/ compliance).

Quando l’hai incollato a GPT-Dev e lui ti risponde con analisi + patch, girami qui la sua risposta (o almeno le parti di codice chiave) e ti aiuto a:

- controllare che non rompa la chat / multi-agente,

- preparare il messaggio giusto per Manus per applicare le modifiche e fare il deploy.