Linee guida aziendali per i nuovi progetti software

Data	**2026-09-18**
Versione	1.2
Per chi	chi apre un nuovo progetto (PM, tech lead), chi lo stima, chi lo sviluppa, chi lo mette in esercizio
Riferimento tecnico	il framework MAEFWK10 (`C:\FWK10\maefwk10`, `CLAUDE.md` e `ONBOARDING.md` alla radice)
Versione leggibile	https://claude.ai/artifact/PMQeeF3uPPGp6Skv3cH8Fo

Questo documento dice **come si fa un progetto nuovo in azienda**: con quale stack, quando si parte dal framework e quando da zero, come si chiamano le cose, quali pattern e librerie si usano, come si testa e si documenta perché il codice resti mantenibile anche con Claude Code, e cosa deve esserci prima di dire «finito». Le regole nascono dal framework MAEFWK10 e dai suoi errori già pagati: ogni volta che una regola ha una storia dietro, la storia è citata, perché una regola senza il perché viene aggirata alla prima scadenza.

---

1. Le tre regole che governano tutto il resto

R1 — Lo standard è lo stack del framework

Ogni nuovo progetto usa lo stack del framework MAEFWK10, descritto al §3: **C# / .NET 10** sul server, **React + TypeScript + Mantine** sul client, **SQL Server o MySQL** come database relazionale, **MongoDB** per log e documenti, **Docker su Kubernetes (K3s)** per l'esercizio. Non è una preferenza: è il perimetro dentro cui l'azienda sa assumere, formare, fare code review, mettere in produzione e mantenere per dieci anni.

R2 — Ogni variazione va chiesta e autorizzata prima, per iscritto

Qualsiasi scostamento dallo stack standard — un linguaggio diverso, un altro framework UI, un altro database, un ORM diverso, una libreria a licenza commerciale nuova, un servizio cloud gestito che sostituisce un pezzo dello stack — richiede una **richiesta esplicita e un'autorizzazione esplicita** della direzione tecnica **prima** che si scriva la prima riga. La richiesta si fa con il modulo del §11. «L'ho già fatto così, ormai è pronto» non è una richiesta: è un debito che qualcun altro pagherà.

Non è una variazione, e non va chiesta: una libreria open source di supporto che non sostituisce un pezzo dello stack (un parser, un client per un'API esterna, un componente Mantine-compatibile). Va però verificata al §9 (licenza, audit vulnerabilità, manutenzione attiva).

R3 — Il punto di partenza lo decidono i requisiti, non le giornate

Si contano i requisiti presenti. Sono quattro:

- utenti che si autenticano (Identity, JWT, OTP, login Google, passkey, API key);
- dati che appartengono a più clienti o società, o una gerarchia di tenant, o un portale clienti;
- ruoli, permessi e claim granulari per entità e verbo, profili a livelli, perimetri per-feature;

- un frontend con CRUD su più entità, che richiede API JSON:API e tipi TypeScript generati dal contratto.

Sono le cose che il framework ha già risolte e collaudate: rifarle costa sempre più che ereditarle, e rifarle male costa molto di più.

Si parte dal fork del framework quando i requisiti presenti sono tre o più.

Il confronto con il team framework non è facoltativo, e non dipende dal conteggio. Prima di chiudere la scelta — qualunque sia il numero — si parla con uno sviluppatore del team framework. È una conversazione di dieci minuti, non un modulo: serve a capire se un requisito è marginale (un login unico per un tool interno) oppure è la punta di qualcosa che porterà gli altri dietro di sé. Chi conosce il framework se ne accorge subito; chi non lo conosce lo scopre a metà progetto.

L'esito del confronto è la decisione, e vale anche contro il conteggio. Se i requisiti sono tre ma il team conclude che il fork non serve, si parte da un progetto nuovo; se è uno solo ma il team vede arrivare gli altri, si parte dal fork. Il numero orienta la conversazione, non la sostituisce.

L'esito si scrive sempre in `docs/DECISIONI.md`, con **chi** ha deciso, **quando**, e **quali requisiti riaprirebbero la scelta**. Senza un nome e una data quella riga non spiega niente: fra un anno nessuno saprà se il progetto non è un fork per una ragione o per una dimenticanza.

Si parte da un **progetto nuovo** quando nessuno di quei requisiti c'è: uno script pianificato, un connettore fra sistemi, un worker senza interfaccia, un tool interno monouso. Lo stack resta comunque quello standard (R1), e valgono le regole minime del §4.

Le giornate non decidono. Una stima alta su un progetto senza quei requisiti non impone il framework; una stima bassa su un progetto che ne ha due non esenta dal partire da lì.

Se in corso d'opera emerge un requisito nuovo — «poi serviva anche il login», «poi i dati erano di due società» — la scelta **si riapre** e va ridiscussa. Non si continua per inerzia su un progetto nato senza.

---

2. Cos'è il framework, in una pagina

MAEFWK10 è un framework enterprise **.NET 10** con architettura **fork-based**: si fa un fork del repository e si aggiunge la logica del cliente direttamente dentro, con normali project reference. Niente plugin caricati a runtime, niente reflection per scoprire moduli esterni. Un solo binario, un solo container.

Cosa porta già fatto:

- **API JSON:API** generate dalle entità (`[Resource]` → CRUD completo con filtri, sort, paginazione, include, sparse fieldset), tramite JsonApiDotNetCore.
- **Autenticazione** ASP.NET Core Identity + JWT Bearer + API key, login Google, OTP, passkey.
- **Multi-tenancy** con query filter automatici (`IHasTenant`), gerarchia tenant, portale clienti.
- **Autorizzazione** a claim per entità e verbo, profili a livelli (0-100), perimetri per-feature.
- **Persistenza** EF Core 10 su SQL Server (MySQL supportato), migration con baseline, seed idempotenti.
- **Log** su file e MongoDB, audit, retention.
- **Job pianificati** Quartz.NET, notifiche push (Firebase), e-mail (MailKit), template.
- **Media e documenti** (upload, categorie, album), export Excel, PDF/immagini.
- **Test d'integrazione** con Testcontainers (SQL Server in Docker) su tutta la superficie API.

- **Deploy** Docker → registry privato → K3s con skill automatiche (`/deploy-dev`, `/deploy-prod`).

La mappa dettagliata è in `docs/ARCHITETTURA-E-MODULI-v2.md`; il quick start in `ONBOARDING.md`.

---

3. Lo stack standard

Le versioni sono quelle in uso nel framework al 15 settembre 2026. Un progetto nuovo parte da queste o da versioni **superiori** nella stessa major; il salto di major si fa prima nel framework, poi nei fork.

3.1 Backend

Componente	Scelta	Versione	Note
Runtime / linguaggio	.NET / C#	10.0 / `LangVersion latest`	`Nullable` **per file**, non globale (vedi `Directory.Build.props`: acceso globalmente cambia lo schema DB)
Web	ASP.NET Core	10.0	Controller che estendono `ApiController`
API	JsonApiDotNetCore	5.10	Specifica JSON:API; formatter Newtonsoft per gli endpoint non JSON:API
ORM	Entity Framework Core	10.0	Provider `SqlServer` (default) o `MySql.EntityFrameworkCore`
Micro-ORM	Dapper	2.1	Solo per query critiche misurate, non come default
Mediator / CQRS	MediatR	13.x	Licenza commerciale, già coperta. Eventi in-process, non persistiti
Mapping	AutoMapper	16.x	Licenza commerciale, già coperta. Via `IMapper` in DI
Validazione	FluentValidation (SimpleResults)	4.x	Regole di valore negli hook `OnWritingAsync`
Scheduler	Quartz.NET	3.17	Job in `MIT.Fwk.Scheduler`, `EnableJobs` in config
Identity / auth	ASP.NET Core Identity + JWT Bearer	10.0	Multi-tenant
Documentale / log	MongoDB.Driver	3.11	Log, documenti, audit. **Mai** `ToJson()` di Bson per risposte HTTP
Logging	log4net + `ILogService`	3.x	File + Mongo; `ILogService` in DI, non helper statici

Componente	Scelta	Versione	Note
OpenAPI	Swashbuckle	10.x	Snapshot del contratto sotto test
E-mail	MailKit	4.x	Via `IEmailService`
Excel / Office	ClosedXML, DocumentFormat.OpenXml	0.104 / 3.5	Import/export
Immagini	SkiaSharp	3.x	Con native assets Linux per il container
Push	FirebaseAdmin	3.x	—
Testo / nomi	Humanizer.Core	3.x	**Attenzione**: cambia i nomi degli schemi Swagger fra major

3.2 Frontend

Riferimento: `docs/NEW-FRONTEND-PLAN.md` (§3 e §9).

Componente	Scelta	Versione	Note
Framework	React	19	SPA, niente SSR per app autenticate
Linguaggio	TypeScript	5.6+ **strict**	Tipi generati dal contratto BE (`docs/contract`)
Build	Vite	6	Alias `@/`
Design system	**MaeDesign**	v3	`docs/MAEDESIGN-v3.md` + `src/shared/design/`: token, primitivi, pattern di pagina. È il sistema; Mantine è la libreria con cui è implementato
Libreria UI	Mantine	9	core, hooks, form, dates, notifications, modals, spotlight, charts, dropzone, tiptap
Data grid	**mantine-datatable**	—	Il componente tabellare unico. Si usa il preset condiviso `useMaeTable`, **non** la configurazione a mano pagina per pagina: altrimenti le stesse regole vengono riconfigurate ovunque, e divergono
Icone	@tabler/icons-react	—	Il set ufficiale Mantine
Routing	TanStack Router	1	File-based, type-safe
Server state	TanStack Query	5	Sopra un adapter JSON:API
UI state	Zustand	5	Minimale, solo stato

Componente	Scelta	Versione	Note
			non-server
Validazione	zod + mantine-form-zod-resolver	—	Form, env, route params
Date	dayjs	—	—
Permessi	CASL (`@casl/ability`, `@casl/react`)	—	Alimentato dai claim JWT
i18n	i18next	—	IT/EN, bootstrap da BE
Mobile	Capacitor	6	Shell native iOS/Android, **niente Cordova**
Font	IBM Plex Sans + IBM Plex Mono	—	Da CDN oppure self-hosted con `@fontsource`. **Il self-hosting non richiede autorizzazione**: in PA il CDN è spesso vietato, e senza questa riga ogni progetto riapre lo stesso caso
Stile	Tema Mantine + CSS variables	—	**Vincolante**: il tema porta forma e tipografia, i colori stanno nelle CSS variables. Nessun valore letterale in `features/`
Lint / format	ESLint + Prettier	—	`npm run lint` a zero warning è un gate

Non ammessi senza autorizzazione (R2): Angular, Vue, Next/Nuxt con SSR, MUI, Ant Design, Bootstrap, Tailwind come sistema primario, jQuery e derivati, Redux per il server state, `mantine-react-table`, set di icone diversi da Tabler.

3.3 Dati

Componente	Scelta	Note
Relazionale	**SQL Server** (default) o **MySQL**	Il provider si sceglie per progetto (`customsettings.json`), il codice resta portabile: niente T-SQL nelle entità, SQL vendor-specific solo in script dichiarati
Documentale	**MongoDB**	Log, audit, documenti opachi. Non è il database dei dati transazionali
Migration	EF Core Migrations	Una history table per DbContext; baseline registrata sui DB esistenti, mai rieseguita
Chiavi	`Guid` (`uniqueidentifier`)	`string` solo per id **assegnati da altri** (fiscale, PEC, ERP, tag NFC)

Non ammessi senza autorizzazione: PostgreSQL, Oracle, SQLite in produzione, Redis, Elastic, altri NoSQL.

3.4 Test

Livello	Backend	Frontend
Unit	xUnit 2.9 + FluentAssertions 8 + NSubstitute 5	Vitest + Testing Library
Integrazione	`WebApplicationFactory` + **Testcontainers.MsSql** (SQL Server vero in Docker)	MSW (mock dal contratto BE)
End-to-end	—	Playwright sugli scenari critici
Contratto	Snapshot Swagger + diff automatico (`SwaggerContractDumpTests`)	Tipi generati dal contratto

3.5 Infrastruttura e strumenti

Ambito	Strumento
Sorgenti	Git; branch `dev` (integrazione) e `main` (rilascio); feature branch `feature/<slug>`
Container	Docker; in locale Rancher Desktop (`rdctl start`)
Orchestrazione	K3s: cluster dev (`*.maestraledev.it`) e pre-prod (`*.demo.maestraledev.it`)
Registry	Registry privato aziendale
Deploy	Skill `/deploy-dev <versione>` e `/deploy-prod <versione>`: build, push, `set image`, log cronologico. Non si replicano gli step a mano
IDE	Visual Studio 2026 o Rider per il BE; VS Code per il FE
DB tooling	SSMS / Azure Data Studio; MongoDB Compass; `sqlcmd` (con `SET QUOTED_IDENTIFIER ON` in testa agli script che creano procedure)
Assistente di codice	Claude Code, con un `CLAUDE.md` nel repo che raccoglie le convenzioni del progetto
Script di supporto	PowerShell per gli script di repo; Python per i tool di analisi in `docs/tools`
Sicurezza supply chain	`NuGetAudit` su tutto l'albero (transitive incluse), SBOM (`scripts/sbom/Generate-Sbom.ps1`)

---

4. Framework o progetto nuovo: come si decide

Si contano i requisiti presenti. **Tre o più → framework. Sotto i tre → progetto nuovo.** In entrambi i casi la scelta si chiude **dopo** il confronto con il team framework, il cui esito prevale sul conteggio e si scrive in `docs/DECISIONI.md` (R3).

Domanda	Se sì
Ci sono utenti che fanno login?	requisito da framework (Identity, JWT, OTP, Google già fatti)

Domanda	Se sì
I dati appartengono a più clienti o società?	requisito da framework (multi-tenant con query filter)
Servono ruoli, permessi e claim granulari per entità e verbo?	requisito da framework (claim, profili a livelli, perimetri)
Ci sarà un frontend con CRUD su più entità?	requisito da framework (JSON:API generato, tipi TS dal contratto)
È uno script pianificato, un connettore, un tool interno monouso senza UI?	progetto nuovo (console/worker .NET)

La **durata prevista** e l'esistenza di una seconda fase non sono requisiti, ma sono segnali: a parità di tutto il resto, un progetto che vivrà a lungo conviene farlo partire dal framework, perché i requisiti di cui sopra tendono ad arrivare dopo.

Un **progetto nuovo** non è un progetto senza regole. Deve comunque avere: repository Git con la struttura del §10, `README` e `CLAUDE.md`, Dockerfile, configurazione esterna al binario (nessun segreto nel repo), log strutturato, almeno i test dei percorsi principali, pipeline di deploy sul cluster.

Un **fork del framework** si fa così: fork del repo, nuovo progetto `MIT.Fwk.<Modulo>` in `Src/`, project reference in `MIT.Fwk.WebApi.csproj`, entità come `DbSet` di `JsonApiDbContext` (o DbContext dedicato che implementa `IJsonApiDbContext` se serve un secondo database), chiave in

`dbconnections.json` uguale al nome della classe DbContext, `DatabaseMigrationOrder` aggiornato. Passo per passo in `CLAUDE.md` § «Adding Custom Modules» e nel `README.md` di `Src/MIT.Fwk.Examples`.

---

5. Architettura e design pattern

5.1 Strati

```
Core            interfacce, contratti, multi-tenancy (IHasTenant, ITenantProvider), helper
Domain          comandi, eventi, validazioni (CQRS leggero su MediatR)
Infrastructure  DbContext, entità, migration, servizi dati (*ManualService), Identity, Mongo
WebApi          controller, ResourceDefinition (hook per entità), middleware, configurazione DI
Scheduler       job Quartz
```

Le dipendenze vanno **verso il basso**: WebApi conosce Infrastructure, Infrastructure conosce Core, mai il contrario. Un'entità non conosce il controller che la espone.

5.2 Pattern da usare

Pattern	Come	Non fare
Dependency Injection ovunque	costruttori con interfacce; `IOptions<T>` per la configurazione; `IConnectionStringProvider`, `ILogService`, `IEmailService`, `IEncryptionService`, `IMapper`	helper statici (`ConfigurationHelper`, `LogHelper`, `MailHelper`: deprecati, spariscono in v9)
DbContext diretto come repository	`_context.Set<T>()`, `AsNoTracking()` sulle letture	repository generici e `IAppService` a strati: rimossi in v8 perché duplicavano EF
Auto-discovery per convenzione	interfaccia `I*ManualService` + classe `*ManualService` → registrazione	registrazioni manuali sparse; `IApplicationServiceHandler` solo

Pattern	Come	Non fare
	automatica come scoped	per ciò che non segue la convenzione
Hook per entità (`IResourceDefinition`)	validazioni in `OnWritingAsync`, seed e side-effect in `OnWriteSucceededAsync`, perimetri in `OnApplyFilter` **e** mascheramento in `OnSerialize`	logica di dominio nel controller; su entità `IHasTenant` la ResourceDefinition **deve** estendere `TenantAwareResourceDefinition`
CQRS leggero	`Command/Event` via `IMediatorHandler` quando un'operazione ha più effetti da orchestrare	event sourcing: l'Event Store è stato rimosso, gli eventi non sono persistiti
DTO + AutoMapper	profili di mapping, `_mapper.Map<Dto>(entity)`	`Ok(entity)`: scavalca mascheramento, sparse fieldset e perimetri
Eccezioni al middleware	lanciare; `UnhandledExceptionMiddleware` produce il corpo d'errore e la riga di log con traceId	`try/catch` nel controller che logga su console e risponde a mano
Errori di dominio tipizzati	`422` con codice `UPPER_SNAKE_CASE` documentato nel contratto (`MILESTONE_ONLY_ON_FEATURE`, `MODULE_DISABLED`)	messaggi liberi in italiano che il FE deve parsare
Seed idempotente al posto giusto	evento di creazione → hook CRUD; catalogo allargato al deploy → seeder `SeedAllTenantsAsync` + endpoint admin; correzione one-shot → migration; bootstrap DB vuoto → migration dedicata	cataloghi nelle migration; regole C# ricopiate in SQL a mano (divergono in silenzio)
Guardie di deduplica	riga scritta solo per stati **terminali**; ordine «prenota → invia → conferma»	archiviare uno stato «non ancora applicabile»: consuma l'unica occasione e la funzione tace per sempre
Job Quartz	idempotenti, rieseguibili, con log d'esito; `tenantProvider == null` nei job disattiva il filtro tenant: iterare i tenant esplicitamente	job che assumono un tenant corrente
Feature-first sul FE	cartelle per dominio (`features/timesheet/…`), non per tipo di file	`components/`, `hooks/`, `utils/` globali con centinaia di file
Server state ≠ UI state	TanStack Query per i dati del server, Zustand per lo stato di interfaccia	Redux/Context per cache di API

5.3 Multi-tenancy e sicurezza, non negoziabili

- Ogni entità con dati di un cliente implementa `IHasTenant`; il query filter è automatico.
- **Mai concedere un claim `.read` prima di aver scritto il perimetro** (`OnApplyFilter`): senza, il claim apre tutte le righe del tenant. Il perimetro va replicato in `OnSerialize` per coprire gli `?include`.
- Ogni controller non JSON:API dichiara `[ClaimEntity]`: il gate è fail-closed all'avvio.

- Gli attributi non-JSON:API di un'entità (`[SkipJwtAuthentication]`) valgono per **tutti** i verbi: usarli sul metodo, non sulla classe.
- Segreti (connection string, chiavi API, secret OAuth) **mai** nel repository, nemmeno in `SeedData`: variabili d'ambiente o Secret Kubernetes. È già successo e ha richiesto una rotazione.
- Password in transito e a riposo secondo i flussi già definiti (`/api/account/login`, MD5 hex sul wire per compatibilità storica, Identity per l'hash a riposo): non inventarne altri.

---

6. Convenzioni di nome

6.1 C#

Cosa	Convenzione	Esempio
Namespace	= cartella, prefisso `MIT.Fwk.`	`MIT.Fwk.Infrastructure.Entities.Project`
Classi, interfacce, enum, metodi, proprietà	`PascalCase`; interfacce con `I`	`ProjectTask`, `ITenantProvider`
Metodi asincroni	suffisso `Async`	`EnsureForEmployeeAsync`
Campi privati	`_camelCase`	`_context`, `_mapper`
Parametri e locali	`camelCase`	`tenantId`
Costanti	`PascalCase`	`MasterTenantFallback`
Un file, una classe	nome file = nome classe	`ProjectTask.cs`
Entità	**singolare**	`ErpEmployee`, non `ErpEmployees`
DTO	suffisso `DTO`	`ProductDTO`
Servizi dati auto-registrati	`I<Nome>ManualService` / `<Nome>ManualService`	`IOtherManualService`
Hook JSON:API	suffisso `ResourceDefinition` (o `SyncHook` se solo side-effect)	`ErpEmployeeResourceDefinition`
Controller	suffisso `Controller`; route in kebab-case	`TenantController` → `/api/tenants`
Job Quartz	suffisso `Job` o `Manager`	`MongoLogBusManager`
Seeder	suffisso `Seeder`, metodo `SeedAllTenantsAsync`	`LeaveSeeder`
Prefisso di modulo	le entità di un dominio condividono un prefisso	`Erp*`, `Project*`, `Leave*`, `Vehicle*`
Booleani	`Is*`, `Has*`, `Can*`, `Allow*`	`IsActive`, `HasUnverifiedPresence`
Commenti	in italiano, spiegano il **perché**; il _cosa_ lo dice il codice	vedi gli `<summary>` in `Warehouse.cs`

6.2 Database

Cosa	Convenzione	Esempio
Tabella	`PascalCase` **singolare**, = nome classe, via `[Table("…")]`	`ProjectTask`, `WorkflowState`

Cosa	Convenzione	Esempio
Tabella di verticale/cliente	prefisso `x_` (entità in `Entities/Custom`)	`x_Warehouse`, `x_SiteReport`
Tabelle Identity	`AspNet*`, non si toccano	`AspNetUsers`, `AspNetRoles`
Colonna	`PascalCase`, = nome proprietà	`EstimatedHours`
Chiave primaria	`Id`, `Guid`, `uniqueidentifier` con `DEFAULT NEWID()`	—
Chiave esterna	`<Entità>Id`, `Guid` o `Guid?` secondo la nullabilità	`ProjectId`, `ParentTaskId`, `AssigneeEmployeeId`
Tenant	`TenantId Guid?`	—
Timestamp	`CreatedAt`, `UpdatedAt`, `<Evento>At`, **UTC**	`UploadedAt`
Date senza ora	`DateOnly` / `date`	`DueDate` se è una scadenza di giorno
Booleani	`Is*/Has*`, `bit NOT NULL` con default	`IsBillable`
Stati e tipi	stringhe **minuscole**, un solo separatore per catalogo, valori nel contratto	`ok`, `at_risk`, `late`
Stringhe indicizzate	larghezza **dichiarata** (`IndexedStringWidths`)	mai `nvarchar(450)` implicito
Id di testo esterni	`string`, solo se assegnati da altri, elencati in `ExternalTextIds`	codice fiscale, tag NFC
Id storici migrati	`LegacyIdGuid.From(...)` (UUIDv5)	`lgc-<tipo>-<id>` → Guid
Migration	`PascalCase_ConSottolineature` descrittivo	`Bootstrap_FreshDatabase`, `Baseline_2026_09`
History table	una per DbContext	`__EFMigrationsHistory_<Context>`
Cancellazioni	FK verso i padri `NoAction`; cascade **applicativa** catalog-driven	`CascadeDeleteDependentsAsync`
Relazioni reciproche `[HasOne]`	disambiguazione esplicita `.WithMany()` in `OnModelCreating`	`ErpEmployee.BusinessGroup` ↔ `ErpBusinessGroup.TeamLeaderEmployee`
Portabilità SQL Server / MySQL	niente funzioni vendor nelle entità; script DBA in cartelle dichiarate	—

6.3 API JSON:API

Cosa	Convenzione	Esempio
Tipo di risorsa e route	**kebab-case plurale**	`project-tasks` → `/api/project-tasks`
Attributi e relationship	**camelCase**	`estimatedHours`, `assigneeEmployee`
Id nel JSON	Guid **minuscolo** (`ToString()` solo lato client, mai dentro una proiezione EF)	—
Campi scritti dal server	`AllowView	AllowFilter	AllowSort; AllowViewda sola solo per[NotMapped] calcolati	uploadedAt`	

Cosa	Convenzione	Esempio
Filtri	sintassi JSON:API (`filter=equals(project.id,'…')`), costanti Guid **tipizzate** nel codice	—
Include	to-one forzate dal framework (tranne `tenant`); entità con più di ~4 to-one disattivano il force-include; **mai** forzare una to-many	—
Endpoint custom	`/api/<kebab>`; azioni admin come `/…/seed`, policy `NeededRoleLevel0`	`/erp-roles/seed`
Errori	`422` + codice `UPPER_SNAKE_CASE`; `403` con codice per i muri (`MODULE_DISABLED`, `MASTER_TENANT_READ_ONLY`)	—
Contratto per il FE	`docs/contract/*.md` **e** Artifact, con riga «Versione leggibile» in testa; snapshot Swagger in `contract-snapshots`	—
JSON opaco in DTO non JSON:API	`JToken` (Newtonsoft), non `JsonNode`/`JsonElement`	`customPages[].config`

6.4 Frontend

Cosa	Convenzione	Esempio
Componenti	`PascalCase`, un file per componente	`TimesheetWeekGrid.tsx`
Hook	`useX`	`useResourceList`, `useCurrentUser`
Variabili, funzioni	`camelCase`	—
Costanti	`UPPER_SNAKE_CASE`	`MEDIA_CAT`
Tipi	`PascalCase`, generati dal contratto quando esistono	`ProjectTask`
File di route	file-based TanStack Router, kebab-case	`routes/settings/profile.tsx`
Cartelle	feature-first: `features/<dominio>/{api,componenents,hooks,pages}`; `shared/` per l'adapter JSON:API e il design system	—
Chiavi i18n	`dominio.schermata.chiave`, mai testo hard-coded	`timesheet.week.submit`
Stile	Tema Mantine per forma e tipografia, CSS variables per i colori (white-label); CSS modules dove serve	nessun valore letterale in `features/`; niente Tailwind come sistema primario

6.5 Git, versioni, configurazione

Cosa	Convenzione	Esempio
Commit	Conventional Commits, `tipo(scope): descrizione` in italiano	`feat(tasks): v1.200.0 - scadenze ereditate…`
Tipi	`feat`, `fix`, `docs`, `refactor`, `test`, `chore`, `perf`	—
Branch	`dev` integrazione, `main` rilascio, `feature/<slug>`, `hotfix/<slug>`	—
Versione	SemVer `vMAJOR.MINOR.PATCH`; il tag immagine di deploy avanza dal tag **in esecuzione**, non dall'ultimo commit	`v1.201.0`
Changelog	entità `Changelog` in-app (`status=released`) + `docs/CHANGELOG.md`; skill `/deploy-changelog`	—
Config	`customsettings.json` (impostazioni), `dbconnections.json` (connessioni, chiave = nome DbContext)	`"OtherDbContext": "Server=…"`
Segreti	fuori dal repo: env / Secret K8s	—
Documentazione	`docs/` con nome `MAIUSCOLO-KEBAB-vN.md`; un solo documento di mappa (`ARCHITETTURA-E-MODULI`), da aggiornare e non duplicare	—

7. Test unitari: obbligatori, non opzionali

Il framework nasce con una suite **d'integrazione** (Testcontainers, WebApi intera) che copre il contratto API. Non basta: ogni progetto, fork o nuovo, ha **anche una suite unitaria** che gira senza Docker, senza database e in meno di un minuto. I test unitari sono la specifica eseguibile del progetto, e sono la cosa che permette a chi arriva dopo, persona o Claude Code, di cambiare il codice senza paura.

7.1 Cosa va coperto

Va unit-testato	Non serve unit-testare
Regole di dominio: calcoli (ore, saldi, scadenze ereditate), transizioni di stato, validazioni di valore	il CRUD generato da JSON:API: lo coprono gli integration test
Servizi con logica: seeder, enricher, job Quartz, classificatori, mapper di import	il mapping EF e le migration: li copre lo schema generato dal modello nei test d'integrazione
Profili AutoMapper: `AssertConfigurationIsValid()` in un test	i DTO senza logica
Helper e parser: date, fusi, id legacy (`LegacyIdGuid`), naming	i controller «passacarte» che delegano a un servizio
Invarianti di convenzione: test che leggono il modello o il grafo delle risorse e falliscono se una regola è violata (nel framework: `AttributeCapabilitiesContractTests`, `ColumnWidthContractTests`,	—

Va unit-testato	Non serve unit-testare
`TenantLinkageExclusionTests`)	

7.2 Regole

- **Progetto separato**: `Tests/<Nome>.Tests.Unit` accanto a `Tests/<Nome>.Tests` (integrazione). Stack: xUnit, FluentAssertions, NSubstitute; `coverlet.collector` per la copertura.
- **Un test, un comportamento**, nome `Metodo_Condizione_Esito` (`EnsureForEmployee_TipoPerEvent_NonCreaBalance`).
- **Deterministici**: niente `DateTime.Now` nel codice testato, si inietta `TimeProvider`; niente rete, disco, DB.
- **Ogni bug corretto porta il test che lo riproduce**, scritto prima del fix e rosso prima del fix.
- **Le convenzioni si fissano con un test**, non con un paragrafo di documento: un test di invariante costa un'ora e non si dimentica, il paragrafo si dimentica alla prima scadenza.
- **Copertura**: nessun numero sacro, ma la logica di dominio sotto il 70% è un segnale da spiegare in review. La copertura del CRUD generato non conta.
- **Con Claude si lavora test-first**: prima il test che descrive il comportamento, poi la modifica. È il modo più economico di dare all'assistente un criterio di «fatto» verificabile invece di una descrizione.

7.3 Cosa insegna il framework

I difetti più costosi del framework sono stati tutti **silenziosi**: la scrittura riusciva e solo la rilettura falliva (`AttrCapabilities`), il seed era corretto ma non era mai girato, `FirstOrDefault` su un

`Guid` restituiva `Guid.Empty` e non `null`. Nessuno di questi si vede in un integration test che fa una POST e legge 201. Si vedono in un test unitario che chiede «cosa succede quando l'input è questo».

---

8. Codice documentato per essere mantenuto con Claude

Il codice di un progetto deve poter essere ripreso da una persona **o da Claude Code** senza chi lo ha scritto. Documentare, oggi, vuol dire **far vivere le regole nei posti che l'assistente legge**: se una regola sta solo nella testa di chi l'ha scritta, o in una chat chiusa, per il progetto non esiste.

8.1 Cosa deve esserci in ogni repository

Artefatto	Cosa contiene	Modello nel framework
`CLAUDE.md` alla radice	cos'è il progetto, come si builda/testa/deploya, le convenzioni **con il perché**, i gotcha già pagati. È il primo file che Claude legge a ogni sessione	`CLAUDE.md` del framework: ogni convenzione ha la storia del failure mode che l'ha generata
`CLAUDE.md` di cartella	le regole di un'area densa, vicino al codice, per non far crescere quello di radice all'infinito	`Src/MIT.Fwk.Infrastructure/Data/CLAUDE.md`, `Src/MIT.Fwk.WebApi/Services/CLAUDE.md`
`ONBOARDING.md`	quick start e mappa «cerchi X → vai in Y»	`ONBOARDING.md`
`docs/`	**un solo** documento di mappa, stato	`docs/ARCHITETTURA-E-MODULI-v2.md`,

Artefatto	Cosa contiene	Modello nel framework
	e piano (da aggiornare, mai duplicare); i contratti per il FE in `docs/contract`; le decisioni e le autorizzazioni in `docs/DECISIONI.md`	`docs/contract/`
`.claude/skills/<nome>/SKILL.md`	una **procedura ripetibile con passi non ovvi**, scritta una volta e rieseguibile da chiunque	`add-migration` (baseline sui DB esistenti), `run-tests` (Testcontainers o LocalDB), `deploy-changelog` (release + contratto FE)
`.claude/agents/<nome>.md`	un **ruolo specializzato** con strumenti limitati e istruzioni proprie	`fwk-explore`: esploratore read-only che risponde con `file:riga`
Commenti nel codice	il **perché** e i failure mode, non il cosa; il riferimento al paragrafo della direttiva che ha generato la regola	gli `<summary>` di `Warehouse.cs` e `ProjectTask.cs`
Test di invariante	la convenzione resa eseguibile (§7)	`AttributeCapabilitiesContractTests`

8.2 Quando si crea una skill, un agente, un documento

- **Skill**: quando una procedura è stata spiegata **due volte** in chat, o ha un passo che se saltato fa danno (la migration baseline, il drop del DB di test prima del run completo, il tag di deploy scelto dall'immagine in esecuzione). Una skill ha un nome imperativo, dice quando usarla nella `description`, e contiene i comandi esatti, non una descrizione dei comandi.
- **Agente**: quando serve un **contesto specializzato ricorrente** che non conviene ricaricare ogni volta: un esploratore read-only del codebase, un revisore dei perimetri multi-tenant, un verificatore del contratto FE. Un agente dichiara i suoi `tools` al minimo necessario: un revisore non scrive file.
- **Documento in `docs/`**: quando una decisione o un disegno deve sopravvivere al codice che lo implementa (una direttiva, un contratto, un piano). Se il contenuto è «come si fa X» è una skill, non un documento.
- **`CLAUDE.md`**: quando Claude sbaglia per una regola che **non era scritta**. La correzione data in chat e non scritta si perde alla sessione dopo; scritta in `CLAUDE.md` con il suo perché vale per tutti.

8.3 Usare quello che c'è, prima di crearne di nuovo

Un fork eredita `.claude/` del framework: le skill `add-migration`, `run-tests`, `deploy-changelog` e l'agente `fwk-explore` funzionano senza modifiche. A livello utente esistono `deploy-dev`, `deploy-prod` e `beep`. Prima di scrivere una skill o un agente si cerca lì e nella libreria aziendale; se esiste e manca qualcosa, si **estende** quella, non se ne crea una seconda con lo stesso scopo.

Per un progetto nuovo si copia lo scheletro: `CLAUDE.md` e `ONBOARDING.md` del framework ridotti a ciò che vale per il progetto, le skill `run-tests` e `add-migration` adattate ai nomi dei progetti, l'agente

`fwk-explore` rinominato sul nuovo codebase.

8.4 Cosa non fare

- ❌ Due documenti che raccontano la stessa mappa: divergono in silenzio e Claude ne legge uno a caso.
- ❌ Un `CLAUDE.md` di radice da migliaia di righe: si sposta per area nei `CLAUDE.md` di cartella.

- ❌ Regole tenute nella memoria personale dell'assistente o in chat: la memoria è di chi la ha scritta, il repo è di tutti.
- ❌ Commenti che ripetono il codice (`// incrementa il contatore`): il commento dice **perché** il contatore esiste e cosa succede se non viene incrementato.
- ❌ Una skill che descrive i comandi invece di contenerli: alla prima esecuzione qualcuno li reinventa.

---

9. Definition of done: cosa deve esserci prima di dire «finito»

1. **Build pulita**: zero nuovi warning; `CS0472`/`CS8073` (confronti impossibili su `Guid`) a zero.
2. **Test verdi**: la suite unitaria e quella d'integrazione girano (in `Debug`; in `Release` il gate licenza spegne l'app); i test nuovi creano i **propri** tenant e non scrivono nel master. Ogni regola di dominio nuova ha il suo test unitario, ogni bug corretto il test che lo riproduce (§7).
3. **Contratto verificato**: diff Swagger contro lo snapshot dell'ultimo deploy senza rinominazioni accidentali; contratto FE aggiornato se cambia la superficie.
4. **Sicurezza**: claim e perimetro per ogni nuova entità; `[ClaimEntity]` sui controller custom; nessun `Ok(entity)`; nessun segreto nel diff; `NuGetAudit` senza advisory nuovi.
5. **Interfaccia conforme a MaeDesign v3**: nessun colore letterale in `features/`, nessuna stringa nel JSX fuori da `t()`, empty state nelle quattro varianti (vuoto, filtrato, bloccato, errore), contrasto verificato. `npm run lint` a zero warning copre i primi due.
6. **Dati**: migration generata e **riletta** (EF accoppia drop+add come rename e sposta i dati nella colonna sbagliata); seed nel trigger giusto (§5.2) e **contato** sul DB target, non presunto.
7. **Osservabilità**: log con traceId sugli errori; job con esito loggato; nessun `Console.WriteLine`.
8. **Documentazione per la manutenzione** (§8): `CLAUDE.md` aggiornato con le convenzioni nuove e il loro perché; la procedura ripetibile introdotta è una skill in `.claude/skills`; il documento di mappa è aggiornato, non duplicato; changelog registrato.
9. **Deploy**: via skill, con versione scelta dal tag in esecuzione; verifica post-deploy con un bearer fresco; backup del DB prima delle migration di contratto.

---

10. Struttura minima di un repository

Per un fork del framework la struttura è quella del repo. Per un progetto nuovo:

```
<progetto>/
├── README.md               cos'è, come si avvia, come si deploya
├── CLAUDE.md               convenzioni e gotcha del progetto, con il perché (per persone e
Claude)
├── ONBOARDING.md           quick start e mappa «dove sta X»
├── .claude/
│   ├── skills/<nome>/SKILL.md   procedure ripetibili (run-tests, add-migration, deploy-
changelog…)
│   └── agents/<nome>.md         ruoli specializzati (esploratore read-only, revisore perimetri…)
├── Directory.Build.props   LangVersion, NuGetAudit su tutto l'albero
├── .editorconfig
├── Dockerfile
├── Src/
│   ├── <Nome>.Core/
│   ├── <Nome>.Infrastructure/
│   └── <Nome>.WebApi/       (o .Worker per servizi senza HTTP)
├── Tests/
```

```
│   ├── <Nome>.Tests.Unit/   xUnit + FluentAssertions + NSubstitute, senza Docker, < 1 minuto
│   └── <Nome>.Tests/        integrazione: WebApplicationFactory + Testcontainers
├── web/                     (se c'è un FE) Vite + React + Mantine, feature-first
├── docs/
│   ├── ARCHITETTURA-*.md    l'unico documento di mappa, stato e piano
│   ├── DECISIONI.md         decisioni e autorizzazioni (§11)
│   ├── contract/            contratto API per il FE
│   └── CHANGELOG.md
└── scripts/                 PowerShell di supporto (mai SQL con dati di produzione)
```

11. Come si chiede una variazione allo stack (R2)

Una pagina, inviata alla direzione tecnica **prima di iniziare**, con queste voci:

Voce	Cosa scrivere
Progetto e stima	nome, cliente, giornate approvate
Cosa si vuole usare	tecnologia, versione, licenza
Al posto di cosa	il pezzo dello stack standard che sostituisce
Perché lo standard non basta	il requisito concreto che lo stack non copre; «lo conosco meglio» non è un requisito
Costo di rientro	quanto costerebbe tornare allo standard fra un anno
Chi la mantiene	chi in azienda, oltre al proponente, sa metterci le mani
Impatto su esercizio	build, container, monitoraggio, backup

L'autorizzazione è scritta, va allegata al repository (`docs/DECISIONI.md`) e vale per **quel** progetto: non si estende ai successivi.

Le deroghe di disegno usano lo stesso canale. Uno scostamento da MaeDesign v3 — identità visiva di un cliente, struttura di pagina diversa da quella standard — si chiede con lo stesso modulo e si scrive nello stesso `docs/DECISIONI.md`. Cambia solo il firmatario: il **referente MaeDesign** autorizza lo strato struttura, la direzione tecnica autorizza lo stack. Due canali paralleli sarebbero il modo migliore per non far rispettare nessuno dei due.

Per le deroghe di disegno il modulo si riduce a uno screenshot e una riga di motivo: la decisione si prende guardando, non leggendo.

12. Checklist di avvio progetto

- [ ] Stima approvata e scritta; punto di partenza deciso secondo R3
- [ ] Confronto con il team framework fatto — **sempre, qualunque sia il conteggio** — ed esito scritto in `DECISIONI.md` con chi ha deciso, quando, e cosa riaprirebbe la scelta
- [ ] Nessuna variazione di stack, oppure autorizzazione scritta allegata (§11)
- [ ] Repository creato con la struttura del §10, `README` e `CLAUDE.md` iniziali
- [ ] Provider DB scelto (SQL Server / MySQL) e scritto in `customsettings.json`
- [ ] Connection string e segreti fuori dal repo
- [ ] Pipeline di deploy sul cluster dev funzionante alla prima settimana

- [ ] Contratto API iniziale in `docs/contract`, tipi TS generati
- [ ] Due suite di test che partono (anche vuote): unitaria senza Docker e d'integrazione con Testcontainers
- [ ] Convenzioni di nome del §6 lette dal team e riportate nel `CLAUDE.md` del progetto
- [ ] `DICHIARAZIONE.md` compilata: le dieci scelte di disegno che, se non decise all'avvio, ogni schermata prende per conto suo
- [ ] `.claude/` presente: skill `run-tests` e `add-migration` adattate, agente esploratore del codebase
- [ ] Primo `CLAUDE.md` con il **perché** di ogni regola già presa, non solo la regola

---

Appendice — riferimenti nel repository del framework

Documento	Cosa contiene
`CLAUDE.md` (radice)	regole del framework e convenzioni con la loro storia
`ONBOARDING.md`	quick start, mappa dei progetti, «dove sta X»
`docs/ARCHITETTURA-E-MODULI-v2.md`	il documento di mappa, stato e piano
`docs/NEW-FRONTEND-PLAN.md`	stack e architettura del frontend React/Mantine
`docs/MIGRATIONS-GUIDE.md`	migration EF per DbContext e ordine di applicazione (`DatabaseMigrationOrder`)
`Src/MIT.Fwk.Examples/`	modulo template (DbContext dedicato, entità, service) da cui copiare
`docs/ROLE-CLAIMS-CATALOG.md`	catalogo dei claim per entità
`docs/security/`	disegno della sicurezza e dei dati riservati
`Src/MIT.Fwk.Infrastructure/Data/CLAUDE.md`	livello dati: DbContext unico, filtri tenant, cascade
`Src/MIT.Fwk.WebApi/Services/CLAUDE.md`	ResourceDefinition e hook