Object Design Document

TECH4ALL

Riferimento	C06_ODD_1.5
Versione	1.5
Data	14/01/2025
Destinatario	Top Management
Presentato da	Marco Capuano, Giovanni Cerchia, Arcangelo Ciaramella, Silvana De Martino, Giovanni Esposito, Luigi Nasta, Giovanni Salsano, Giuseppe Staiano
Approvato da	

Revision History

Data	Versione	Cambiamenti	Autori
20/11/2024	0.1.1	Aggiunta component off-the-shelf.	Luigi Nasta, Arcangelo Ciaramella.
22/11/2024	0.1.3	Aggiunta Design Goals e Trade-offs.	Marco Capuano, Silvana De Martino.
22/11/2024	0.1.4	Modifica linee guida.	Luigi Nasta, Arcangelo Ciaramella.
22/11/2024	0.1.5	Aggiunta Design Pattern Proxy.	Silvana De Martino.
22/11/2024	0.1.6	Aggiunta Design Pattern Facade.	Giovanni Cerchia.
23/11/2024	1.1	Revisione Design Pattern: Facade.	Luigi Nasta, Arcangelo Ciaramella.
23/11/2024	1.2	Revisione Design Pattern: Proxy.	Giuseppe Staiano, Giovanni Salsano.
29/11/2024	1.3	Revisione ODD.	Tutti.
02/01/2025	1.4	Revisione ODD.	Tutti.
14/01/2024	1.5	Revisione finale dei contenuti e del formato.	Silvana De Martino, Arcangelo Ciaramella.

Team members

Nome	Ruolo nel progetto	Acronimo	Informazioni di contatto
Ferdinando Boccia	Project Manager	FB	f.boccia28@studenti.unisa.it
Domenico D'Antuono	Project Manager	DD	d.dantuono7@studenti.unisa.it
Silvana De Martino	Team Member	SDM	s.demartino30@studenti.unisa.it
Luigi Nasta	Team Member	LN	l.nasta4@studenti.unisa.it
Giovanni Salsano	Team Member	GSA	g.salsano14@studenti.unisa.it
Arcangelo Ciaramella	Team Member	AC	a.ciaramella7@studenti.unisa.it
Giovanni Esposito	Team Member	GE	g.esposito282@studenti.unisa.it
Giovanni Cerchia	Team Member	GC	g.cerchia6@studenti.unisa.it
Marco Capuano	Team Member	MC	m.capuano37@studenti.unisa.it
Giuseppe Staiano	Team Member	GS	g.staiano11@studenti.unisa.it

Sommario

1. Introduzione

Tech4All si propone di aiutare persone con analfabetismo digitale a riuscire a apprendere le metodologie di base per l'utilizzo di sistemi IT al fine di poter accedere a molte delle piattaforme governative, come quelle relative ad ASL e SPID, ma anche per permettergli di utilizzare in maniera più consapevole piattaforme legate allo svago, all'interazione e all'e-commerce.

In questa prima sezione del documento verranno descritti gli object design goal, i trade-offs, e le linee guida per la fase di implementazione, riguardanti la nomenclatura, la documentazione e la convenzione sui formati.

1.1 Object Design Goals

Rank	ID Design Goal	Descrizione	Origine
1	ODG_1 Manutenibilità	Il sistema dovrà garantire la presenza di codice leggibile, modulare, scalabile su un alto numero di task, e deve permettere l'integrazione di nuovo codice oltre che la gestione dei dati persistenti tramite un DBMS.	DG_M_1, DG_P_1, DG_M_2, DG_M_3
2	ODG_2 Robustezza	Il sistema deve garantire la protezione delle informazioni dell'utente, anche tramite la separazione dei permessi. In oltre il testing deve riguardare il 75% delle funzionalità dello stesso.	DG_D_1, DG_D_6
3	ODG_3 Gestione errori	Il sistema in caso di errore deve preservare lo stato e comunicare all'utente l'errore tramite interfacce apposite, che aiutino lo stesso a risolvere le problematicità.	DG_D_2, DG_D_3
4	ODG_4 Tempi di risposta	Il sistema dovrebbe fornire una risposta frontend in tempi non superiori ai 20 secondi nell'80% dei casi.	DG_P_3
5	ODG_5 costi	Il sistema avrà un costo iniziale stimato di 107.560€, in cui saranno compresi i costi di manutenzione ed assistenza, affidate al team di sviluppo stesso.	DG_C_1

1.2 Object Design Trade-Offs

Trade-off	Descrizione
Robustezza vs Costi	Per assicurarsi di rientrare nei costi, il team sarà, in fase di testing, a sacrificare alcuni casi di test, a costo di diminuire la robustezza del sistema.
Tempi di risposta vs Manutenibilità	Per garantire un elevato livello di manutenibilità, ed in particolare la persistenza dei dati in un DB relazionale con relativi controlli ed interrogazioni, il sistema potrebbe dilatare i tempi di risposta previsti. A fronte di tempi più lunghi assicurerà la persistenza dello stato del sistema e la corretta gestione dei dati persistenti.

1.3 Definizioni, Acronimi e Abbreviazioni

- ODG: Object Design Goals;
- DBMS: Relazionale (Relational Database Management System);
- COTS: Components Off-the-Shelf;
- HTTP: Hypertext Transfer Protocol.

Vengono riportati di seguito alcune definizioni presenti nel documento:

- Design Pattern: template di soluzioni a problemi ricorrenti impiegati per ottenere riuso e flessibilità.

1.4 Riferimenti

Libri e documentazioni:

- C06_RAD_2.1
- C06_SDD_1.1

1.5 Component Off-the-Shelf

Nell'implementazione del nostro sistema saranno utilizzati:

- Backend*:* Node.js insieme a Express, un framework per applicazioni web che facilita la costruzione del WebServer.
- Frontend*:* React, una libreria open-source di JavaScript per lo sviluppo dell'interfaccia utente.

Inoltre, verranno utilizzati alcuni *COTS (Component Off-the-Shelf)* per facilitare lo sviluppo e garantire efficienza e qualità:

- Database*:* I dati verranno memorizzati tramite MySQL, un DBMS relazionale ampiamente utilizzato e affidabile.
- Manutenibilità del codice*:* La leggibilità e la qualità del codice verranno garantite attraverso l'utilizzo di ESLint per analisi statica e buone pratiche di stile.
- Testing e integrazione*:* Utilizzeremo *GitHub Action* per il testing automatico e per il monitoraggio dell'integrazione continua (CI).

Questa combinazione di strumenti e tecnologie garantirà un'implementazione scalabile, manutenibile ed efficiente del sistema.

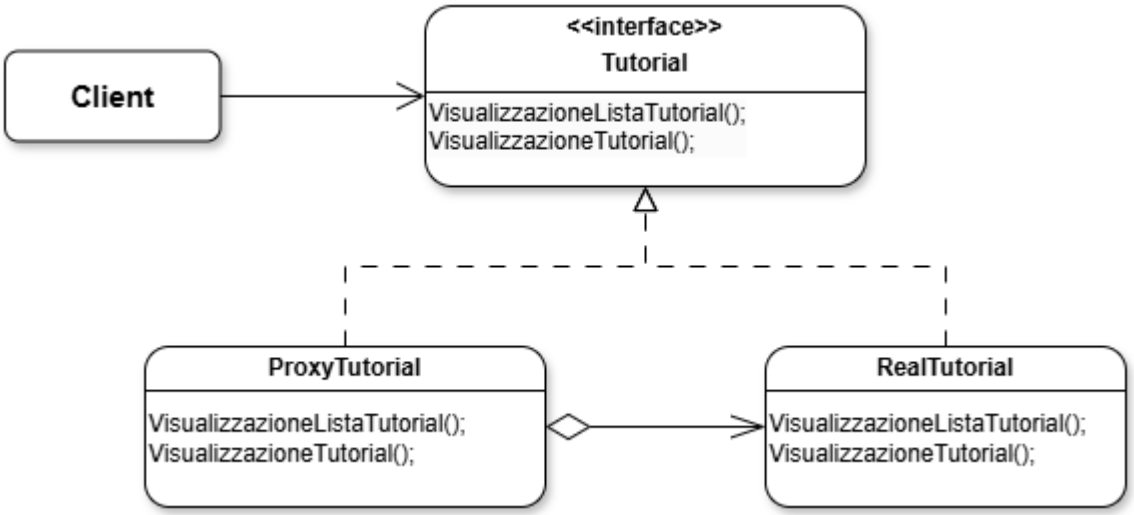
1.6 Design Patterns

1.6.1 Proxy

Per il nostro caso d'uso, caricare tutte le informazioni dettagliate sui tutorial al momento del caricamento della pagina sarebbe un'operazione dispendiosa in termini di memoria e risorse, contraria ai ODG relativi al risparmio di tempo e risorse definiti nella documentazione. L'adozione del Proxy consente di caricare inizialmente solo i dati necessari per visualizzare le informazioni dei tutorial .

Tramite l'impiego del Proxy, la visualizzazione delle icone presenti nell'area dedicata alla visualizzazione degli stessi richiederà il caricamento unicamente dell'immagine di copertina e del tutorial.

Quando un utente interagirà direttamente con un tutorial, ad esempio cliccando sulla sua immagine di copertina, il Proxy richiamerà l'oggetto "reale", accedendo in questo modo alle altre informazioni relative ad esso, come il testo, tramite l'impiego delle funzioni relative.



Vantaggi dell'implementazione:

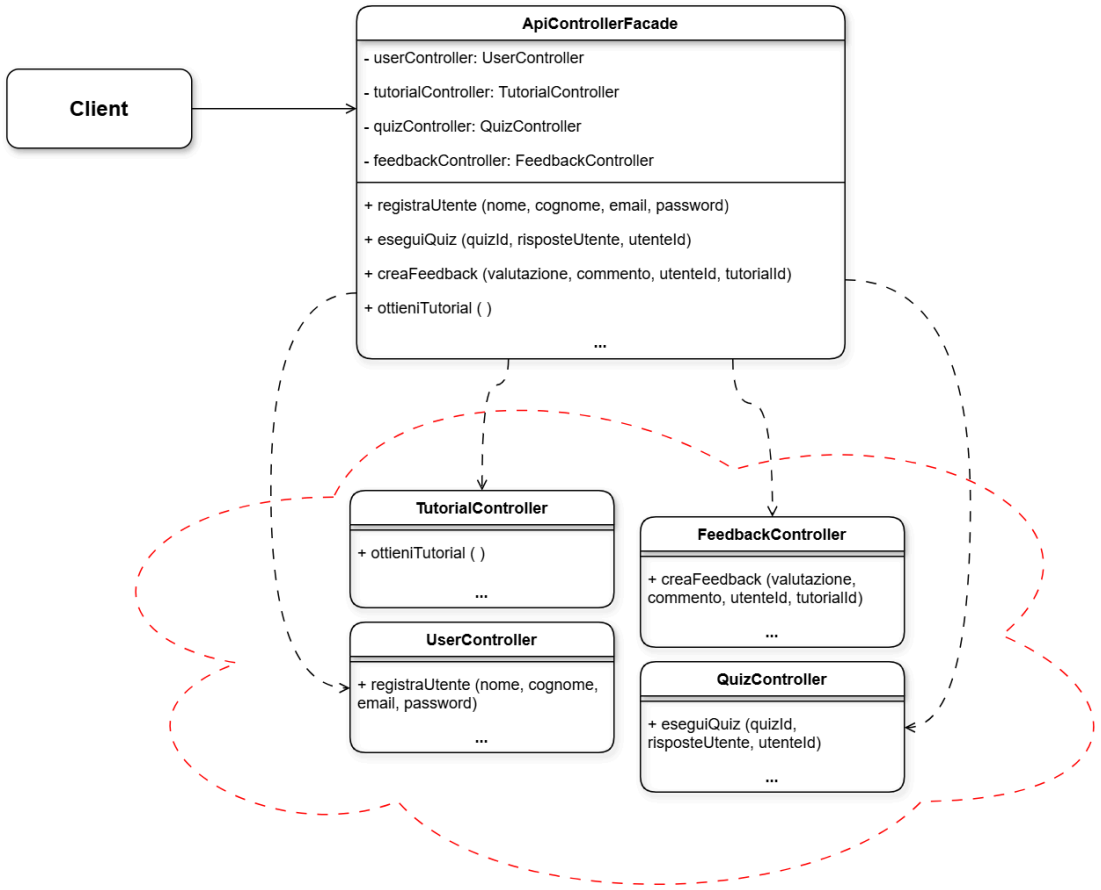
- Ottimizzazione delle risorse: Il caricamento iniziale della pagina per la visualizzazione dei tutorial è alleggerito, poiché vengono richiamati solo dati essenziali.
- Efficienza migliorata: Il sistema carica le informazioni dettagliate solo quando realmente necessarie, riducendo i tempi di attesa per l'utente.
- Scalabilità**:** Anche con un numero crescente di tutorial, il sistema mantiene buone prestazioni, caricando dati completi solo su richiesta.

Con questa implementazione, l'oggetto Proxy gestisce l'accesso all'oggetto "reale", riducendo il carico sulla memoria durante la visualizzazione della pagina iniziale. Questo approccio migliora l'esperienza utente e allinea il sistema agli obiettivi di design prefissati.

1.6.2 Facade

Il Facade è un design pattern strutturale che fornisce un'interfaccia semplificata a un sistema complesso o a un insieme di sottosistemi. Il suo scopo principale è ridurre la dipendenza e la complessità tra il client (chi utilizza il sistema) e le componenti interne del sistema stesso.

Nel sistema in questione il pattern Facade è stato implementato attraverso la creazione di una classe centrale chiamata ApiControllerFacade. Questa classe funge da punto di accesso unico per il client (il front-end), mascherando la complessità dei vari controller sottostanti e fornendo metodi semplificati per le operazioni di invio di richieste API al back-end. La classe ApiControllerFacade espone metodi come ottieniTutorial() e creaFeedback() direttamente al client, nascondendo i dettagli implementativi dei controller sottostanti.



Vantaggi dell'implementazione:

- Disaccoppiamento: il client non interagisce direttamente con i singoli controller, ma utilizza un'interfaccia unificata fornita dal Facade. Ciò riduce la dipendenza tra il front-end e il back-end.
- Flessibilità e manutenibilità: eventuali modifiche a un controller interno non richiedono cambiamenti nel client, purché l'interfaccia del Facade rimanga stabile.

- Semplificazione: il client ha accesso a metodi chiari e intuitivi, senza dover conoscere i dettagli interni della gestione delle API.

2. Glossario

Sigla/termine	Definizione
Facade	Un'interfaccia unificata che semplifica l'uso di un sistema complesso, offrendo un accesso più semplice e intuitivo alle sue funzionalità.
Proxy	Un pattern strutturale che fornisce un surrogato o un rappresentante di un oggetto per controllarne l'accesso. Fornisce un livello di astrazione che permette di aggiungere logica senza modificare direttamente l'oggetto reale.
Design Patterns	Schemi di progettazione che catturano le migliori pratiche e conoscenze acquisite nel tempo dalla comunità di sviluppatori.