Laurea Triennale in informatica-Università di Salerno
Corso di *Ingegneria del Software* -Prof.ssa F. Ferrucci e Prof. F. Palomba

Object Design Document

Riferimento	ODD
Versione	2.0.0
Data	11/12/2024
Destinatario	Prof.ssa Filomena Ferrucci, Prof.re Fabio Palomba
Presentato da	Team Members
Approvato da	Raffaella Spagnuolo, Alessia Ture

Revision History

Data	Versione	Descrizione	Autori
19/11/2024	0.1.0	Stesura Introduzione	BF, MM, GR, GB
19/11/2024	0.2.0	Stesura Object Design Goals	BF, MM, GR, GB
20/11/2024	0.3.0	Stesura trade-off	BF, MM, GR, GB
20/11/2024	0.4.0	Stesura Component off-the-shelf	BF, MM, GR, GB
21/11/2024	0.5.0	Stesura Pattern Design	Tutti i TMs
21/11/2024	0.6.0	Stesura Glossario	BF, MM, GR, GB
21/11/2024	0.7.0	Aggiunta diagrammi PD	BF, MM, GR, GB
22/11/2024	1.0.0	Prima revisione	FR, MR, ATg, ATr
11/12/2024	2.0.0	Revisione PMs	RS

Project Managers

Nome	Acronimo	Contatto
Raffaella Spagnuolo	RS	r.spagnuolo6@studenti.unisa.it
Alessia Ture	ATu	a.ture@studenti.unisa.it

Team Members

Nome	Acronimo	Contatto
Giovanni Balzano	GB	g.balzano10@studenti.unisa.it
Benito Farina	BF	b.farina5@studenti.unisa.it
Marco Meglio	MM	m.meglio4@studenti.unisa.it
Ferdinando Ranieri	FR	f.ranieri12@studenti.unisa.it
Marco Renella	MR	m.renella1@studenti.unisa.it
Giuseppe Russo	GR	g.russo248@studenti.unisa.it
Anna Tagliamonte	ATg	a.tagliamonte9@studenti.unisa.it
Alessandra Trotta	ATr	a.trotta56@studenti.unisa.it

Sommario

1 Object Design Goals, Trade-Off e Linee Guida

1.1 Introduzione

ZeroWaste Home si propone di semplificare la gestione delle risorse alimentari domestiche, aiutando gli utenti a ridurre gli sprechi alimentari e promuovendo uno stile di vita più sostenibile. L'obiettivo principale è supportare famiglie e individui nella pianificazione e ottimizzazione dei consumi alimentari, fornendo strumenti innovativi come il monitoraggio delle scadenze e l'integrazione con una community digitale.

In questa prima sezione del documento verranno descritti gli **Object Design Goals (ODG)**, i **Trade-Off** e le **Linee Guida per la fase di implementazione**. Gli ODG rappresentano i principi chiave che guidano la progettazione del sistema, con un'attenzione particolare a usabilità, affidabilità, prestazioni e manutenibilità. I Trade-Off, invece, illustrano i compromessi necessari per bilanciare obiettivi di design spesso contrastanti, come la velocità di risposta e la robustezza del sistema.

Infine, verranno presentate le **Linee Guida di Implementazione**, riguardanti la nomenclatura, la documentazione e le convenzioni sui formati. Questi standard assicurano uniformità e coerenza nella scrittura del codice, migliorando la collaborazione tra i membri del team e facilitando la manutenzione futura del sistema.

1.2 Object Design Goals

Rank	ID Design Goals	Descrizione	Origine
1	**ODG_01: Alta Manutenibilità**	Il sistema deve adottare principi di progettazione a oggetti con struttura modulare, alta coesione e basso accoppiamento, garantendo leggibilità del codice, commenti chiari e documentazione completa.	DG_10
2	**ODG_02: Sicurezza dei dati**	I dati sensibili degli utenti devono essere gestiti solo tramite metodi pubblici (getter e setter) che garantiscono una corretta manipolazione dei dati (es. crittografia SHA 512 su dati sensibili).	DG_03
3	**ODG_03: Performance controllo Input**	Il sistema deve garantire la validazione degli input e l'aggiornamento dei dati validati entro 1 secondo.	DG_4
4	**ODG_04: Performance con carico elevato**	Il sistema deve supportare almeno 500 utenti simultanei, garantendo scalabilità e tempi di risposta di 2 secondi per operazioni generali e mantenere tempi di caricamento sotto i 3 secondi, con risposte inferiori a 1 secondo per modifiche alla lista della spesa.	DG_5, DG_6

1.3 Trade-Offs

Trade-Off	Descrizione
Alta Manutenibilità Vs Performance con carico elevato	Il trade-off tra **alta manutenibilità (ODG_01)** e **performance con carico elevato (ODG_04)** deriva dal fatto che la modularità e le astrazioni, utili per la manutenzione, aumentano la complessità esecutiva, mentre le ottimizzazioni per la performance possono sacrificare modularità e leggibilità, complicando la manutenzione. Preferiamo dare precedenza all'**alta manutenibilità**.
Sicurezza dei dati Vs Performance controllo input	Il trade-off tra **sicurezza dei dati (ODG_02)** e **performance controllo input (ODG_03)** deriva dall'overhead computazionale della crittografia, che può rallentare l'aggiornamento dei dati entro 1 secondo. Preferiamo dare precedenza alla **sicurezza dei dati.**

1.4 Definizioni, acronimi e abbreviazioni

Definizioni

Questo Glossario fornisce una raccolta di termini tecnici e concetti chiave utilizzati nel progetto.

Acronimi

- **ODD**: Object Design Document
- **ODG**: Object Design Goals
- **COTS**: Component Off-the-Shelf
- **SHA**: Secure Hash Algorithm
- **RNF**: Requisiti Non Funzionali
- **DP**: Pattern Design
- **API**: Application Programming Interface
- **UML**: Unified Modeling Language
- **JSON**: JavaScript Object Notation

Abbreviazioni

- **FAC**: Facade

1.5 Riferimenti

- Requirements Analysis Document, versione 2.0.0
- System Design Document, versione 2.0.0
- Object-Oriented Software Engineering (Using UML, Patterns, and Java) Third Edition - Bernd Bruegge & Allen H. Dutoit

1.6 Component Off-the-Shelf

I COTS che verranno usati per il **front-end** sono:

- **Angular:** Framework open-source per lo sviluppo del front-end, utilizzato per creare interfacce utente moderne e dinamiche grazie alla sua architettura modulare e al supporto integrato per il data binding;
    - **Versione 18.2.11**: Questa versione è stata scelta per la sua stabilità e il supporto continuo da parte della comunità. La versione più recente di Angular (18.x) offre miglioramenti nelle performance e supporto a nuove funzionalità di sviluppo.
- **PrimeNG:** Libreria di componenti UI basata su Angular, scelta per semplificare lo sviluppo delle interfacce grafiche fornendo elementi predefiniti e altamente personalizzabili, migliorando coerenza e velocità;
    - **Versione 17.18.11:** La versione scelta è compatibile con *Angular 18.2.11* e offre un'ampia gamma di componenti UI personalizzabili, migliorando la velocità di sviluppo e la coerenza visiva dell'applicazione.
- **Figma:** Strumento per la progettazione e la prototipazione delle interfacce utente, utile per creare layout condivisibili e collaborare in tempo reale durante le fasi di design;
    - **Versione:** Non essendo un'applicazione con versioni tradizionali, Figma si aggiorna automaticamente, garantendo sempre l'accesso alle nuove funzionalità per la progettazione e prototipazione in tempo reale.

I **COTS** che verranno usati per il **back-end** sono:

- **Spring Framework:** Framework back-end open-source utilizzato per gestire la logica di business e l'accesso ai dati, offrendo strumenti avanzati per creare sistemi scalabili e di facile manutenzione;
    - **Versione 3.3.5:** La versione di Spring Boot 3.3.5 è scelta per la sua stabilità e il supporto a Java 23, che consente di utilizzare le ultime versioni del linguaggio Java. È una versione ampiamente testata e supporta strumenti come Spring Data JPA, essenziale per la gestione dell'accesso ai dati.

- **MySQL Workbench:** Software per la gestione del database utilizzato per progettare, amministrare e monitorare il modello dei dati di sistema in modo efficiente;
    - **Versione 8.0:** La versione 8.0 è scelta per la sua robustezza nelle operazioni di amministrazione e progettazione del database, offrendo funzionalità avanzate per la gestione di database MySQL in modo efficiente.
- **API Open Food Facts:** Interfaccia che consente di recuperare informazioni sui prodotti alimentari tramite i codici a barre;
    - **Versione:** Poiché non esiste una versione formale dell'API, si fa riferimento all'endpoint stabile che offre l'accesso ai dati relativi ai prodotti alimentari tramite codici a barre.

Gli **Strumenti di Supporto** che verranno usati sono:

- **Prettier:** Strumento per la formattazione automatica del codice, utilizzato per garantire un codice leggibile e uniforme, semplificando le revisioni e riducendo gli errori;

    - **Versione 3.3.3:** La versione di Prettier scelta è la 3.3.3, utilizzata per garantire una formattazione automatica e uniforme del codice, migliorando la leggibilità e la manutenzione del codice.

1.7 Design Patterns

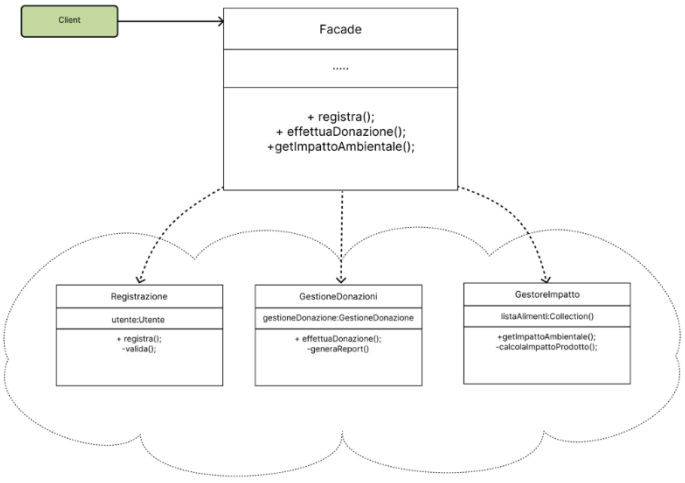
1.7.1 Facade

Il **Facade Design Pattern** 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.

- **Obiettivo**:
  - Fornisce un punto di accesso unico che nasconde la complessità dei sottosistemi sottostanti e riduce l'accoppiamento;
  - Questo tipo di design pattern rende il sistema chiuso, che per natura è più manutenibile a discapito dell'efficienza, siccome la comunicazione deve passare attraverso un layer aggiuntivo;
  - Protegge il client e i sottosistemi dai cambiamenti interni. Eventuali modifiche ad un sottosistema non influenzano direttamente gli altri se l'interfaccia del Facade rimane stabile;
- **Implementazione nel progetto**:

  L'utilizzo del design pattern permetterà il disaccoppiamento dei client dai dettagli implementativi del sottosistema, riducendo la dipendenza tra le classi, permettendo flessibilità e manutenibilità. Nel caso di **ZeroWaste Home**, verrà utilizzato per interfacciare il Service Layer utilizzato nell'architettura con l'Application Layer e per interfacciare l'Application Layer con il livello superiore;

1.7.1.1 Diagramma

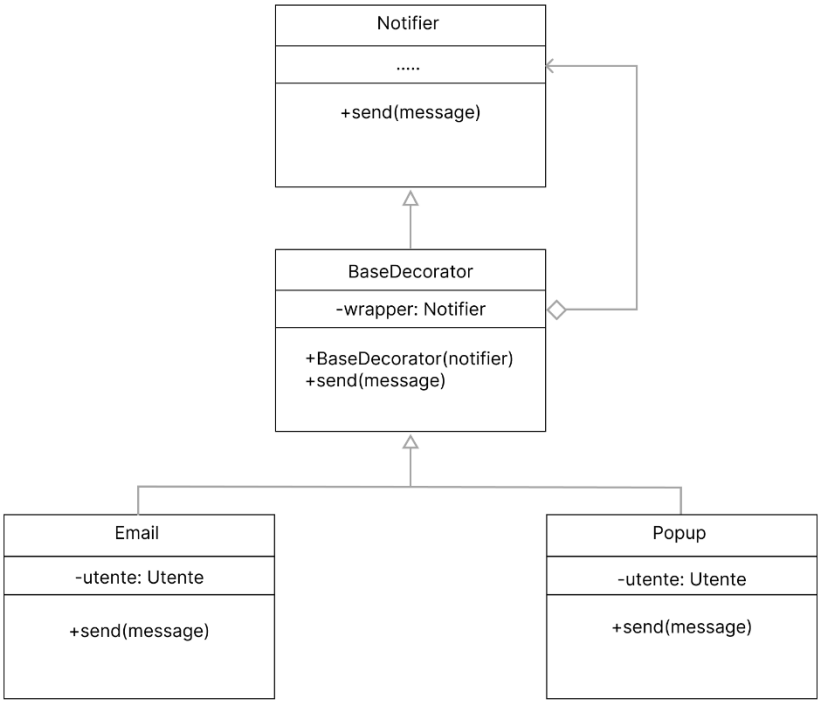


1.7.2 Decorator

Decorator è un modello di progettazione strutturale che consente di associare nuovi comportamenti agli oggetti inserendoli all'interno di speciali oggetti **wrapper** che contengono i comportamenti.

- **Obiettivo:**
  - Permette di aggiungere nuove funzionalità a un oggetto in modo dinamico, senza alterare il suo codice originale o quello delle sue classi;
  - Il Decorator consente di combinare comportamenti in modo modulare, migliorando la manutenibilità del codice e riducendo la proliferazione di sottoclassi;
  - Questo pattern è particolarmente utile in sistemi dove i comportamenti di un oggetto possono cambiare o dove è necessario combinare funzionalità diverse in modo flessibile;
- **Implementazione nel progetto**:
  - Nel contesto di *ZeroWaste Hom*e, il Decorator sarà utilizzato per arricchire dinamicamente gli oggetti relativi alle notifiche degli utenti;
  - Gli utenti possono scegliere personalizzazioni sulle funzionalità riguardanti le notifiche (es. popup, e-mail);

1.7.2.1 Diagramma



1.8 Linee guida per la documentazione delle interfacce

Questa sezione contiene una lista di regole da seguire per la progettazione e documentazione delle interfacce nel progetto. Gli sviluppatori devono rispettare queste linee guida per garantire uniformità e chiarezza.

- **Nomenclatura uniforme**:
  - Utilizzare nomi descrittivi e coerenti per le interfacce e i loro metodi, rispettando il formato verboOggetto (es. getProductData);
  - Evitare abbreviazioni ambigue e assicurarsi che i nomi siano significativi e intuitivi.
- **Commenti esaustivi**:
  - Ogni metodo pubblico deve includere un commento che:
    - Descriva chiaramente la funzionalità del metodo;
    - Specifichi il ruolo di ogni parametro;
    - Indichi il tipo e il significato del valore restituito;
    - Elenchi eventuali eccezioni lanciate.
- **Specifiche degli input e output**:
  - Definire chiaramente:
    - I tipi di dati accettati (es. stringhe, numeri, JSON);
    - I formati richiesti (es. lunghezza massima, pattern accettati);
    - I dati restituiti e i loro tipi (es. oggetti, collezioni, status code);
  - Documentare eventuali vincoli sugli input (es. valori obbligatori, intervalli validi).
- **Eccezioni e gestione degli errori**:
  - Specificare quali eccezioni possono essere sollevate dai metodi.
  - Documentare come gestire gli errori più comuni e i messaggi associati.
- **Utilizzo di diagrammi UML**:
  - Fornire diagrammi UML per rappresentare:
    - Le relazioni tra le interfacce e i loro componenti.

2 Glossario

A.

Angular: Framework open-source per lo sviluppo di applicazioni web dinamiche.

API Open Food Facts: Interfaccia per il recupero di informazioni sui prodotti alimentari tramite codici a barre.

B.

Back-end: Parte del sistema responsabile della logica di business, accesso ai dati e interazione con il database.

C.

COTS (Component Off-the-Shelf): Componenti software o hardware di terze parti utilizzati per semplificare lo sviluppo.

Community: Sezione del sistema che permette agli utenti di interagire, condividere risorse e segnalare problemi.

D.

Design Patterns: Soluzioni progettuali riutilizzabili a problemi ricorrenti nello sviluppo software, pensate per migliorare qualità, manutenibilità e riusabilità del codice.

Decorator: Design Pattern strutturale che permette di aggiungere dinamicamente comportamenti o responsabilità ad un oggetto, senza alterarne la classe.

F.

Facade: Design Pattern strutturale che fornisce un'interfaccia semplificata e unificata per accedere a un sistema complesso o a un insieme di sottosistemi.

Figma: Piattaforma collaborativa basata su cloud per il design di interfacce utente, prototipazione e collaborazione in tempo reale.

Front-End: Parte di un'applicazione web o software con cui l'utente interagisce direttamente, comprendente l'interfaccia utente (UI) e la logica di interazione.

O.

Object Design Goals: Principi e linee guida che orientano la progettazione di oggetti in un sistema software, mirati a migliorare la modularità, la riusabilità, la manutenibilità e la coesione del codice.

P.

PrimeNG: Libreria di componenti UI per Angular che offre un set completo di elementi pronti all'uso per creare interfacce web moderne e reattive.

Prettier: Strumento di formattazione del codice che applica uno stile coerente a diversi linguaggi di programmazione, migliorando leggibilità e uniformità.

S.

Spring: Framework open-source per lo sviluppo di applicazioni Java, progettato per semplificare la gestione delle dipendenze e fornire strumenti per creare applicazioni robuste e scalabili.