

WeCook
Object Design Document
Versione 1.1



WeCook

Data: 05/02/2025

Progetto: WeCook	Versione: 1.1
Documento: Object Design Document	Data: 05/02/2025

Coordinatore del progetto:

Nome	Matricola

Partecipanti:

Nome	Matricola
Simone Francesco Albino	0512116140
Valentina Ferrentino	0512116842
Giovanni Semioli	0512117412

Scritto da:	Simone Francesco Albino - Valentina Ferrentino - Giovanni Semioli
--------------------	---

Revision History

Data	Versione	Descrizione	Autore
15/12/2024	1.0	Prima stesura	Team
05/02/2025	1.1	Aggiunta dei Design Pattern e della divisione in pacchetti	Team

Progetto: WeCook	Versione: 1.1
Documento: Object Design Document	Data: 05/02/2025

Indice

1. Introduzione	3
1.1 Object design goals	3
1.2 Interface documentation guidelines	4
1.3 Riferimenti	4
2. Packages	5
3. Interfacce delle classi	6
4. Design Pattern	8
Adapter	8
Singleton	8

Progetto: WeCook	Versione: 1.1
Documento: Object Design Document	Data: 05/02/2025

1. Introduzione

We Cook nasce con l'idea di connettere persone di tutto il mondo che condividono una passione comune: la cucina. La piattaforma ha l'obiettivo di mettere in contatto tradizioni culinarie, sapori e storie personali, abbattendo le barriere culturali e geografiche, offrendo uno spazio dove chiunque possa condividere le proprie radici, la propria idea di cucina, in modo da permettere ad ogni ingrediente di essere valorizzato al meglio.

1.1 Object design goals

Affidabilità: Il sistema deve garantire stabilità operativa, protezione dei dati e gestione sicura di eventuali errori, assicurando il corretto funzionamento anche in condizioni di input non validi o inattesi.

Efficienza: La progettazione deve ottimizzare il sistema per tempi di risposta rapidi, garantendo che le operazioni come il caricamento delle pagine, delle immagini e dei contenuti avvengano entro limiti prestabiliti, minimizzando l'uso delle risorse.

User-Friendliness: Il design deve essere intuitivo e orientato all'utente, con notifiche chiare per fornire un feedback immediato e utile, contribuendo a migliorare l'esperienza e la comprensione del sistema.

Robustezza: La progettazione deve prevedere soluzioni che garantiscano una gestione solida degli errori e un comportamento consistente anche in caso di malfunzionamenti o input non previsti.

Flessibilità: Il sistema deve essere progettato per adattarsi facilmente a nuove esigenze o aggiornamenti futuri, riducendo al minimo gli sforzi necessari per introdurre miglioramenti o estensioni.

Riusabilità: Il sistema deve essere progettato con componenti modulari e generici, come notifiche, layout responsive, logging e validazione centralizzati, che possano essere facilmente riutilizzati in altre parti del sistema.

Progetto: WeCook	Versione: 1.1
Documento: Object Design Document	Data: 05/02/2025

1.2 Interface documentation guidelines

Questo sottoparagrafo sarà completato con la stesura del codice.

1.3 Riferimenti

Come ispirazione per la realizzazione di WeCook sono stati presi come riferimento diverse piattaforme, analizzandone il design e le principali funzioni social, come il feed delle notizie, la gestione dei profili utente e le interazioni tra utenti (Instagram, Facebook, ecc).

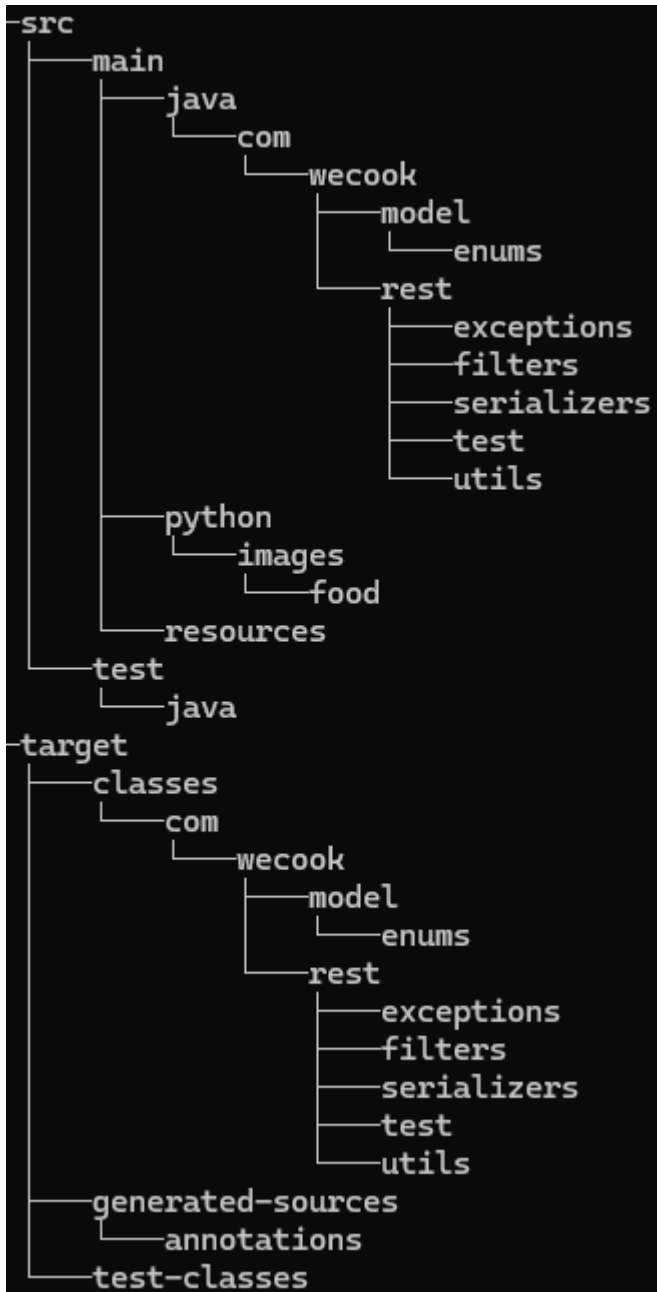
Il progetto si basa, inoltre, su metodologie descritte nei testi di elencati di seguito:

- Bruegge, Bernd, e Dutoit, Allen H., Object-Oriented Software Engineering Using UML, Patterns, and Java™ (Third Edition), Pearson;
- Requirements Analysis Document;
- System Design Document.

Progetto: WeCook	Versione: 1.1
Documento: Object Design Document	Data: 05/02/2025

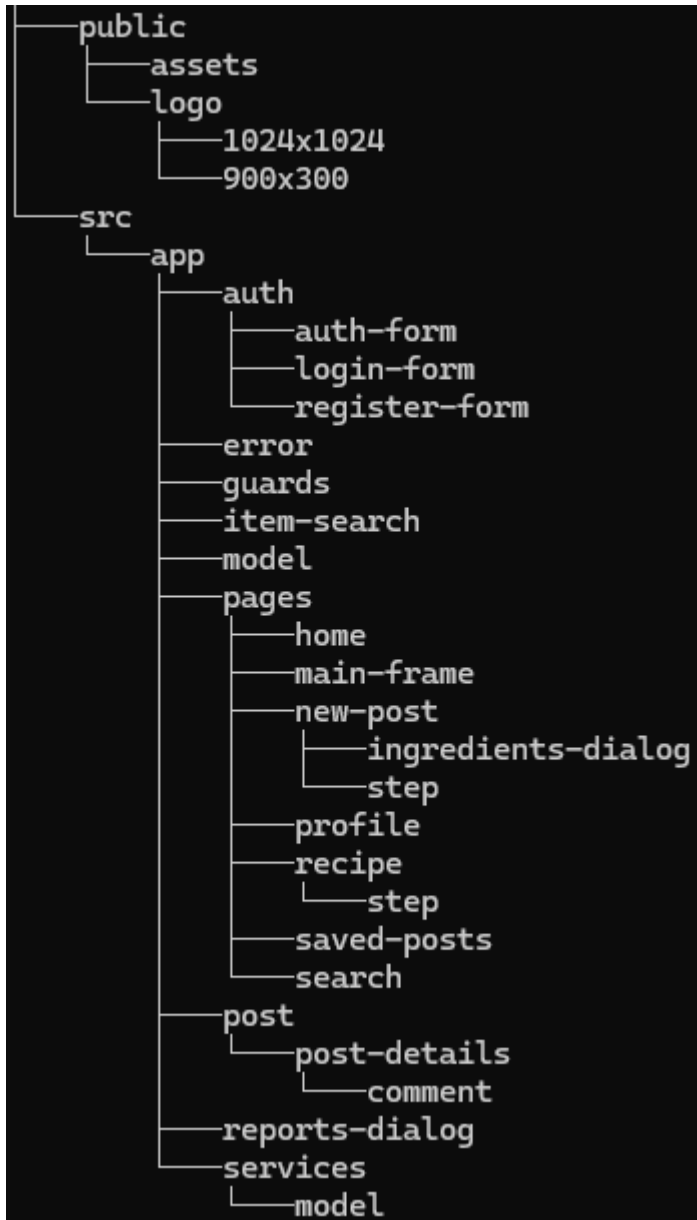
2. Packages

I package generati dal server sono i seguenti



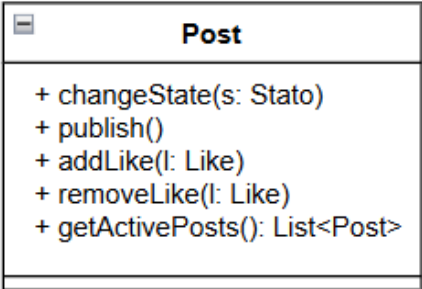
Progetto: WeCook	Versione: 1.1
Documento: Object Design Document	Data: 05/02/2025

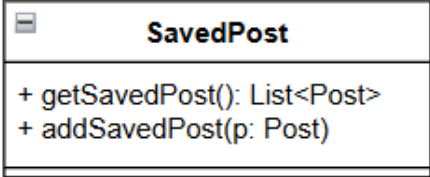
I package generati dal web sono i seguenti

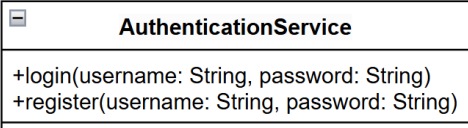


Progetto: WeCook	Versione: 1.1
Documento: Object Design Document	Data: 05/02/2025

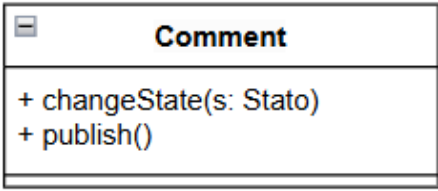
3. Interfacce delle classi

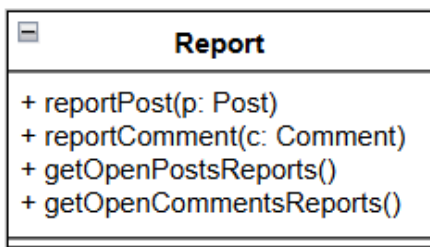
context PostService::changeStato(s) post: getStato() = s context PostService::publish() context PostService::addLike(l) pre: not contains(l) post: contains(l) context PostService::removeLike(l) pre: contains(l) post: not contains(l) context PostService::getActivePosts()	 <pre> classDiagram class Post { +changeState(s: Stato) +publish() +addLike(l: Like) +removeLike(l: Like) +getActivePosts(): List<Post> } </pre>
---	--

context SavedPostService::getSavedPost() context SavedPostService::addSavedPost(p) pre: not contains(p) post: contains(p)	 <pre> classDiagram class SavedPost { +getSavedPost(): List<Post> +addSavedPost(p: Post) } </pre>
--	---

context AuthenticationService::login(username, pwd) pre: contains(username) context AuthenticationService::register(username, pwd) pre: not contains(username) post: contains(username)	 <pre> classDiagram class AuthenticationService { +login(username: String, password: String) +register(username: String, password: String) } </pre>
--	---

Progetto: WeCook	Versione: 1.1
Documento: Object Design Document	Data: 05/02/2025

context CommentService::changeState(s) post: getStato() = s context CommentService::publish()	 <pre> classDiagram class Comment { +changeState(s: Stato) +publish() } </pre>
--	--

context ReportService::repostPost(p) post: p.getState = "active" context ReportService::repostComment(c) post: c.getState = "active" context ReportService::getOpenPostsReports() context ReportService::getOpenCommentsReports()	 <pre> classDiagram class Report { +reportPost(p: Post) +reportComment(c: Comment) +getOpenPostsReports() +getOpenCommentsReports() } </pre>
--	--

context RoleService::hasRole(u) pre: contains(u) post: getUtente() = u	 <pre> classDiagram class RoleService { +hasRole(u: Utente) } </pre>
---	--

Progetto: WeCook	Versione: 1.1
Documento: Object Design Document	Data: 05/02/2025

4. Design Pattern

Adapter

L'**Adapter Pattern** è un design pattern strutturale che permette a interfacce incompatibili di lavorare insieme, fungendo da "adattatore" tra due sistemi. Viene utilizzato quando si vuole far interagire classi con interfacce diverse senza modificarne il codice originale; esistono due tipi principali di adapter, il class adapter e l'object adapter, il primo utilizza l'ereditarietà multipla per adattare un'interfaccia ad un'altra, il secondo utilizza l'ereditarietà singola e la delegazione.

Nel caso di WeCook l'Adapter Pattern è stato utilizzato nelle classi:

- PostSerializer;
- RecipeSerializer;
- StepSerializer;
- RecipeIngredientSerializer;
- CommentSerializer;
- StandardUserSerializer.

Queste classi sono state implementate per poterle adattare al formato JSON che viene restituito all'interfaccia del Client.

Singleton

Il **Singleton** è un design pattern creazionale che garantisce che di una classe possa esistere una sola istanza a livello globale e fornisce un punto di accesso centralizzato a tale istanza. È utile in tutti gli scenari in cui è necessario controllare l'accesso ad una risorsa condivisa. Nel caso di WeCook, il pattern singleton è stato utilizzato:

- Per gestire l'accesso al JWTManager che gestisce l'autenticazione degli utenti mediante token JWT
- Per gestire la classe wrapper CustomGson che permette di accedere in modo veloce ad una istanza di un oggetto Gson configurato ad hoc per il sistema senza doverlo riconfigurare ogni volta che lo si vuole usare.