Laurea in Informatica - Università degli Studi di Salerno Corso di Ingegneria del Software - Prof. A. De Lucia

Data: 08/10/2024

TEAM
Gianluca Fusco 0512116485
Antonio Giorgio 0512106036
Cristian Di Popolo 0512105370

Revision History

Data	Versione	Descrizione	Autore
13/12/2024	1.0	Introduzione e packages	G. Fusco A. Giorgio C. Di Popolo
16/12/2024	1.1	Interfaccia delle classi	G. Fusco A. Giorgio C. Di Popolo

Indice

1 INTRODUZIONE

1.1 Object Design Trade-Offs

Comprensibilità vs Tempo:

Data la scarsa disponibilità di tempo, il codice non verrà scritto dando priorità alla comprensibilità ma verrà scritto in funzione del tempo del relativo task.

Riusabilità vs Costi:

Dato il budget designato per il progetto, pari a zero. Non saranno integrati sistemi che prevedano un costo di utilizzo.

Sicurezza vs Efficienza:

Dal punto di vista della sicurezza, abbiamo implementato un sistema di autenticazione basato su username e password crittografata con algoritmi sicuri come BCrypt, evitando l'archiviazione in chiaro. Inoltre, il controllo degli accessi è regolato da ruoli, garantendo che ogni utente acceda solo alle risorse autorizzate. Dal punto di vista dell'efficienza, queste misure essenziali bilanciano protezione dei dati e rispetto delle scadenze, offrendo una base sicura e migliorabile per il progetto.

Trasparenza vs Efficienza:

Per la gestione dei dati, abbiamo scelto MySQL come sistema di database, privilegiando la trasparenza e l'affidabilità nelle operazioni. MySQL offre un DBMS che garantisce consistenza, semplificando l'interazione con i dati e riducendo il rischio di errori nella persistenza. Questo approccio, sebbene comporti un lieve sacrificio in termini di efficienza rispetto a soluzioni meno strutturate, ci consente di mantenere un sistema comprensibile, affidabile e facile da mantenere, in linea con le risorse disponibili e gli obiettivi del progetto.

1.2 Linee guida per l'interfaccia

- **Classi e Interfacce**

Nomi delle Classi e Interfacce:
Devono seguire la convenzione UpperCamelCase.
I nomi devono descrivere chiaramente il ruolo della classe o interfaccia (ad esempio OrderManager, ProductService).

- **Metodi**

Nomi dei Metodi:
Devono seguire la convenzione lowerCamelCase.
I nomi devono descrivere la funzionalità del metodo e includere un verbo (es.: addProduct, updateOrderStatus).
Metodi getter e setter devono seguire la regola di nominazione get[nomeAttributo] e set[nomeAttributo].

- **Variabili**

Nomi delle Variabili:
Devono seguire la convenzione lowerCamelCase.
Le variabili di istanza devono essere private.
Le variabili locali possono essere dichiarate insieme, ma ogni variabile d'istanza deve essere dichiarata singolarmente.

- **Convenzioni di Gestione degli Errori**

Le condizioni di errore devono lanciare eccezioni invece di restituire valori di errore.
Utilizzare eccezioni specifiche e personalizzate per l'applicazione.

- **Altre Linee Guida**

Indentazione del codice:
Utilizzare 4 spazi o una tabulazione per l'indentazione.

1.3 Definizioni, acronimi e abbreviazioni

APP: Applicazione di acquisto online per prodotti alimentari.

ADM
Amministratore: utente responsabile della gestione della piattaforma e del catalogo prodotti.

USR
Utente generico: qualsiasi utilizzatore della piattaforma.

CUS
Cliente: utente loggato che ha la possibilità di acquistare prodotti.

PROD
Prodotto: dati e informazioni relative a un prodotto alimentare disponibile per l'acquisto.

CAT
Catalogo: l'elenco aggiornato di prodotti disponibili sulla piattaforma.

CAR
Carrello: spazio dove l'utente aggiunge i prodotti prima dell'acquisto.

API
Application Programming Interface: interfaccia per l'interazione tra sistemi diversi.

UI
User Interface: interfaccia utente che permette l'interazione con l'applicazione.

JSON
JavaScript Object Notation: formato di dati per lo scambio di informazioni tra client e server.

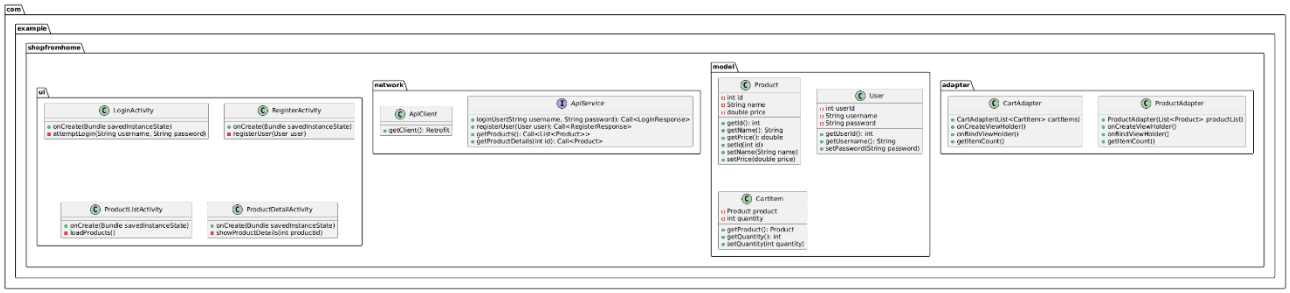
CSS
Cascading Style Sheets: fogli di stile per la formattazione dell’interfaccia utente.

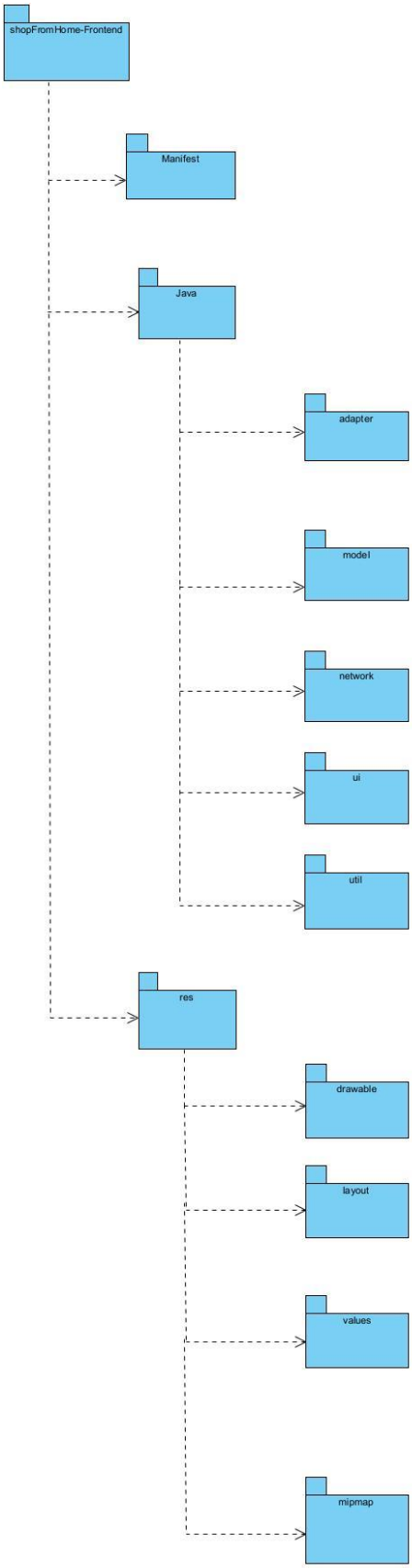
JS
JavaScript: linguaggio di programmazione per funzioni interattive nel frontend.

2 PACKAGES

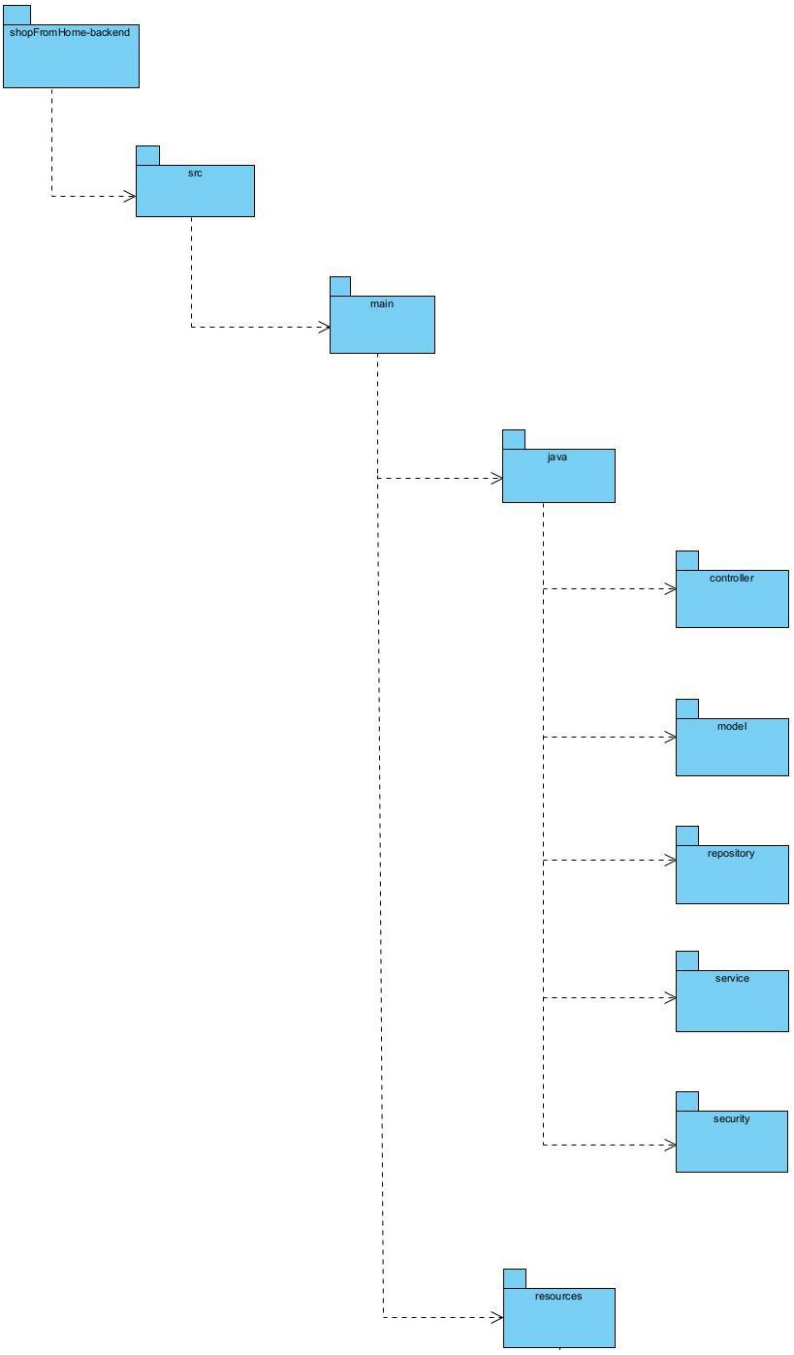
Il sistema ShopFromHome è diviso in packages nel modo seguente:

Frontend:



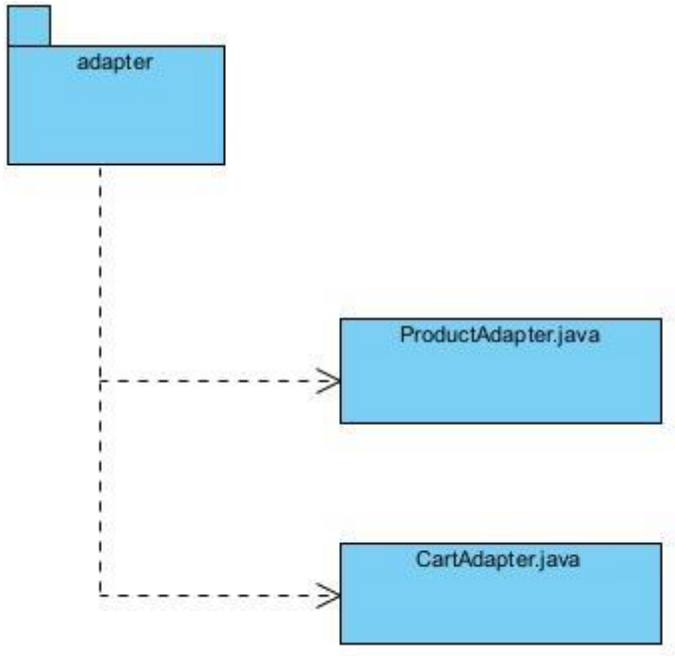

shopFromHome-Frontend
Manifest
Java
adapter
model
network
ui
util
res
drawable
layout
values
mipmap

Backend:


shopFromHome-backend
src
main
java
controller
model
repository
service
security
resources

3 CLASS INTERFACE

Adapter:



Nome Classe	ProductAdapter
Descrizione	Adapter per la visualizzazione della lista dei prodotti nel frontend.
Metodi	onCreateViewHolder(viewType: int) onBindViewHolder(holder: ViewHolder, position: int) getItemCount()
Invariante di classe	/

Nome Metodo	Descrizione
onCreateViewHolder(viewType: int)	Crea una nuova vista per un prodotto.
onBindViewHolder(holder: ViewHolder, position: int)	Associa i dati del prodotto alla view.
getItemCount()	Restituisce il numero di prodotti.

Nome Classe	CartAdapter
Descrizione	Adapter per la visualizzazione delle categorie nel frontend.
Metodi	onCreateViewHolder(viewType:int) onBindViewHolder(holder: ViewHolder, position: int) getItemCount()
Invariante di classe	/

Nome Metodo	Descrizione
onCreateViewHolder(viewType: int)	Crea una vista per un articolo del carrello.
onBindViewHolder(holder: ViewHolder, position: int)	Associa i dati del carrello alla vista.
getItemCount()	Restituisce il numero di articoli nel carrello.

Nome Classe	CategoryAdapter
Descrizione	Adapter per la visualizzazione delle categorie nel frontend.
Metodi	onCreateViewHolder(viewType:int) onBindViewHolder(holder: ViewHolder, position: int) getItemCount()
Invariante di classe	/

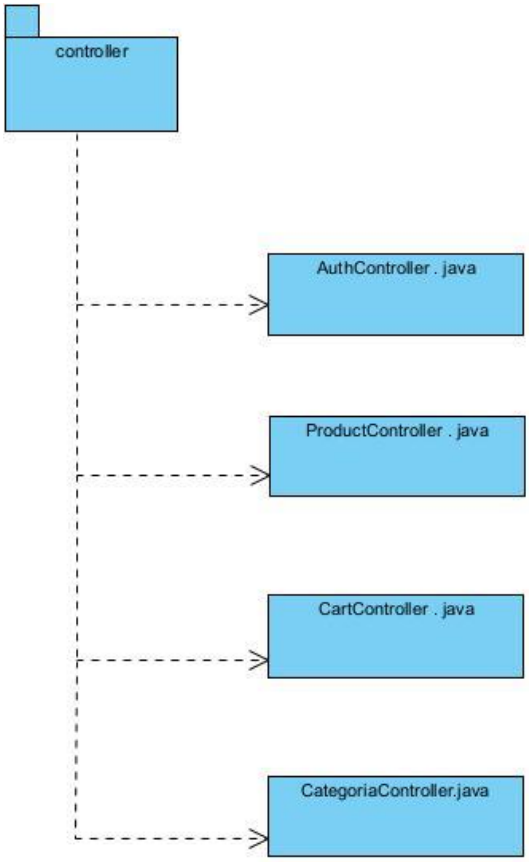
Nome Metodo	Descrizione
onCreateViewHolder(viewType: int)	Crea una vista per una categoria.
onBindViewHolder(holder: ViewHolder, position: int)	Associa i dati della categoria alla vista.
getItemCount()	Restituisce il numero di categorie.

View:

Nome Classe	CategoryActivity
Descrizione	Gestisce l'interfaccia utente per la gestione delle categorie.
Metodi	onCreate() onAddCategoryClick() onUpdateCategoryClick() onDeleteCategoryClick() displayCategories()
Invariante di classe	/

Nome Metodo	Descrizione
onCreate()	Inizializza l'interfaccia utente per la gestione delle categorie.
onAddCategoryClick()	Metodo per aggiungere una nuova categoria.
onUpdateCategoryClick()	Metodo per modificare una categoria esistente.
onDeleteCategoryClick()	Metodo per eliminare una categoria.
displayCategories()	Visualizza la lista delle categorie.

Controller:



Nome Classe	AuthController
Descrizione	Gestisce l'autenticazione e la registrazione degli utenti.
Metodi	authenticateUser(username: String, password: String) registerUser(user: User)
Invariante di classe	/

Nome Metodo	Descrizione
authenticateUser(username: String, password: String)	Autentica un utente.
registerUser(user: User)	Registra un nuovo utente.

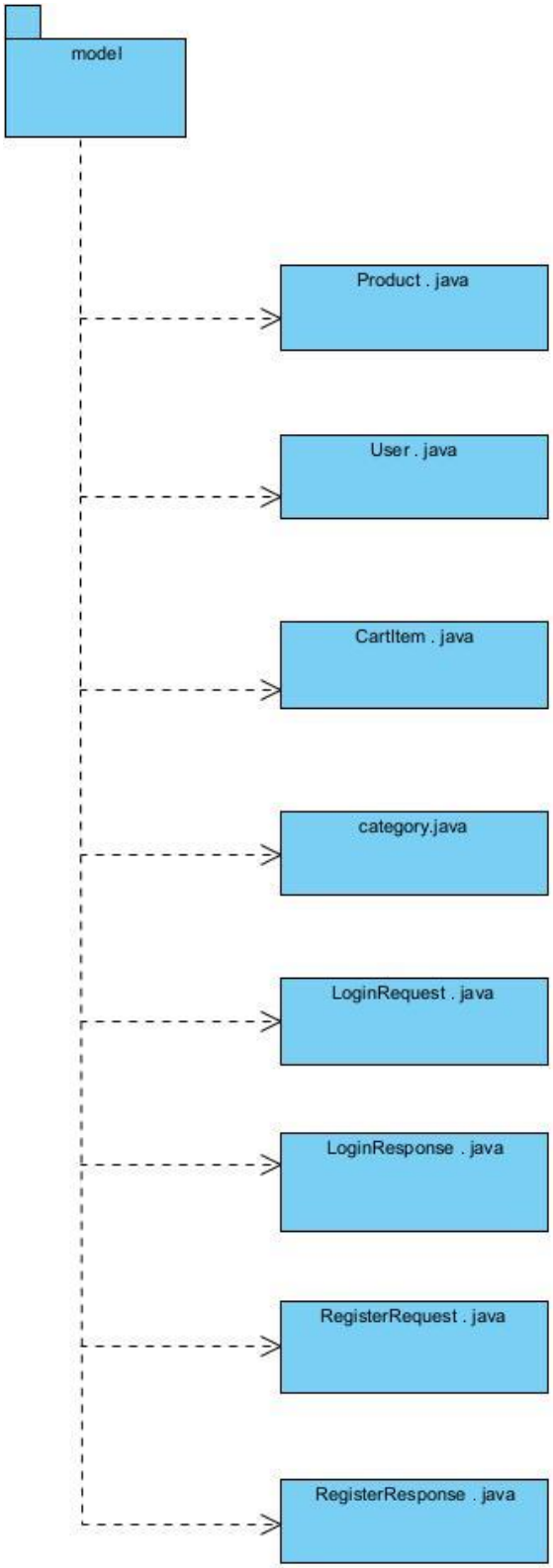
Nome Classe	ProductController
Descrizione	Gestisce le operazioni sui prodotti.
Metodi	getProducts() getProductById(id: int) createProduct(product: Product) updateProduct(id: int, product: Product) deleteProduct(id: int)
Invariante di classe	/

Nome Metodo	Descrizione
getProducts()	Restituisce la lista di tutti i prodotti.
getProductById(id: int)	Restituisce un prodotto per ID.
createProduct(product: Product)	Crea un nuovo prodotto.
updateProduct(id: int, product: Product)	Modifica un prodotto esistente.
deleteProduct(id: int)	Elimina un prodotto.

Nome Classe	CartController
Descrizione	Gestisce le operazioni sul carrello degli utenti.
Metodi	getCart(userId: int) addItemToCart(userId: int, productId: int, quantity: int) removeItemFromCart(userId: int, productId: int) updateCartItem(userId: int, productId: int, quantity: int)
Invariante di classe	/

Nome Metodo	Descrizione
getCart(userId: int)	Restituisce il carrello dell'utente.
addItemToCart(userId: int, productId: int, quantity: int)	Aggiunge un articolo al carrello.
removeItemFromCart(userId: int, productId: int)	Rimuove un articolo dal carrello.
updateCartItem(userId: int, productId: int, quantity: int)	Modifica la quantità di un articolo nel carrello.

Model:


model
Product . java
User . java
CartItem . java
category.java
LoginRequest . java
LoginResponse . java
RegisterRequest . java
RegisterResponse . java

Nome Classe	User
Descrizione	Rappresenta un utente nel sistema.
Metodi	getId() getUsername() getPassword() setPassword(password: String)
Invariante di classe	/

Nome Metodo	Descrizione
getId()	Restituisce l’ID dell’utente.
getUsername()	Restituisce lo username dell’utente.
getPassword()	Restituisce la password dell’utente.
setPassword(password: String)	Imposta la password dell’utente.

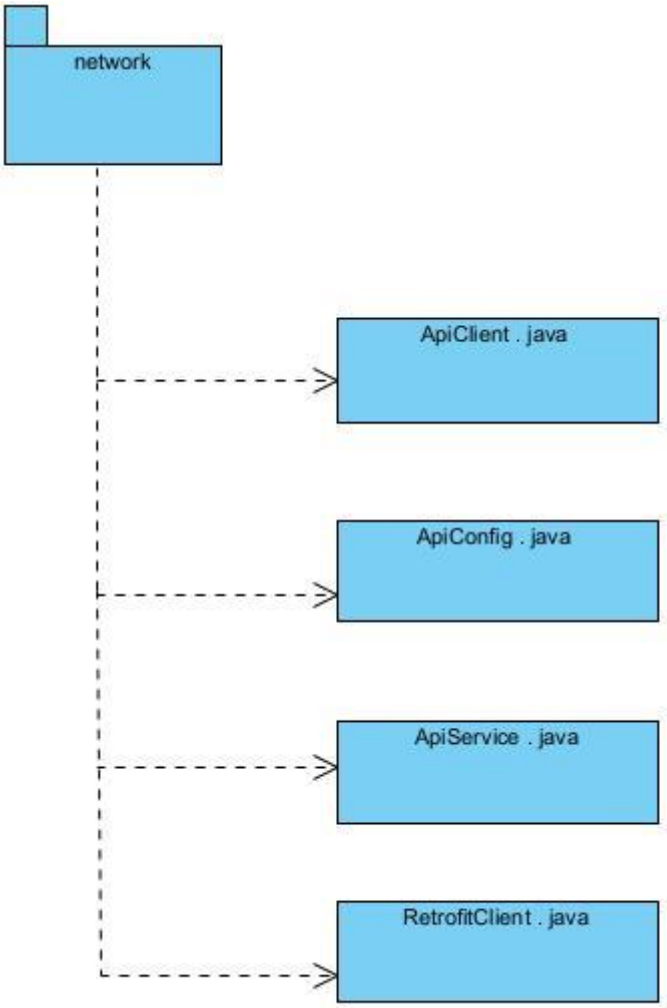
Nome Classe	Product
Descrizione	Rappresenta un prodotto nel sistema.
Metodi	getId() getName() getPrice() setPrice(password: double)
Invariante di classe	/

Nome Metodo	Descrizione
getId()	Restituisce l'ID del prodotto.
getName()	Restituisce il nome del prodotto.
getPrice()	Restituisce il prezzo del prodotto.
setPrice(password: double)	Imposta il prezzo del prodotto.

Nome Classe	CartItem
Descrizione	Rappresenta un articolo del carrello.
Metodi	getProductId() getQuantity() setQuantity(quantity: int)
Invariante di classe	/

Nome Metodo	Descrizione
getProductId()	Restituisce l'ID del prodotto.
getQuantity()	Restituisce la quantità dell'articolo.
setQuantity(quantity: int)	Imposta la quantità dell'articolo.

API:



Nome Classe	ApiService
Descrizione	Gestisce le chiamate API per interagire con il backend.
Metodi	getProducts() getProductById(id: int) createProduct(product: Product) updateProduct(id: int, product: Product) deleteProduct(id: int)
Invariante di classe	/

Nome Metodo	Descrizione
getProducts()	Recupera la lista di tutti i prodotti.
getProductById(id: int)	Recupera un prodotto per ID.
createProduct(product: Product)	Crea un nuovo prodotto.
updateProduct(id: int, product: Product)	Modifica un prodotto esistente.
deleteProduct(id: int)	Elimina un prodotto.

Nome Classe	ApiConfig
Descrizione	Configura le impostazioni per le chiamate API.
Metodi	initialize()
Invariante di classe	/
Nome Metodo	**Descrizione**
initialize()	Inizializza le impostazioni per le chiamate API.