UNIVERSITÀ DEGLI STUDI DI CATANIA

DIPARTIMENTO DI MATEMATICA E INFORMATICA

CORSO DI LAUREA MAGISTRALE IN INFORMATICA

Matteo Imbrosciano

Shop

Ralazione

Prof: Emiliano Alessio Tramontana

Prof: Andrea Francesco Fornaia

ANNO ACCADEMICO 2023/24

INDICE

1 Introduzione

Il progetto Product Shop ha l'obiettivo di creare un sistema a microservizi client-server che permetta agli utenti di gestire la configurazione di prodotti come T-Shirt, Scarpe e Pantaloni. Il sistema è progettato per mantenere una lista degli acquisti del cliente e implementa un meccanismo di Circuit Breaker per prevenire chiamate a servizi esterni non funzionanti, garantendo così la resilienza e l'affidabilità dell'applicazione. L'utente può interagire con l'applicazione attraverso varie funzionalità, tra cui la visualizzazione dei prodotti disponibili, l'aggiunta di prodotti al carrello, la visualizzazione del carrello e la ricerca di prodotti specifici. Le tecnologie utilizzate per implementare il sistema includono Java 19 come linguaggio di programmazione e Spring Boot come framework per facilitare lo sviluppo dei microservizi. La comunicazione tra i componenti avviene tramite API RESTful, garantendo interoperabilità e facilità di integrazione. Il sistema è composto da diversi microservizi, ciascuno con un ruolo specifico:

- **Client:** Questo microservizio permette all'utente di inserire il proprio nome per creare una sessione. Utilizza il Circuit Breaker per gestire le chiamate alle API REST, delegando le richieste a HttpRequest. Il client può visualizzare la lista dei prodotti disponibili, cercare prodotti, effettuare acquisti e visualizzare il proprio carrello.

- **Server:** Il server riceve le richieste dal client e gestisce i dati dei prodotti in vendita. Permette la ricerca dei prodotti e, in caso di richieste relative al carrello o agli acquisti, inoltra le richieste al microservizio di gestione della sessione.

- **Sessione:** Questo microservizio si occupa di gestire la sessione degli utenti. Mantiene una mappatura tra il nome del cliente e la lista degli acquisti effettuati, serializzando e salvando lo stato della sessione nel file system per garantirne la persistenza.

Il progetto utilizza il pattern Circuit Breaker per gestire la resilienza delle chiamate ai servizi esterni. Il Circuit Breaker cambia stato (Closed, Open, HalfOpen) in base ai risultati delle richieste inviate, garantendo un uso efficiente delle risorse e mantenendo l'applicazione reattiva anche in caso di malfunzionamenti dei servizi esterni. La struttura modulare e l'uso di design pattern come Session State rendono il sistema flessibile.

2 Analisi Tecnica

Nome Progetto Product Shop

Cliente Product Shop S.R.L.

Versione Documento 1.0

Autore Matteo Imbrosciano

Data Creazione 01/06/2024

2.1 Business Requirement

Creare un sistema a microservizi client-server, in cui il client contatta per il server per la configurazione di Product(come T-Shirt, Scarpe e Pantaloni). Il sistema deve tenere la lista degli acquisti del client. Implementare un sistema di Circuit Breaker in modo da evitare le chiamate quando il sistema non funziona bene. L'utente deve visualizzare i prodotti disponibili, aggiungere un prodotto al carrello, visualizzare il suo carrello e cercare un prodotto.

2.2 Tecnologie di Implementazione

Linguaggio: Java 19

Framework: Spring Boot

Comunicazione HTTP: API RESTful

2.3 Microservizi

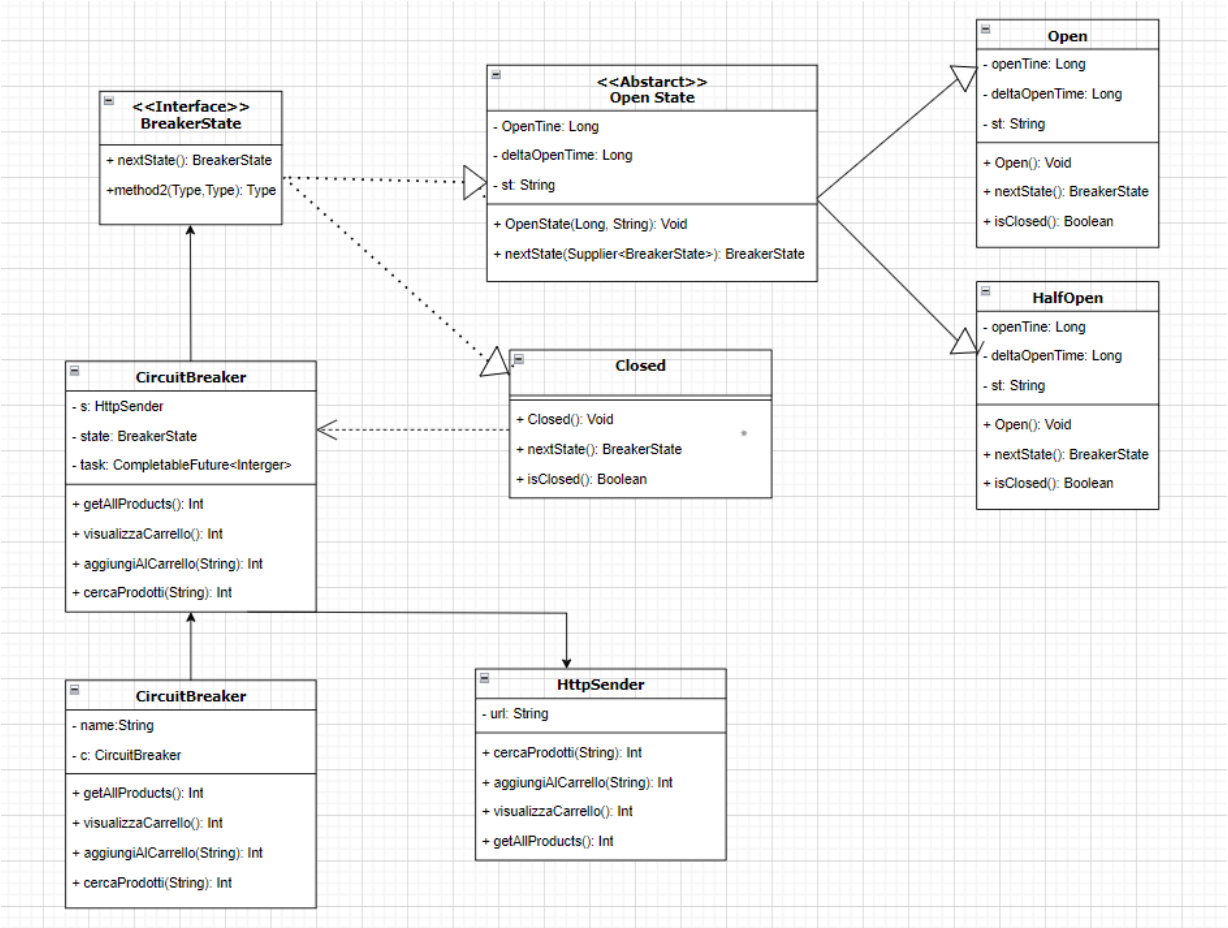
Ho implementato i seguenti microservizi:

- Client: permette all'utente di inserire il nome con cui sarà salvata la sessione. Contiene il Cicuit Breaker tramite il quale effettuiamo le chiamate REST API. Il Cicuit Breaker ha un riferimento alla classe HttpRequest per mandare le richieste al server. Il client può visualizzare la lista dei prodotti(T-Shirt, Scarpe, Pantaloni) disponibili, può cercare un prodotto scrivendo una parola contenuta nel nome di esso, può acquistare un capo e visualizzare la lista dei suoi acquisti.

- Server: riceve le richieste dal Client. Il Server contiene i dati dei prodotti in vendita, permette la ricerca dei dispositivi. Nel caso in cui il client chiede di visualizzare il suo carrello o chiede di acquistare un prodotto il Server inoltra la richiesta al microservizio che si occupa di gestire la sessione.

- Sessione: si occupa di gestire la sessione dei client. Contiene una mappatura in cui ogni nome del client e associata la lista degli acquisti effettuati. Lo stato viene serializzato e salvato nel file system del microservizio così da salvare i dati dell'utente anche quando esso non è collegato.

2.4 Diagramma UML Client



Questo diagramma mostra come il pattern Circuit Breaker può essere implementato utilizzando diverse classi e stati per gestire la resilienza di un'applicazione. Il CircuitBreaker cambia stato in base al risultato delle richieste inviate tramite HttpSender, garantendo che le risorse siano utilizzate in modo efficiente e che l'applicazione rimanga reattiva anche quando i servizi esterni non funzionano correttamente.

2.4.1 Microservizio Client: Classi

BreakerState (Interfaccia): Questa interfaccia rappresenta lo stato generico del circuito. Definisce il metodo nextState() per determinare il prossimo stato del circuito e isClosed() e il Circuit Breaker è nello stato chiuso (Closed), serve a controllare se il Circuit Breaker permette il passaggio delle richieste.

OpenState (Classe Astratta): Questa classe astratta rappresenta lo stato "aperto" del circuito e fornisce un'implementazione parziale che può essere estesa da altre classi concrete. Contiene attributi per gestire il tempo di apertura e delta del tempo di apertura.

Open (Classe Concreta, Estende OpenState): Rappresenta lo stato in cui il circuito è aperto e non permette il passaggio delle richieste. Implementa i metodi per determinare il prossimo stato e verificare se è chiuso.

HalfOpen (Classe Concreta, Estende OpenState): Rappresenta lo stato in cui il circuito sta testando se può tornare a uno stato chiuso. Implementa i metodi per l'apertura e il controllo dello stato.

Closed (Classe Concreta): Rappresenta lo stato in cui il circuito è chiuso e le richieste possono passare liberamente. Contiene metodi per determinare il prossimo stato e verificare se è chiuso.

Classi Operative:

CircuitBreaker (Classe Concreta): Questa classe implementa il Circuit Breaker e gestisce lo stato corrente (che può essere Closed, Open, o HalfOpen). Utilizza HttpSender per inviare le richieste.

Client (Classe Concreta): Rappresenta il cliente che utilizza il CircuitBreaker per inviare richieste.

HttpSender (Classe Concreta): Rappresenta il componente responsabile di inviare le richieste HTTP al servizio esterno.

2.4.2 Descrizione del Flusso

Client: Un'istanza di Client utilizza il CircuitBreaker per inviare richieste a un servizio esterno tramite HttpSender.

Metodi disponibili per il Client includono getAllProducts(), visualizzaCarrello(), aggiungiAlCarrello(String), e cercaProdotti(String).

CircuitBreaker: Il CircuitBreaker gestisce il passaggio delle richieste basato sul suo stato corrente (Closed, Open, o HalfOpen).

Utilizza HttpSender per inviare richieste al servizio esterno.

Il metodo nextState() nei vari stati determina quando il circuito dovrebbe cambiare stato.

HttpSender: HttpSender è responsabile di effettuare le richieste HTTP al servizio esterno.

Include metodi per ottenere prodotti, visualizzare il carrello, aggiungere prodotti al carrello e cercare prodotti.

2.4.2 Design Pattern: Circuit Breaker

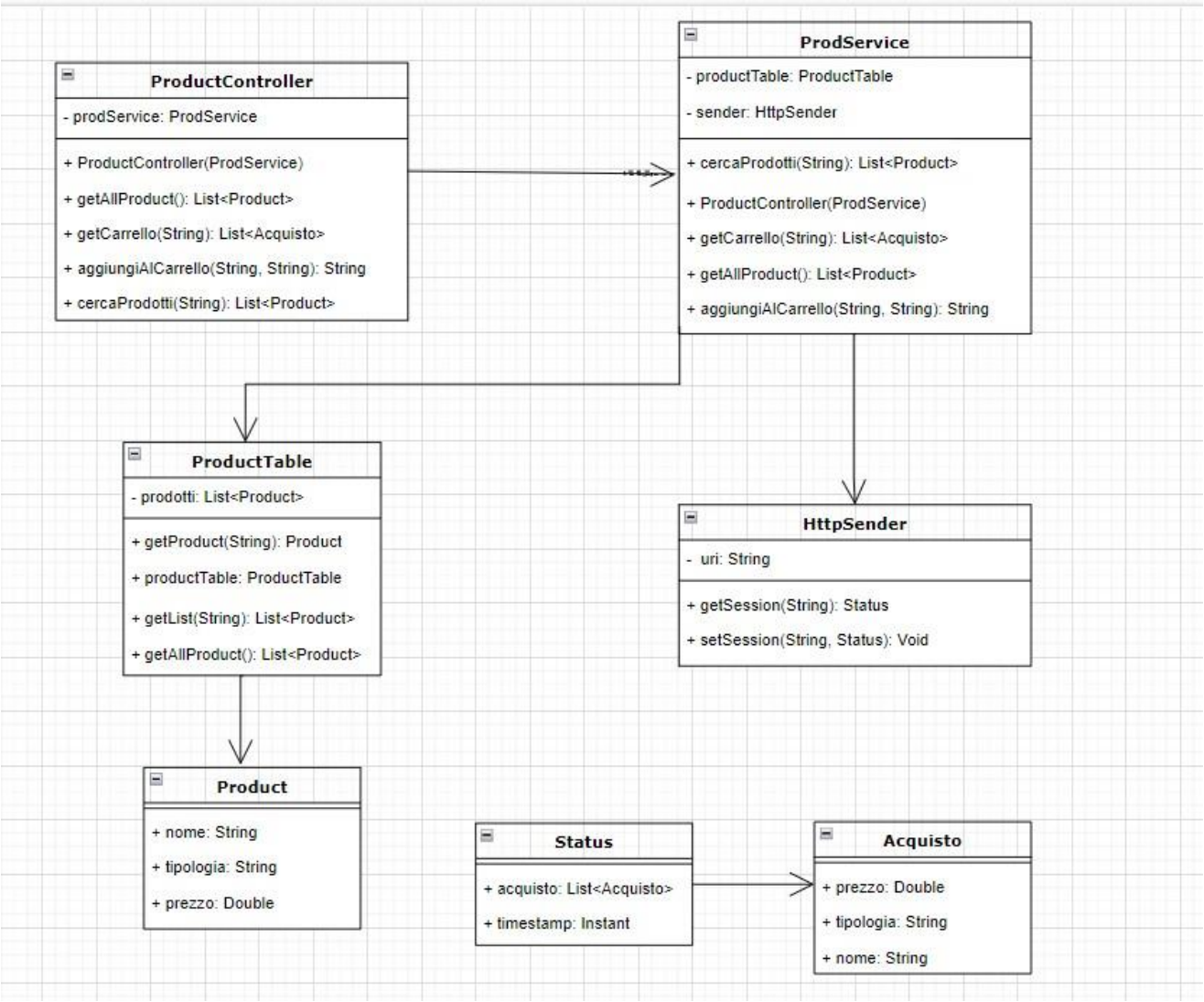
Il pattern Circuit Breaker viene utilizzato per prevenire chiamate continue a un servizio esterno fallito. Quando il numero di errori supera una soglia, il Circuit Breaker "si apre" per un certo periodo, bloccando le richieste al servizio e dando tempo al servizio di recuperare.

Closed State: Le richieste passano normalmente.

Open State: Le richieste sono bloccate per un certo periodo (openTime).

HalfOpen State: Alcune richieste sono inviate per verificare se il servizio è ripristinato.

2.5 Diagramma UML Server



Il sistema è progettato per essere modulare e facile da mantenere, con chiari separazioni di responsabilità tra le diverse classi. ProductController fornisce un punto di accesso centralizzato, ProdService gestisce la logica di business, ProductTable funge da repository di dati, e HttpSender gestisce la comunicazione HTTP e le sessioni. Questa struttura è a tre livelli rende il sistema flessibile e facilmente estensibile.

2.5. Microservizio Server: Classe

ProductController: ProductController funge da punto di accesso principale per i client che desiderano eseguire operazioni sui prodotti e sul carrello. Utilizza un'istanza di ProdService per delegare le operazioni necessarie.

ProdService: ProdService incapsula la logica di business relativa ai prodotti e alle operazioni del carrello. Si interfaccia con ProductTable per accedere ai dati dei prodotti e con HttpSender per gestire la comunicazione HTTP e le sessioni.

ProductTable: ProductTable agisce come repository locale per i prodotti. Fornisce metodi per recuperare singoli prodotti, liste di prodotti e tutti i prodotti disponibili.

HttpSender: HttpSender gestisce la comunicazione con servizi esterni via HTTP, incluse le operazioni di gestione delle sessioni utente.

Product: Product rappresenta un prodotto con attributi di nome, tipologia e prezzo.

Status: Status tiene traccia dello stato della sessione, incluso il carrello con gli acquisti e il timestamp dell'ultima modifica.

Acquisto: Acquisto rappresenta un singolo acquisto nel carrello di un utente.

2.5.2 Descrizione del Flusso

Interazione del Client con ProductController: Il client interagisce con ProductController per eseguire operazioni come ottenere tutti i prodotti (getAllProduct), visualizzare il carrello (getCarrello), aggiungere un prodotto al carrello (aggiungiAlCarrello) e cercare prodotti (cercaProdotti).

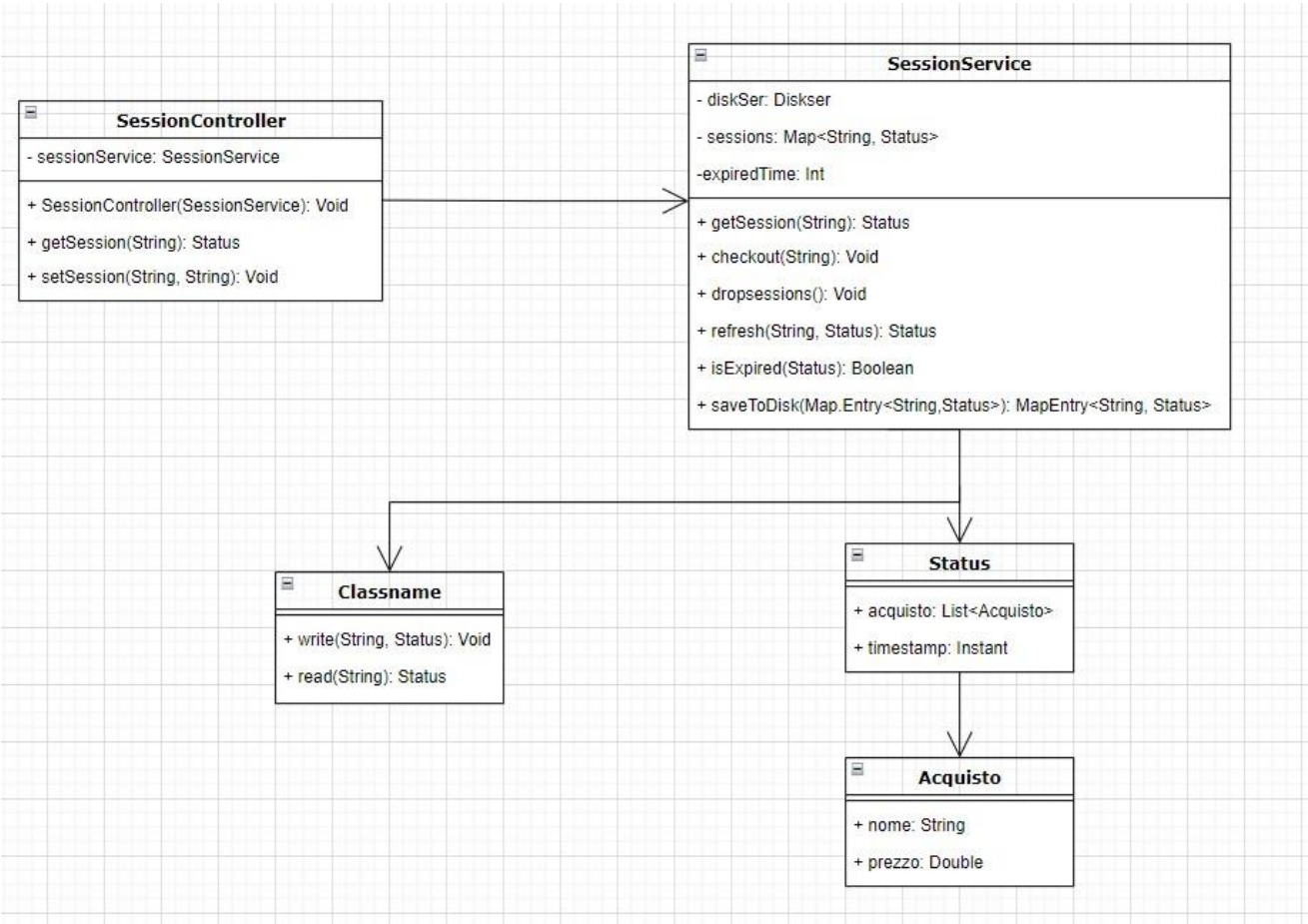
ProductController Delega a ProdService: ProductController utilizza ProdService per delegare le richieste. Questo permette a ProductController di rimanere semplice e focalizzato sullo scopo dell'interfaccia utente.

ProdService Gestisce la Logica di Business: ProdService esegue le operazioni richieste interfacciandosi con ProductTable per accedere ai dati dei prodotti e con HttpSender per gestire le sessioni utente e altre comunicazioni HTTP necessarie.

ProductTable come Repository di Prodotti: ProductTable fornisce l'accesso ai dati dei prodotti, permettendo di recuperare informazioni dettagliate su specifici prodotti o liste di prodotti.

HttpSender Gestisce le Sessioni: HttpSender gestisce le sessioni utente, fornendo metodi per ottenere lo stato di una sessione (getSession) e per impostare lo stato di una sessione (setSession).

2.6 Diagramma UML Session



Questo diagramma mostra un sistema ben strutturato per la gestione delle sessioni utente, utilizzando il pattern Session State organizzato in un’architettura a tre livelli. SessionController offre un'interfaccia chiara per le operazioni di sessione, delegando la logica complessa a SessionService, che a sua volta utilizza DiskSer e altre classi di supporto per le operazioni di persistenza e gestione dello stato. Questo design facilita la manutenzione e l'espansione del sistema, mantenendo una separazione pulita delle responsabilità. Il pattern Session State gestisce lo stato dell'utente su più richieste, consentendo di mantenere informazioni come il carrello degli acquisti, lo stato di autenticazione, ecc. durante l'interazione con l'applicazione.

2.6.1 Microservizio Session: Classe

SessionController: SessionController funge da punto di accesso principale per la gestione delle sessioni. Utilizza un'istanza di SessionService per eseguire le operazioni necessarie. Fornisce metodi per ottenere (getSession) e impostare (setSession) lo stato di una sessione.

SessionService: SessionService gestisce la logica di business delle sessioni. Mantiene una mappa delle sessioni (sessions) e utilizza DiskSer per operazioni di salvataggio su disco. Include metodi per recuperare una sessione (getSession), effettuare il checkout (checkout), rimuovere sessioni (dropsessions), aggiornare una sessione (refresh), verificare se una sessione è scaduta (isExpired) e salvare le sessioni su disco (saveToDisk).

DiskSer: DiskSer non è dettagliato nel diagramma, ma presumibilmente è responsabile delle operazioni di I/O su disco, gestendo il salvataggio e il caricamento delle sessioni.

Status: Status rappresenta lo stato di una sessione, inclusi gli acquisti effettuati (acquisto) e il timestamp dell'ultima attività (timestamp).

Acquisto: Acquisto rappresenta un singolo acquisto effettuato all'interno di una sessione, con attributi per il nome del prodotto (nome) e il prezzo (prezzo).

Classname: Classname sembra essere un placeholder per una classe che gestisce le operazioni di scrittura e lettura delle sessioni. Presumibilmente, questa classe interagisce con DiskSer per salvare e recuperare lo stato delle sessioni.

2.6.2 Descrizione del Flusso

Interazione del Client con SessionController: Il client utilizza SessionController per eseguire operazioni di gestione delle sessioni come ottenere una sessione (getSession) e impostare una sessione (setSession).

SessionController Delega a SessionService: SessionController utilizza SessionService per eseguire la logica di business relativa alle sessioni. Questo permette a SessionController di rimanere focalizzato sull'interfaccia utente e delegare la complessità al servizio.

SessionService Gestisce la Logica di Business: SessionService gestisce tutte le operazioni relative alle sessioni, come il recupero (getSession), l'aggiornamento (refresh), il controllo di scadenza (isExpired), e il salvataggio su disco (saveToDisk). Mantiene una mappa delle sessioni correnti (sessions).

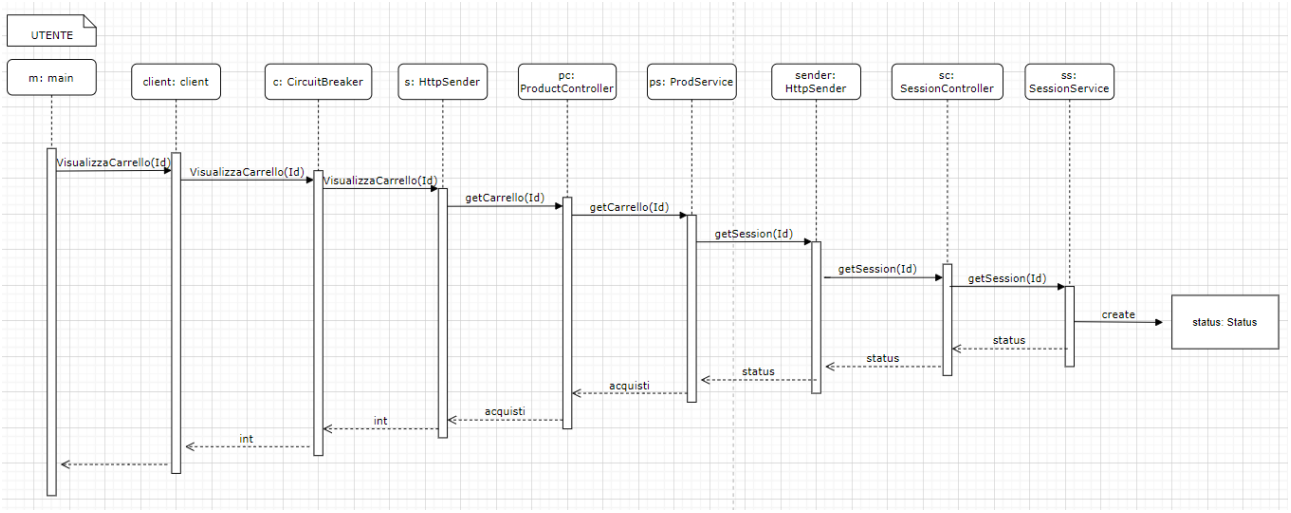
DiskSer Gestisce l'I/O su Disco: Anche se DiskSer non è dettagliato, è presumibile che gestisca il salvataggio e il caricamento delle sessioni su disco, facilitando la persistenza dello stato delle sessioni.

Classname Esegue Operazioni di I/O: Classname fornisce metodi per scrivere (write) e leggere (read) lo stato delle sessioni, interagendo probabilmente con DiskSer.

2.6.3 Design Pattern: Session State

Il design pattern **Session State** viene utilizzato per mantenere lo stato dell'utente su più richieste in un'applicazione, tipicamente in un contesto web. Questo pattern permette di memorizzare informazioni come il carrello degli acquisti, lo stato di autenticazione e altre preferenze dell'utente, in modo che possano essere persistenti tra diverse interazioni con il sistema.

2.7 Diagramma di frequenza: Visualizzazione del Carrello



Flusso delle Interazioni

- Utente -> m: main: L'utente avvia l'operazione chiamando VisualizzaCarrello(id) nel punto di ingresso principale dell'applicazione (main).

- m: main -> client: client: Il metodo VisualizzaCarrello(id) viene chiamato sull'oggetto client.
- client: client -> c: CircuitBreaker: Il client chiama VisualizzaCarrello(id) sul CircuitBreaker (c). Il CircuitBreaker è responsabile di controllare se il servizio è disponibile e di gestire eventuali fallback.

- c: CircuitBreaker -> s: HttpSender: Il CircuitBreaker chiama VisualizzaCarrello(id) su HttpSender (s). L'HttpSender si occupa di inviare la richiesta HTTP al server.

- s: HttpSender -> pc: ProductController: L'HttpSender invia una richiesta HTTP al ProductController (pc), che gestisce le richieste relative ai prodotti.

- pc: ProductController -> ps: ProdService: Il ProductController chiama getCarrello(id) sul ProdService (ps). Il ProdService contiene la logica di business per recuperare gli acquisti nel carrello.

- ps: ProdService -> sender: HttpSender: Il ProdService chiama getSession(id) sul HttpSender (sender) per ottenere lo stato della sessione associata all'ID dell'utente.

- sender: HttpSender -> sc: SessionController: L'HttpSender invia una richiesta al SessionController (sc) per ottenere la sessione.

- sc: SessionController -> ss: SessionService: Il SessionController chiama getSession(id) sul SessionService (ss).

- ss: SessionService -> status: Status: Il SessionService recupera lo stato della sessione, crea un nuovo Status se necessario, e restituisce lo stato al SessionController.

- ss: SessionService -> sc: SessionController: SessionService restituisce lo stato (status) al SessionController.

- sc: SessionController -> sender: HttpSender: SessionController restituisce lo stato (status) al HttpSender.

- sender: HttpSender -> ps: ProdService: HttpSender restituisce lo stato (status) al ProdService.

- ps: ProdService -> pc: ProductController: ProdService utilizza lo stato della sessione per recuperare gli acquisti e li restituisce al ProductController.

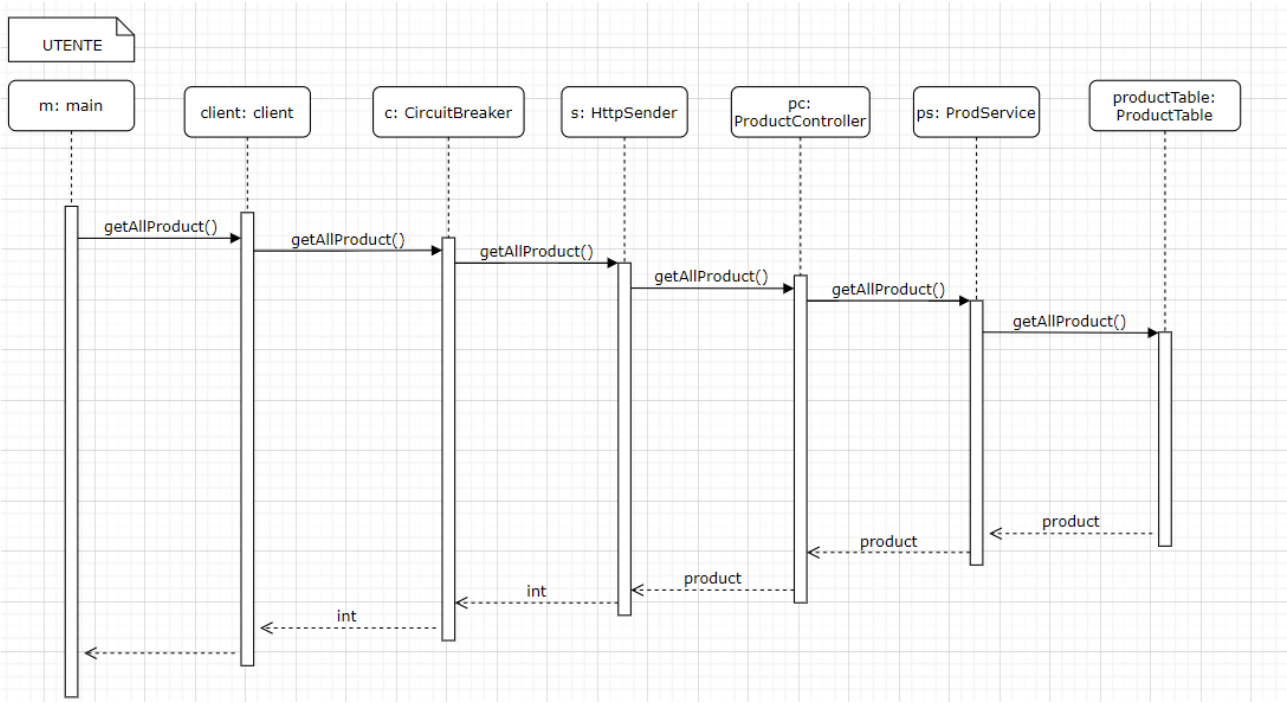
- pc: ProductController -> s: HttpSender: ProductController restituisce gli acquisti a HttpSender.

- s: HttpSender -> c: CircuitBreaker: HttpSender restituisce gli acquisti a CircuitBreaker.

- c: CircuitBreaker -> client: client: CircuitBreaker restituisce gli acquisti a client.

- client: client -> m: main: client restituisce gli acquisti al punto di ingresso principale (main), che a sua volta li fornisce all'utente.

2.8 Diagramma di frequenza: Visualizzazione Prodotti



Flusso delle Interazioni

- Utente -> m: main: L'utente inizia l'operazione chiamando il metodo getAllProduct() nel punto di ingresso principale dell'applicazione (main).

- m: main -> client: client: Il metodo getAllProduct() viene chiamato sull'oggetto client.

- client: client -> c: CircuitBreaker: Il client chiama getAllProduct() sul CircuitBreaker (c). Il CircuitBreaker è responsabile di controllare se il servizio è disponibile e di gestire eventuali fallback.

- c: CircuitBreaker -> s: HttpSender: Il CircuitBreaker chiama getAllProduct() su HttpSender (s). L'HttpSender si occupa di inviare la richiesta HTTP al server.

- s: HttpSender -> pc: ProductController: L'HttpSender invia una richiesta HTTP al ProductController (pc), che gestisce le richieste relative ai prodotti.

- pc: ProductController -> ps: ProdService: Il ProductController chiama getAllProduct() sul ProdService (ps). Il ProdService contiene la logica di business per recuperare i prodotti.

- ps: ProdService -> productTable: ProductTable: Il ProdService chiama getAllProduct() sul ProductTable (productTable). ProductTable è responsabile di accedere ai dati dei prodotti dal database o da un'altra fonte di dati.

- productTable: ProductTable -> ps: ProdService: ProductTable recupera i dati dei prodotti e li restituisce a ProdService.

- ps: ProdService -> pc: ProductController: ProdService restituisce i dati dei prodotti a ProductController.

- pc: ProductController -> s: HttpSender: ProductController restituisce i dati dei prodotti a HttpSender.

- s: HttpSender -> c: CircuitBreaker: HttpSender restituisce i dati dei prodotti a CircuitBreaker.

- c: CircuitBreaker -> client: client: CircuitBreaker restituisce i dati dei prodotti a client.

- client: client -> m: main: client restituisce i dati dei prodotti al punto di ingresso principale (main), che a sua volta li fornisce all'utente.