OBJECT DESIGN DOCUMENT

The Spoon

Versione	**0.6**
Data	17/01/2024
Destinatario	Esame di Ingegneria del Software 2023/24
Presentato da	Alessandro Pascarella (0512109149) Vincenzo Catone (0512106704) Jacopo Gennaro Esposito (0512121285)

Revision History

Data	Versione	Descrizione	Autori
09/01/2024	0.1	Prima stesura	Alessandro Pascarella Vincenzo Catone Jacopo Gennaro Esposito
10/01/2024	0.2	Definizione dei package	Alessandro Pascarella Vincenzo Catone Jacopo Gennaro Esposito
11/01/2024	0.3	Definizione e stesura dei design pattern	Alessandro Pascarella Vincenzo Catone Jacopo Gennaro Esposito
12/01/2024	0.4	Implementazione delle interfacce delle classi e del class diagram ristrutturato	Alessandro Pascarella Vincenzo Catone Jacopo Gennaro Esposito
14/01/2024	0.5	Revisione finale	Alessandro Pascarella Vincenzo Catone Jacopo Gennaro Esposito
17/01/2024	0.6	Revisione	Alessandro Pascarella Vincenzo Catone Jacopo Gennaro Esposito

Team members

Nome	Ruolo nel progetto	Acronimo	Informazioni di contatto
Alessandro Pascarella	Team Member	AP	a.pascarella9@studenti.unisa.it
Vincenzo Catone	Team Member	VC	v.catone@studenti.unisa.it
Jacopo Gennaro Esposito	Team Member	JE	j.esposito9@studenti.unisa.it

Sommario

1. Introduzione.....5
1.1 Linee guida per la scrittura del codice.....5
1.2 Definizione, acronimi e abbreviazioni.....5
1.3 Riferimenti e link utili.....5
2. Packages.....6
3. Packages.....10
4. Class Diagram Ristrutturato.....26
5. Elementi di Riuso.....26
6. Glossario.....28

1. Introduzione

L'obiettivo prefissato di The Spoon è di diventare la principale piattaforma per la prenotazione di tavoli e la gestione degli ordini nei ristoranti d'Italia. Ciò consente agli utenti di effettuare ordini online per ridurre i tempi di attesa e di prenotare un tavolo senza stress, offrendo ai clienti un'esperienza utente intuitiva e ai ristoratori uno strumento efficiente ed efficace per gestire prenotazioni e ordini.

1.1 Linee guida per la scrittura del codice

Le linee guida prevedono una serie di convenzioni e canoni che gli sviluppatori dovrebbero utilizzare e rispettare nella progettazione delle interfacce del sistema. Per la definizione e costituzione delle linee guida si è provveduta a fare riferimento alla convenzione Java nota come **Java spring**.

Link a documentazione ufficiale sulle convenzioni

Di seguito una lista di link alle convenzioni usate per definire le linee guida:

- **Java Spring:** https://docs.spring.io/spring-framework/reference/index.html
- **HTML:** https://www.w3schools.com/html/html5_syntax.asp

1.2 Definizione, acronimi e abbreviazioni

Di seguito, sono riportate alcune definizioni presenti nel documento:

- **Package:** raggruppamento di classi, interfacce o file correlati;
- Design pattern: template di soluzioni a problemi ricorrenti, impiegati per ottenere riuso e flessibilità;
- **interfaccia:** insieme di signature delle operazioni offerte dalla classe;
- **Javadoc:** sistema di documentazione offerta da Java, che viene generato sotto forma di interfaccia in modo da rendere la documentazione accessibile e facilmente leggibile.

1.3 Riferimenti e link utili

Di seguito una lista di riferimenti e link ad altri documenti utili durante la lettura:

- Requirements Analysis Document
- System Design Document
- Test Plan
- Javadoc di The Spoon

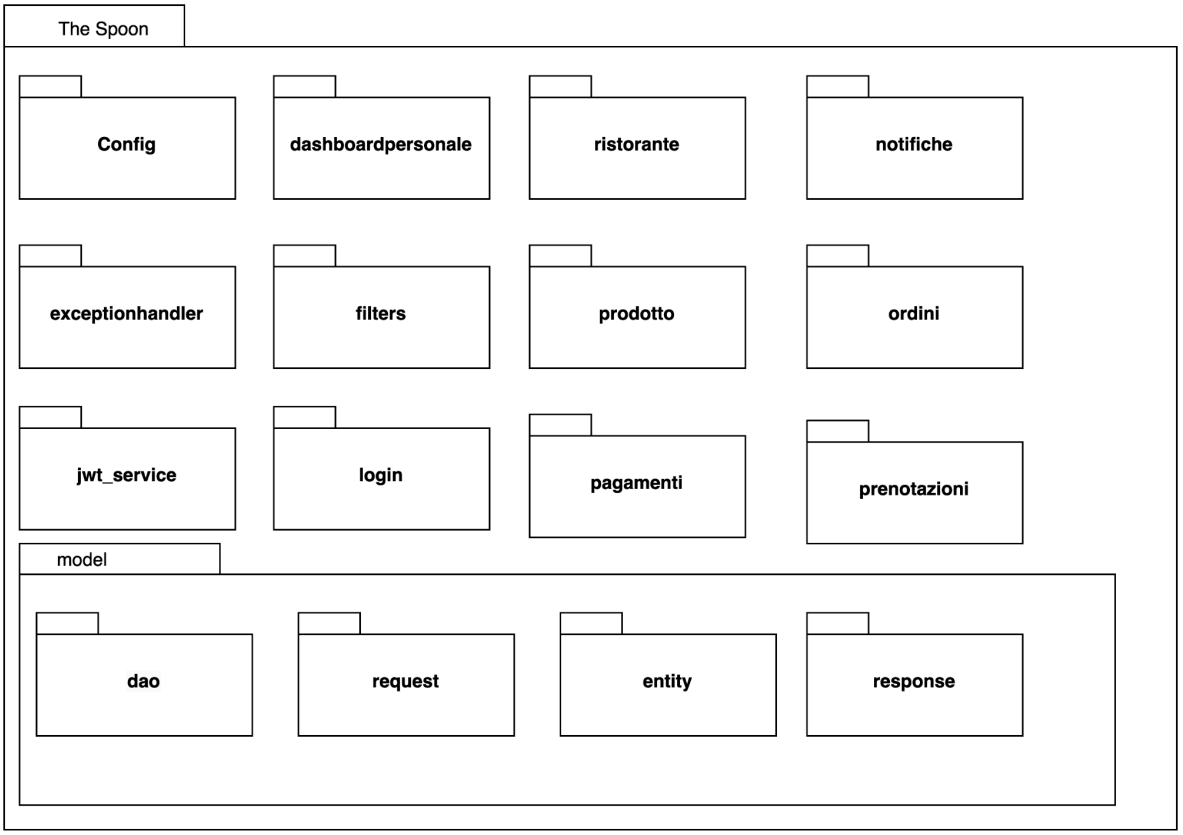
2. Packages

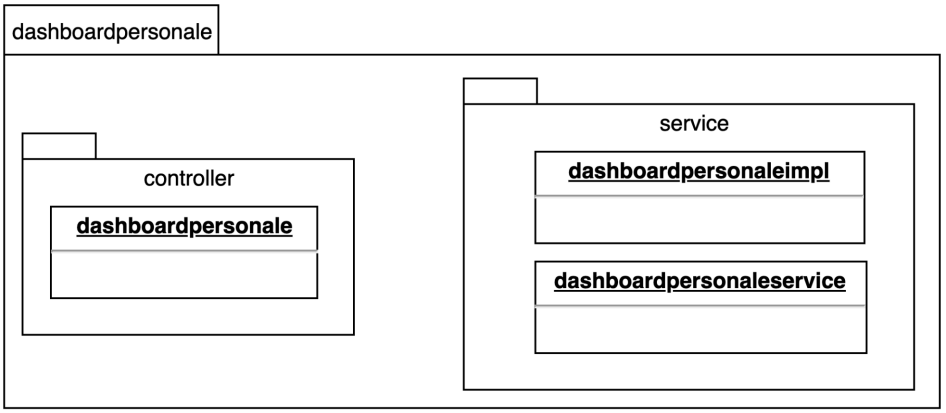
In questa sezione viene mostrata la suddivisione del sistema in package, in base a quanto definito nel documento System Design. Tale suddivisione è motivata dalle scelte architetturali prese e ricalca la struttura di directory standard definita da Maven.

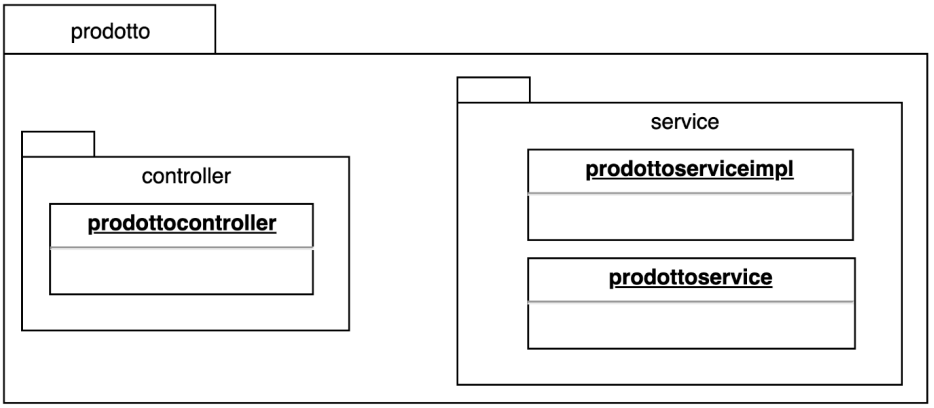
- **.idea**
- **.mvn,** contiene tutti i file di configurazione per Maven
- **src,** contiene i file sorgenti
    - **main**
        - **java,** contiene tutte le classi Java relativamente alle componenti Logic e Data
    - **test,** contiene tutto il necessario per il testing
        - **java,** contiene le classi Java necessarie per l'implementazione del testing
- **target,** contiene tutti i file prodotti dal sistema di build di Maven

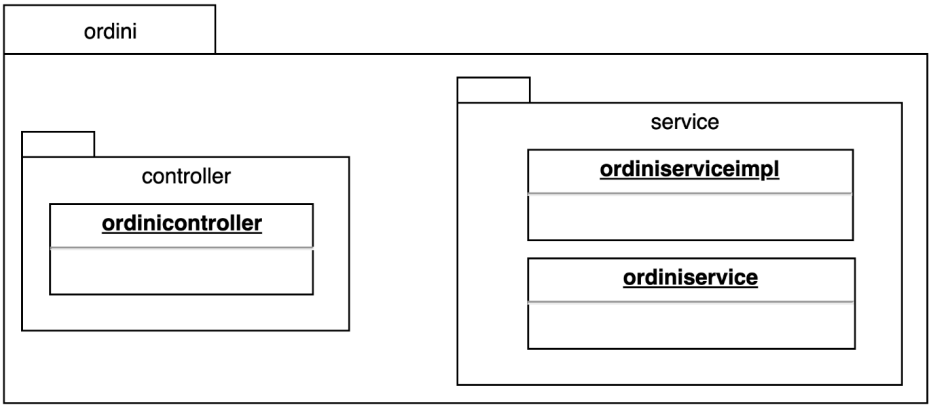
Package di The Spoon

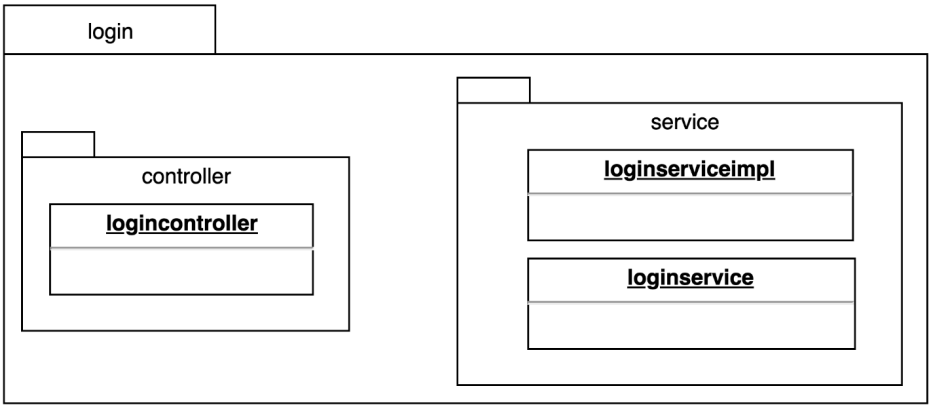
Nella seguente sezione si procede a mostrare e ad evidenziare la struttura del package principale di The Spoon.
La struttura prevede un **package principale** con diversi **sottosistem**i, ognuno dei quali a sua volta è suddiviso in un **package controller,** responsabile della gestione degli input e del coordinamento delle azioni, e un **package service**, contenente la logica di business o di servizio per fornire le funzionalità specifiche del sottosistema. Questa organizzazione modulare facilita la manutenzione, l'estensione e la comprensione del codice, promuovendo una separazione chiara delle responsabilità e consentendo una maggiore flessibilità nell'evoluzione del software.

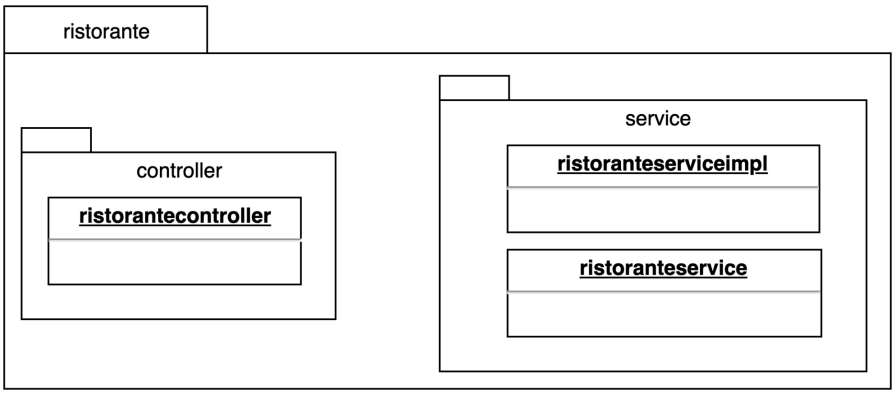

The Spoon
Config
dashboardpersonale
ristorante
notifiche
exceptionhandler
filters
prodotto
ordini
jwt_service
login
pagamenti
prenotazioni
model
dao
request
entity
response

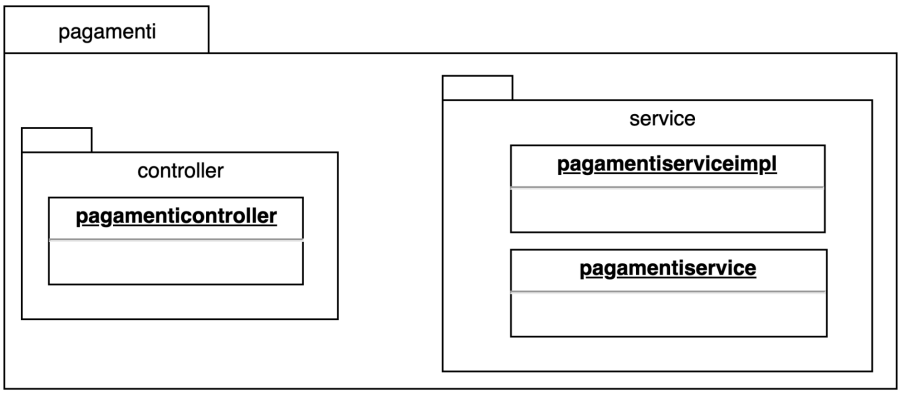

dashboardpersonale
controller
dashboardpersonale
service
dashboardpersonaleimpl
dashboardpersonaleservice

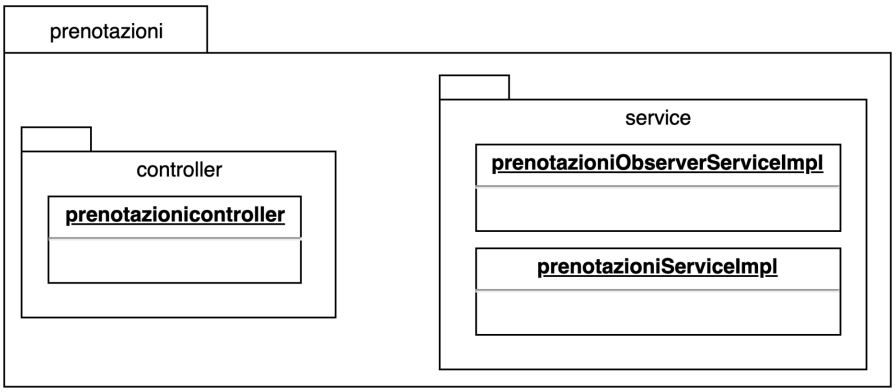

prodotto
controller
prodottocontroller
service
prodottoserviceimpl
prodottoservice

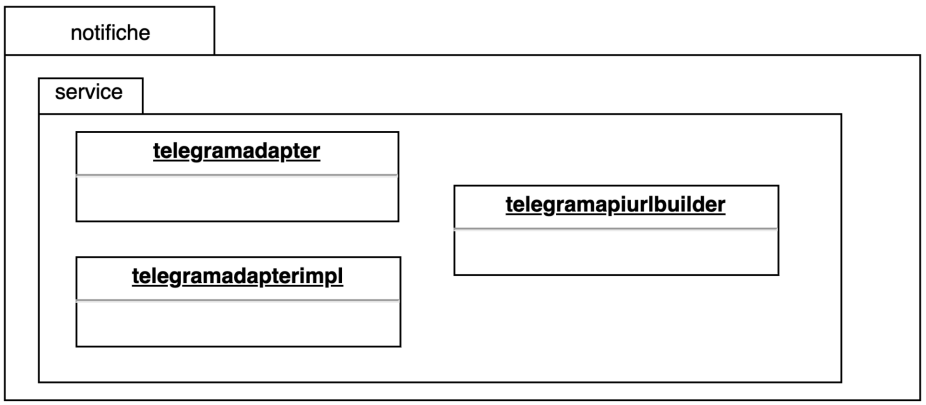

ordini
controller
ordinicontroller
service
ordiniserviceimpl
ordiniservice


login
controller
logincontroller
service
loginserviceimpl
loginservice


ristorante
controller
ristorantecontroller
service
ristoranteserviceimpl
ristoranteservice


pagamenti
controller
pagamenticontroller
service
pagamentiserviceimpl
pagamentiservice


prenotazioni
controller
prenotazionicontroller
service
prenotazioniObserverServiceImpl
prenotazioniServiceImpl


notifiche
service
telegramadapter
telegramadapterimpl
telegramapiurlbuilder

JavaDoc di The Spoon

Per motivi di leggibilità si è scelto di creare un sito, hostato su GitHub pages, contenente la JavaDoc di The Spoon. In tale maniera, chiunque può consultare la documentazione aggiornata dell'intero sistema.

Di seguito, il link al sito in questione: https://github.com/Pascareddum/TheSpoon

1. Package DashboardPersonale

it.unisa.thespoon.login.dashboardpersonale.controller

Nome classe	DashboardPersonaleController
Descrizione	Controller contenente gli endpoint delle API di TheSpoon per il sottosistema Dashboard Personale
Metodi	+getAllRistoratoreDetails(String email) +getRistoratoreDetails(String email) +saveRistoratore(Ristoratore ristoratore) +updatePassword(UpdatePasswordRequest updatePasswordRequest, String email) +updateRistoratoreDetails(UpdateRistoratoreRequest updateRistoratoreRequest, String email)
Invariante di classe	/

Nome Metodo	**getRistoratoreDetails**
Descrizione	Metodo adibito ad ottenere i dettagli del ristoratore
Pre-condizione	**context:** DashboardPersonaleController:: +Post(request,response) **pre:** request.getParameter("autenthication"). != null
Post-condizione	**context:** DashboardPersonaleController:: +doPost(request,response) **post:** Ristoratore.RistoratoreDataDisplay

Nome metodo	**updateRistoratoreDetails**
Descrizione	Metodo adibito alla modifica dei dettagli del ristoratore

Pre-condizione	**context:** DashboardPersonaleController:: +doPost(request,response) **pre:** updateRistoranteRequest != null and request.getParameter("autenthication") != null
Post-condizione	**context**: DashboardPersonaleController:: +doPost(request,response) **Post:** ResponseEntity

Nome metodo	**updatePassword**
Descrizione	Metodo adibito alla modifica della password dell'account ristoratore
Pre-condizione	**context:** DashboardPersonaleController:: +Post(request,response) **pre:** updatePasswordRequest != null and request.getParameter("autenthication"). != null
Post-condizione	**context**: DashboardPersonaleController:: +doPost(request,response) **Post:** ResponseEntity Risposta

Nome metodo	**getAllRistoratoreDetails**
Descrizione	Metodo adibito ad ottennere tutti i dettagli del ristoratore
Pre-condizione	**context:** DashboardPersonaleController:: +Get(request,response) **pre:** updatePasswordRequest != null
Post-condizione	**context**: DashboardPersonaleController:: +doPost(request,response) **Post:** ResposeEntity Ristoratore != null

Nome metodo	**saveRistoratore**
Descrizione	Metodo adibito a salvare le modifiche apportate al ristoratore
Pre-condizione	**context:** DashboardPersonaleController:: +Post(request,response) **pre:** Ristoratore ristoratore != null
Post-condizione	**context**: DashboardPersonaleController:: +doPost(request,response) **Post:** ResposeEntity != null

2. Package Login

it.umisa.thespoon.login.control

Nome classe	LoginController
Descrizione	Controller contenente gli endpoint delle API di TheSpoon per il sottosistema di login
Metodi	+JwtAuthenticationResponse login(LoginRequest loginRequest) +JwtAuthenticationResponse signUp(SignupRequest registerRequest)
Invariante di classe	/

Nome Metodo	JwtAuthenticationResponse login(LoginRequest loginRequest)
Descrizione	Firma del metodo per il login
Pre-condizione	**context:** LoginController +Post(request, response) pre: loginRequest != null
Post-condizione	**context:** LoginController +doPost(request, response) **post:** Token di autenticazione != null

Nome Metodo	JwtAuthenticationResponse signUp(SignupRequest registerRequest)
Descrizione	Firma del metodo per la registrazione
Pre-condizione	**context:** LoginController +Post(request, response) **pre**: registerRequest != null
Post-condizione	**context:** LoginController +doPost(request, response) **post:** Token di autenticazione != null

3. Package Prodotto

package it.unisa.thespoon.prodotto.controller

Nome classe	ProdottoController
Descrizione	Controller contenente gli endpoint delle API di TheSpoon per il sottosistema Prodotto
Metodi	+getProdotto(Integer Id) +insertProdotto(InsertProdottoRequest insertProdottoRequest) +removeProdotto(Integer Id) +saveProdotto(Prodotto prodotto)
Invariante di classe	/

Nome metodo	insertProdotto(InsertProdottoRequest insertProdottoRequest)
Descrizione	Metodo adibito all'inserimento di un nuovo prodotto nel sistema
Pre-condizione	**Context:** ProdottoController +doPost(request,response) **pre:** insertProdottoRequest != null
Post-condizione	**context:** ProdottoController +doPost(request,response) **post:** ResponseEntity Codice != null

Nome metodo	removeProdotto(Integer Id)
Descrizione	Metodo adibito alla rimozione di un nuovo prodotto dal sistema
Pre-condizione	**Context:** ProdottoController +doPost(request,response) **Pre:** id != null
Post-condizione	**context:** ProdottoController +doPost(request,response) **post:** REsponseEntity Codice != null

Nome metodo	getProdotto(Integer Id)
Descrizione	Metodo per recuperare un prodotto dato il suo ID

Pre-condizione	**Context:** ProdottoController +doPost(request,response) **Pre:** Id != null
Post-condizione	**context:** ProdottoController +doPost(request,response) **post:** Optional != null

Nome metodo	saveProdotto(Prodotto prodotto)
Descrizione	Metodo per salvare i prodotti
Pre-condizione	**context:** ProdottoController +doPost(request,response) **pre:** prodotto != null
Post-condizione	**context:** ProdottoController +doPost(request,response) **post:** Prodotto != null

4. Package Ordini

package it.unisa.thespoon.ordini.controller

Nome classe	OrdiniController
Descrizione	Controller contenente gli endpoint delle API di TheSpoon per il sottosistema ordine
Metodi	+confermaOrdine(Integer idOrdine, String email) +getOrdineByIdOrdineAndIdRistorante(Integer idOrdine, Integer idRistorante) +getProdttoByIdOrdineIdRistorante(Integer idRistorante, Integer idOrdine, String email) +insertOrdine(InsertOrdineRequest insertOrdineRequest) +ordiniByIdRistorante(idOrdine, String name) +getStato(Byte stato, Ordine ordine)
Invariante di classe	/

Nome metodo	insertOrdine(InsertOrdineRequest insertOrdineRequest)
Descrizione	Metodo per inserire un nuovo ordine

Pre-condizione	**context:** OrdiniController +doPost(request, response) **pre:** insertOrdineRequest != null
Post-condizione	**context:** OrdiniController +doPost(request, response) **post**:ResponseEntity Response != null

Nome metodo	confermaOrdine(Integer idOrdine, String email)
Descrizione	Firma del metodo per confermare un nuovo ordine
Pre-condizione	**context:** OrdiniController +doPost(request, response) **pre:** idOrdine != null and email != null
Post-condizione	**context:** OrdiniController +doPost(request, response) **post:** ResponseEntity Response != null

Nome metodo	ordiniByRistorante(Integer idRistorante, String email)
Descrizione	Firma del metodo per ottenere i dettagli di un ordine
Pre-condizione	**context:** OrdiniController +doPost(request, response) **pre:** idRistorante != null and email != null
Post-condizione	**context:** OrdiniController +doPost(request, response) **post:** ResponseEntity Response != null

Nome metodo	getProdottoByIdOrdineRistorante(Integer idRistorante, Integer idOrdine, String email)
Descrizione	Firma del metodo per ottenere i dettagli dei prodotti associati ad un ordine dato il suo id e l'id del ristorante
Pre-condizione	**context:** OrdiniController

	+doGet(request, response) **pre:** idRistorante != null and idOrdine != null and email != null
Post-condizione	**context:** OrdiniController +doPost(request, response) **post:** ResponseEntity Response != null

Nome metodo	getOrdineByIdOrdinedAndIdRistorante(Integer idOrdine, Integer idRistorante)
Descrizione	Metodo per ottenere i dettagli di un ordine dato il suo ID e l'id del ristorante
Pre-condizione	**context:** OrdiniController +doGet(request, response) **pre:** idOrdine != null idRistorante != null
Post-condizione	**context:** OrdiniController +doPost(request, response) **post:** Ordine != null

Nome metodo	setStato(Byte stato, Ordine ordine)
Descrizione	Metodo che permette di aggiornare lo stato di un ordine sul db, chiamato da altri sottosistemi
Pre-condizione	**context:** OrdiniController +doPost(request, response) **pre:** stato = false and ordine != null
Post-condizione	**context:** OrdiniController +doPost(request, response) **post:** stato = true and ordine != null

5. Package Ristorante

package it.unisa.thespoon.ristorante.controller

Nome classe	RistoranteController
Descrizione	Controller contenente gli endpoint delle API di TheSpoon per il sottosistema Ristorante
Metodi	+addProductToMenu(Integer idMenu, Integer idProdotto, Integer idRistorante, String name) +getAllRistorantiByRistoratore(Integer idRistoratore) +getMenusByID(Integer idMenu) +getMenusByRistoranteID(Integer ID) +getProdottiByMenuID(Integer idMenu) +getRistoranteByID(Integer idRistorante) +getTavoliRistorante(Integer idRistorante) +getTavoloByID(String idTavolo, Integer idRistorante) +insertMenu(InsertMenuRequest insertMenuRequest, String email) +insertRistorante(InsertRistoranteRequest insertRistoranteRequest, String email) +insertTavolo(InsertTavoloRequest insertTavoloRequest, String email) +removeProductFromMenu(Integer idMenu, Integer idProdotto, Integer idRistorante, String mail) +searchRistorante(SearchRistoranteRequest searchRistoranteRequest, String nomeRistorante) +updateRistorante(UpdateRistoranteRequest updateRistoranteRequest, Integer idRistorante, String email)
Invariante di classe	/

Nome metodo	addProductToMenu(Integer idMenu, Integer idProdotto, Integer idRistorante, String email)
Descrizione	Metodo per aggiungere un prodotto ad un menu

Pre-condizione	**context: Ristorante**Controller +doPost(request, response) **pre:** idMenu != null and idProdotto != null and idRistorante != null and email != null
Post-condizione	**context:** OrdiniController +doPost(request, response) **post:** ResponseEntity HttpStatus Codice != null

Nome metodo	removeProductFromMenu(Integer idMenu, Integer idProdotto, Integer idRistorante, String email
Descrizione	Metodo per rimuovere un prodotto ad un menu
Pre-condizione	**context: Ristorante**Controller +doPost(request, response) **pre:** idMenu != null and idProdotto != null and idRistorante != null and email != null
Post-condizione	**context:** OrdiniController +doPost(request, response) **post:** HttpStatus ResponseEntity Codice != null

Nome metodo	getMenusByRistoranteId(Integer id)
Descrizione	Metodo per ottenere i menù associati ad un ristorante
Pre-condizione	**context: Ristorante**Controller +doGet(request, response) **pre:** id != null
Post-condizione	**context:** OrdiniController +doGet(request, response) **post:** ResponseEntity ResponseEntity != null

Nome metodo	getMenusById(Integer idMenu)
Descrizione	Metodo per ottenere i menù associati ad un dato id
Pre-condizione	**context: Ristorante**Controller +doGet(request, response)

	pre: id != null
Post-condizione	**context:** OrdiniController +doGet(request, response) **post:** ResponseEntity ResponseEntity != null

Nome metodo	getProdottiByMenuId (Integer idMenu)
Descrizione	Metodo per ottenere i prodotti associati ad un dato menù
Pre-condizione	**context: Ristorante**Controller +doGet(request, response) **pre:** idMenu != null
Post-condizione	**context:** OrdiniController +doGet(request, response) **post:** ResponseEntity ResponseEntity != null

Nome metodo	insertTavolo (InsertTavoloRequest insertTavoloRequest, String email)
Descrizione	Metodo per inserire un nuovo tavolo
Pre-condizione	**context: Ristorante**Controller +doPost(request, response) **pre:** insertTavoloRequest != null and email != null
Post-condizione	**context:** OrdiniController +doGet(request, response) **post:** ResponseEntity ResponseEntity != null

Nome metodo	getTavoliRistorante(Integer idRistorante)
Descrizione	Metodo per recuperare i tavoli associati ad un ristorante
Pre-condizione	**context: Ristorante**Controller +doGet(request, response) **pre:** idRistorante != null
Post-condizione	**context:** OrdiniController +doGet(request, response) **post:** ResponseEntity Set != null

Nome metodo	getTavoloById(String idTavolo, Integer idRistorante)
Descrizione	Metodo per recuperare i dettagli di un tavolo dato il suo ID
Pre-condizione	**context: Ristorante**Controller +doGet(request, response)

	pre: idTavolo != null and idRistorante != null
Post-condizione	**context:** OrdiniController +doGet(request, response) **post:** ResponseEntity Set != null

Nome metodo	getAllRistorantiByRistoratore(Integer idRistoratore)
Descrizione	Metodo che permette di recuperare tutti i ristoranti associati ad un ristoratore
Pre-condizione	**context:** OrdiniController +doGet(request, response) **pre:** idRistoratore != null
Post-condizione	**context:** OrdiniController +doPost(request, response) **post:** ResponseEntity ResponseEntity != null

Nome metodo	getRistoranteById(Integer idRistorante)
Descrizione	Firma del metodo per recuperare i dettagli di un ristorante
Pre-condizione	**context:** OrdiniController +doGet(request, response) **pre:** idRistoratore != null
Post-condizione	**context:** OrdiniController +doPost(request, response) **post:** ResponseEntity ResponseEntity != null

Nome metodo	searchRistorante(SearchRistoranteRequest searchRistoranteRequest, String nomeRistorante)
Descrizione	Metodo per recuperare i ristoranti dati una serie di parametri

Pre-condizione	**context:** OrdiniController +doPost(request, response) **pre:** searchRistoranteRequest != null and nomeRistorante != null
Post-condizione	**context:** OrdiniController +doPost(request, response) **post:** ResponseEntity ResponseEntity != null

Nome metodo	insertMenu(InsertMenuRequest insertMenuRequest, String email)
Descrizione	Metodo per aggiungere un menu ad un ristorante
Pre-condizione	**context:** OrdiniController +doPost(request, response) **pre:** insertMenuRequest!= null and email!= null
Post-condizione	**context:** OrdiniController +doPost(request, response) **post:** ResponseEntity ResponseEntity != null

Nome metodo	insertRistorante(InsertRistoranteRequest insertRistoranteRequest, String email)
Descrizione	Metodo adibito all'inserimento di un nuovo ristorante
Pre-condizione	**context:** OrdiniController +doPost(request, response) **pre:** insertRistoranteRequest!= null and email!= null
Post-condizione	**context:** OrdiniController +doPost(request, response) **post:** ResponseEntity ResponseEntity != null

Nome metodo	updateRistorante(UpdateRistoranteRequest updateRistoranteRequest, Integer idRistorante, String email)
Descrizione	Metodo adibito all'inserimento di un nuovo ristorante
Pre-condizione	**context:** OrdiniController +doPost(request, response)

	pre: updateRistoranteRequest!= null and idRistorante != null and email!= null
Post-condizione	**context:** OrdiniController +doPost(request, response) **post:** ResponseEntity ResponseEntity != null

6. Package Notifiche

package it.unisa.thespoon.notifiche.service

Nome classe	TelegramAdapterImpl
Descrizione	Adapter per le API di Telegram
Metodi	+inviaMessaggioNotifica(Integer chatID, String text)
Invariante di classe	/

Nome metodo	inviaMessaggioNotifica(Integer chatId, String text)
Descrizione	Metodo per inviare un messaggio attravverso le API di Telegram
Pre-condizione	**context:** TelegramAdapter +doPost(request, response) **pre:** chatId!= null and text != null
Post-condizione	**context:** OrdiniController +doPost(request, response) **post:** /

7. Package Pagamenti

package it.unisa.thespoon.pagamenti.controller

Nome classe	PagamentiController

Descrizione	Controller contenente gli endpoint delle API di TheSpoon per il sottosistema Pagamento
Metodi	+creaSessionePagamento(Integer idOrdine, Integer idRistoratore) +handleSuccessoPagamento(com.stripe.model.checkout.Session session)
Invariante di classe	/

Nome metodo	creaSessionePagamento(Integer idOrdine, Integer idRistorante)
Descrizione	Metodo adibito a creare una sessione di pagamento in un ristorante
Pre-condizione	**context:** PagamentiController +doPost(request, response) **pre:** idOrdine!= null and idRistorante != null
Post-condizione	**context:** OrdiniController +doPost(request, response) **post:** /

Nome metodo	handleSuccessoPagamento(com.stripe.model.checkout.Session session)
Descrizione	Metodo che indica il successo di un pagamento
Pre-condizione	**context:** PagamentiController +doPost(request, response) **pre:** session != null
Post-condizione	**context:** OrdiniController +doPost(request, response) **post:** /

8. Package Prenotazioni

package it.unisa.thespoon.prenotazioni.controller

Nome classe	PrenotazioniController

Descrizione	Controller contenente gli endpoint delle API di TheSpoon per il sottosistema Ordine
Metodi	+insertPrenotazion(InsertPrenotazioneRequest insertPrenotazioneRequest) +updatePrenotazione(UpdatePrenotazioneRequest updatePrenotazioneRequest) +getAllPrenotazioni(Integer idRistorante, String email) +getAllTavoliByIDPrenotazione(Integer idPrenotazione, String Email) +confermaPrenotazione(Integer idPrenotazione, String Email) +generatePassword()
Invariante di classe	/

Nome metodo	insertPrenotazione(InsertPrenotazioneRequest insertPrenotazioneRequest)
Descrizione	Metodo adibito all'inserimento di una nuova prenotazione all'interno di TheSpoon
Pre-condizione	**context:** PrenotazioniController +doPost(request, response) **pre:** insertPrenotazioniRequest != null
Post-condizione	**context:** OrdiniController +doPost(request, response) **post:** ResponseEntity Response != null

Nome metodo	updatePrenotazione(UpdatePrenotazioneRequest updatePrenotazioneRequest)
Descrizione	Firma del metodo adibito alla modifica di una prenotazione all'interno di TheSpoon

Pre-condizione	**context:** PrenotazioniController +doPost(request, response) **pre:** updatePrenotazioniRequest != null
Post-condizione	**context:** OrdiniController +doPost(request, response) **post:** ResponseEntity Response != null

Nome metodo	getAllPrenotazioni(Integer idRistorante, String email)
Descrizione	Firma del metodo per ottenere le prenotazioni di un ristorante dato il suo id
Pre-condizione	**context:** PrenotazioniController +doGet(request, response) **pre:** idRistorante!= null and email != null
Post-condizione	**context:** OrdiniController +doPost(request, response) **post:** ResponseEntity Response != null

Nome metodo	getAllTavoliByIdPrenotazione(Integer idPrenotazione, String email)
Descrizione	Firma del metodo per ottenere i tavoli associati ad una prenotazione dato il suo id
Pre-condizione	**context:** PrenotazioniController +doGet(request, response) **pre:** idPrenotazione != null and email != null
Post-condizione	**context:** OrdiniController +doPost(request, response) **post:** ResponseEntity Response != null

Nome metodo	confermaPrenotazione(Integer idPrenotazione, String email)
Descrizione	Firma del metodo per confermare una prenotazione

Pre-condizione	**context:** PrenotazioniController +doPost(request, response) **pre:** idPrenotazione != null and email != null
Post-condizione	**context:** OrdiniController +doPost(request, response) **post:** ResponseEntity Response != null

Nome metodo	generatePassword()
Descrizione	Metodo per generare una password per le prenotazioni, verrà usata in fase di modifica prenotazione.
Pre-condizione	**context:** PrenotazioniController +doPost(request, response) **pre:** /
Post-condizione	**context:** OrdiniController +doPost(request, response) **post:** /

4. Class Diagram Ristrutturato

Per questione di visibilità si consiglia la visione del Class Diagram Ristrutturato al seguente link: Class Diagram.jpg

5. Elementi di Riuso

Nella seguente sezione si procede a descrivere e a dettagliare i design pattern utilizzati per procedere allo sviluppo e alla realizzazione di The Spoon. Per i design pattern utilizzati si procede a definire:

- Breve descrizione del design pattern utilizzato;
- Problematica da risolvere per il contesto applicativo di The Spoon;
- Breve spiegazione della risoluzione della problematica presentata nello sviluppo applicativo di The Spoon;
- Un grafico della struttura delle classi, che implementano il pattern;

Observer

Il pattern Observer è un design pattern comportamentale che stabilisce una dipendenza uno-a-molti tra gli oggetti in modo che quando uno degli oggetti cambia stato, tutti i suoi dipendenti vengono notificati e aggiornati automaticamente. Questo pattern è utile quando un oggetto deve informare altri oggetti di eventuali cambiamenti senza che essi siano fortemente accoppiati.
All'interno di The Spoon, il pattern Observer viene utilizzato per osservare il cambiamento di stato all'interno dei sottosistemi prenotazioni e ordini in maniera tale da poter implementare una notifica nel momento in cui lo stato della prenotazione e dell'ordine cambiano.

Facade

Il pattern Facade è un design pattern strutturale che fornisce un'interfaccia unificata per un insieme di interfacce in un sistema. Lo scopo principale di questo pattern è quello di semplificare l'interazione tra un client e un sottosistema complesso fornendo un'interfaccia semplificata e unificata.
The Spoon, essendo un sistema molto complesso, sfrutta il design pattern Facade per implementare tutta la sua logica di business e rendere più facile l'interfacciarsi con essa. Nello specifico, The Spoon utilizza il Facade per ogni suo sottosistema, implementandolo attraverso delle interfacce che sono usate per accedere ai metodi interni.

Adapter

Il pattern Adapter è un pattern di progettazione software che consente di adattare un'interfaccia di una classe per renderla compatibile con un'altra interfaccia utilizzata dalle classi client. Il pattern Adapter converte l'interfaccia di una classe in un'altra interfaccia che le classi client aspettano di trovare. In questo modo le classi esistenti possono lavorare insieme, anche se le loro interfacce sono incompatibili.
All'interno di The Spoon il pattern Adapter viene utilizzato nel contesto delle API di Telegram per adattare le API al nostro sistema e permettere un utilizzo "trasparente" delle API, in modo che ogni volta che viene effettuata una chiamata rest basterà chiamare l'interfaccia adapt.

6. Glossario

Sigla/Termine	Definizione
Package	Raggruppamento di classi e interfacce
Applicazione Web	Applicazione accessibile attraverso web per mezzo di una rete come ad esempio internet.
Observer pattern	Il pattern Observer è un design pattern comportamentale che stabilisce una dipendenza uno-a-molti tra gli oggetti in modo che quando uno degli oggetti cambia stato, tutti i suoi dipendenti vengono notificati e aggiornati automaticamente. Questo pattern è utile quando un oggetto deve informare altri oggetti di eventuali cambiamenti senza che essi siano fortemente accoppiati.
Facade pattern	Il pattern Facade è un design pattern strutturale che fornisce un'interfaccia unificata per un insieme di interfacce in un sistema. Lo scopo principale di questo pattern è quello di semplificare l'interazione tra un client e un sottosistema complesso fornendo un'interfaccia semplificata e unificata.
Adapter pattern	Il pattern Adapter è un pattern di progettazione software che consente di adattare un'interfaccia di una classe per renderla compatibile con un'altra interfaccia utilizzata dalle classi client. Il pattern Adapter converte l'interfaccia di una classe in un'altra interfaccia che le classi client aspettano di trovare. In questo modo le classi esistenti possono lavorare insieme, anche se le loro interfacce sono incompatibili.