Università degli Studi di Padova

FACOLTÀ DI INGEGNERIA
Corso di Laurea Triennale in Ingegneria Informatica

Tesi di laurea triennale

Applicazione per l'efficientamento della raccolta di ordini nei ristoranti tramite selezione rapida dai menù: sviluppo del back-end

Candidato:
Michele Zadro
Matricola 1218668

Relatore:
Nicola Zingirian

Anno Accademico 2021–2022

Sommario

La seguente tesi descrive il lavoro svolto durante il periodo di stage dal 11/07/2022 al 02/09/2022 della durata di circa trecento ore, presso l'azienda Sync Lab S.r.l. di Padova. L'attività svolta durante questa esperienza lavorativa riguarda la progettazione e lo sviluppo della parte di back-end di un'applicazione web. Il progetto a cui ho preso parte si intitola "SushiLab" e ha come scopo quello di permettere agli utenti che vanno a mangiare a un ristorante di sushi di poter ordinare più facilmente tramite l'apposita applicazione. I principali obbiettivi da raggiungere erano: lo studio delle tecnologie necessarie alla realizzazione del progetto e l'implementazione fisica di quest'ultimo.

Indice

Capitolo 1

Introduzione

1.1 L'azienda

Sync Lab è un'azienda che si occupa di realizzare prodotti e soluzioni tecnologiche per diversi mercati quali: Sanità, Industria, Energia, Telco, Finanza e Trasporti & Logistica. Viene fondata a Napoli nel 2002 e cresce rapidamente aprendo numerose sedi presso Roma (2004), Milano (2006), Padova (2008), Verona (2019) e Como (2021). Attualmente, Sync Lab ha più di 150 clienti diretti e finali, 300 dipendenti distribuiti tra le 6 sedi presenti in Italia.

Figura 1.1: Logo di SyncLab

1.2 Introduzione a Sushilab

Sushilab è il progetto a cui ho preso parte durante il periodo di stage. Ha come obiettivo principale quello di permettere ai clienti di un ristorante di sushi di poter ordinare tramite l'applicazione. Il cliente che desidera utilizzare questo servizio, quando si reca nel ristorante, deve come prima cosa creare una sessione del tavolo a cui siede. Una volta creata, avrà a disposizione un codice che permetterà a chi vorrà sedersi al suo tavolo di partecipare alla sessione tramite l'applicazione. Quando un cliente entra in una sessione può visualizzare il menù del ristorante e, da lì, selezionare i piatti che desidera ordinare. Infine, quando il cameriere passa al tavolo per ritirare le ordinazioni, basterà che uno dei clienti del tavolo mostri, tramite l'applicazione, la

lista completa degli ordini del tavolo. Dall'altro lato, il gestore del ristorante deve poter, tramite applicazione, creare i menù del ristorante e selezionare, per ciascuno di essi, una fascia oraria di validità. Per entrambi i casi d'uso (cliente o gestore) è richiesta la registrazione tramite email e password per potersi autenticare nel sistema.

1.3 Front-end e Back-end

Per comprendere meglio il ruolo da me svolto durante il tirocinio, bisogna analizzare prima le differenze tra front-end e back-end. Nel campo della progettazione e sviluppo software, il front-end è la parte di un sistema visibile all'utente, con cui egli può interagire tramite un'interfaccia utente. Il back-end invece, è la parte software non visibile all'utente (tipicamente installata in un computer chiamato server) che elabora e restituisce i dati al front-end.

Posizione occupata nel progetto

Il ruolo da me svolto durante il periodo di tirocinio è stato quello di back-end developer. Nello specifico ho sviluppato un'applicazione server che si occupasse di ricevere richieste HTTP (effettuate dal front-end) e rispondere ad esse nel formato opportuno.

1.4 Strumenti utilizzati

Visual Studio Code

Visual Studio Code è un editor di codice sorgente, sviluppato da Microsoft, che offre la possibilità di installare diverse estensioni tra cui il supporto a Java e Spring. Tramite Visual Studio Code, è possibile ricevere suggerimenti da parte dell'IDE mentre si sta scrivendo il codice, facilitando e velocizzando quindi, lo sviluppo del software.

Spring

Spring è uno dei principali framework open source per lo sviluppo di applicazioni basate su Java. Spring rende la programmazione più veloce, semplice e sicura e si suddivide in diversi moduli: di seguito verranno trattati quelli che hanno avuto rilevanza nello sviluppo del progetto.

Spring Core

Spring Core permette di ricevere e rispondere alle richieste HTTP effettuate al server web.

Spring Boot

Spring Boot permette di poter eseguire le applicazione su un server web integrato, senza dover ricorrere ad altri appositi (come Tomcat, Jetty...). Quindi il metodo main lancia l'intera applicazione web, compreso il server web integrato.

Spring Data JPA

Spring Data JPA, che fa parte della famiglia Spring Data, serve a facilitare l'implementazione di applicazioni che accedono ai dati. Questo framework è stato di rilevante importanza nello sviluppo del software in quanto ha permesso di accedere e salvare dati nel database del server web.

Postman

Postman è un software che permette, tramite interfaccia, di effettuare richieste HTTP a un server web e di visualizzare le risposte che vengono ricevute. Grazie all'interfaccia è possibile visualizzare i dati che vengono trasmessi nel body di un messaggio HTTP (solitamente in formato JSON). È quindi risultato essere uno strumento utile per testare il corretto funzionamento dell'applicazione.

Stoplight

Stoplight è uno strumento che permette di progettare le API, ovvero le interfacce di programmazione delle applicazioni. Viene usato principalmente come interfaccia tra chi sviluppa il front-end e chi il back-end. Infatti, in esso sono contenute tutte le richieste che il client può inviare al server e le rispettive risposte che si aspetta di ricevere da esso.

Git

Git è un software per il controllo di versione di codice. È stato utilizzato principalmente per salvare le versioni del codice scritto e per memorizzare i file del progetto nel cloud (GitHub) in caso di perdita dei file.

PostgreSQL

PostgreSQL è un database relazionale open source che è stato utilizzato nel progetto per memorizzare tutti i dati che necessitavano di essere salvati (utenti, piatti, menù...).

1.5 Architettura a microservizi e monolitica

Queste due categorie descrivono l'architettura adottata nello sviluppo delle applicazioni odierne.

Microservizi

L'architettura a microservizi, spesso preferibile a quella monolitica, è caratterizzata dal fatto che l'applicazione è costituita da una serie di servizi indipendenti, ciascuno incentrato su un particolare aspetto, che comunicano tra loro per scambiarsi informazioni. La separazione dei componenti rende semplice il mantenimento delle applicazioni. I servizi si sviluppano e distribuiscono in modo indipendente, sono più facili da mantenere, correggere e aggiornare.

Meccanismi di comunicazione in un'architettura a microservizi

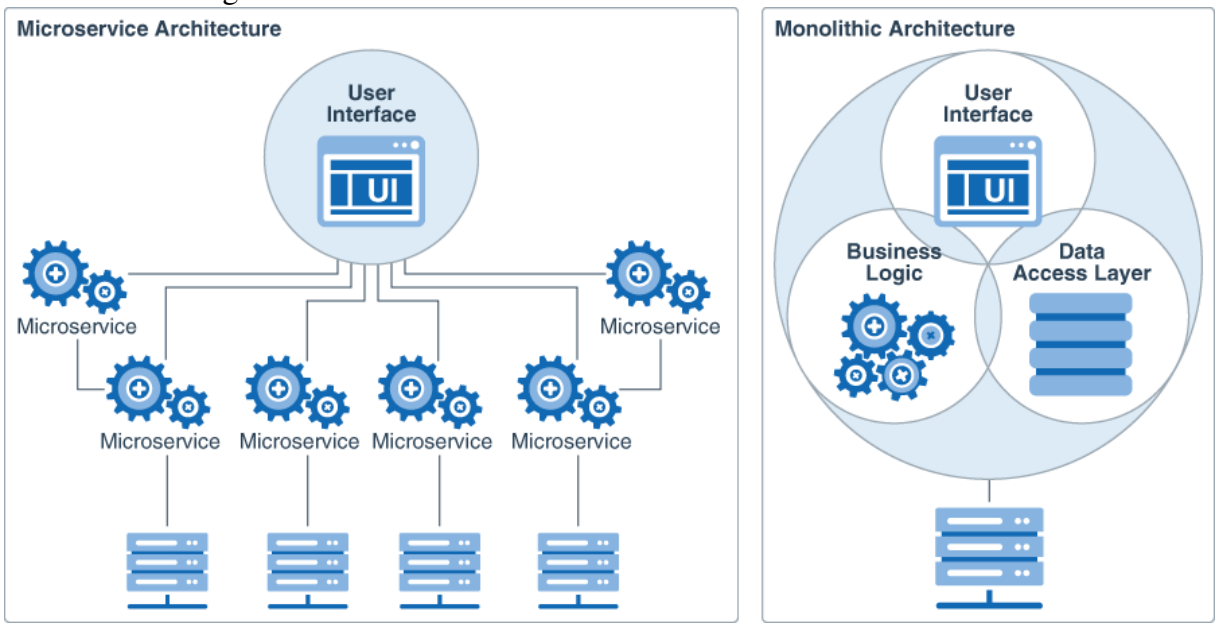
In un'architettura a microservizi, la comunicazione tra il client e l'applicazione, e anche la comunicazione tra i vari microservizi, differisce da quella di un'applicazione con architettura monolitica. Esiste un componente creato apposta per gestire la comunicazione tra client-microservizi e microservizi-microservizi che è l'API Gateway. Esso si occupa di gestire le richieste del client, chiamando tutti i microservizi necessari e inviando l'output al client. Anche i servizi che devono comunicare tra loro si interfacciano tramite messaggi HTTP poiché, non risiedendo nello stesso processo, non possono effettuare normali chiamate di metodi come avviene nelle applicazioni monolitiche.

Monolite

Le applicazioni monolitiche vengono sviluppate come un'unica singola entità, sono facili da implementare poiché hanno una sola base di codice, tipicamente raccolte in un unico project che vengono distribuite all'interno di un unico pacchetto. Questo tipo di architettura è adatta a applicazioni di piccole dimensioni, soggette a pochi cambiamenti. Quando però l'applicazione

cresce in dimensione e complessità diventa difficile muoversi nel progetto rapidamente in fase di sviluppo, test e implementazione.

Figura 1.2: Architettura a microservizi vs architettura monolitica



Architettura adottata nel progetto

Poiché il tempo per realizzare l'applicazione era contenuto, e poiché le dimensioni del progetto erano relativamente ridotte, si è deciso, assieme al tutor, di adottare un'architettura monolitica per lo sviluppo dell'applicazione.

Capitolo 2

Descrizione dello stage

2.1 Modalità di lavoro

Durante il percorso di tirocinio, sebbene la sede dell'azienda fosse a Padova, la maggior parte del lavoro è stato svolto da remoto. L'azienda adotta un metodo di lavoro che consiste nel trovarsi in sede circa una volta a settimana per confrontarsi sul lavoro svolto, sui progressi fatti e sulle difficoltà riscontrate. Il restante del tempo lavorativo viene svolto in autonomia e, qualora ci si trovi in difficoltà, si possono utilizzare gli strumenti di comunicazione da remoto per risolvere dubbi o problemi.

2.2 Pianificazione del percorso

Lo stage ha avuto una durata di 8 settimane e ha previsto lo svolgimento di 300 ore lavorative. Il primo mese è stato impiegato per lo studio delle tecnologie necessarie allo sviluppo del progetto, il secondo per la progettazione e l'implementazione dell'applicazione. La ripartizione settimanale delle attività è stata la seguente:

- **Prima Settimana - Formazione**
  - Incontro con persone coinvolte nel progetto per discutere i requisiti e le richieste relativamente al sistema da sviluppare.
  - Ripasso Java Standard Edition
  - Studio teorico dell'architettura a microservizi
- **Seconda Settimana - Formazione**
  - Studio Spring Boot.

  - Studio Spring Core.
  - Studio Maven.
  - Studio Git.
  - Studio protocollo HTTP
  - Studio messaggi JSON

- **Terza Settimana - Formazione**
  - Studio Spring Data JPA.
  - Studio tool Stoplight.

- **Quarta Settimana - Progettazione**
  - Inizio progettazione progetto Sushilab.

- **Quinta Settimana - Implementazione**
  - Implementazione back-end dei servizi per il progetto SushiLab

- **Sesta Settimana - Implementazione**
  - Implementazione back-end dei servizi per il progetto SushiLab

- **Settima Settimana - Implementazione**
  - Implementazione back-end dei servizi per il progetto SushiLab

- **Ottava Settimana - Test**
  - Test dell'applicazione

Capitolo 3

Analisi dei requisiti del progetto

In questo capitolo vengono analizzati i requisiti che l'applicazione deve soddisfare elencando i casi d'uso.

3.1 Dettagli del progetto

Il progetto Sushilab nasce per poter gestire numerosi ristoranti, quindi anche gestori diversi, permettendo ai clienti di utilizzare la stessa applicazione per ordinare in diversi locali. La versione (semplificata) del progetto a cui ho lavorato, invece, presuppone che ci sia un solo ristorante: in questo modo non bisogna preoccuparsi di riconoscere in quale ristorante si trovano i clienti quando decidono di creare una nuova sessione.

Quando un cliente ha creato una sessione avrà a disposizione un codice univoco di 7 caratteri alfanumerici e un codice QR che permetteranno, a chi siede nello stesso tavolo, di poter accedere alla sessione appena creata. La scelta sull'utilizzo del metodo per unirsi alla sessione (tramite scansione o tramite digitazione) è a discrezione dell'utente finale.

Il menù, dove il cliente può navigare, comprenderà una lista di piatti, ognuna dei quali ha: nome, numero, foto, prezzo, valutazione media, valutazione personale dell'utente, il numero di porzioni, ingredienti, data dell'ultima volta che è stato ordinato e note per il cameriere.

I clienti avranno la possibilità di inserire degli ingredienti a cui sono allergici in una blacklist.

È prevista, sia per i clienti che per i gestori, una finestra di login, una di registrazione, e una di recupero password.

Il gestore deve poter aggiungere, rimuovere e modificare i piatti. Questi ultimi verranno aggiunti al/ai menù che il gestore creerà. Ognuno di questi menù avrà delle fasce orarie di

validità (es: Lunedì dalle 9:30 alle 18:00).

3.2 Casi d'uso

3.2.1 Attori principali

Gli attori che possono interagire con l'applicazione sono i seguenti:

- **Utenti non autenticati**: coloro che non si sono autenticati o registrati nell'applicazione web.

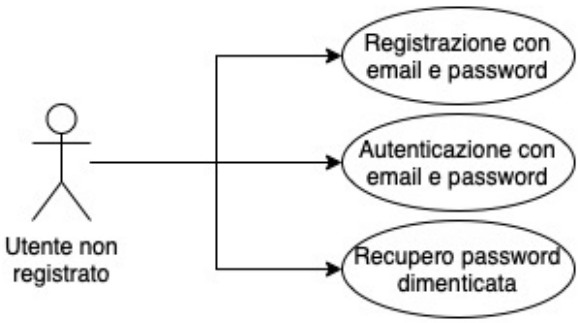
- **Clienti del ristorante**: coloro che, dopo essersi autenticati, risultano essere dei clienti all'applicazione web (coloro che generalmente sfogliano il menù e ordinano).

- **Gestori del ristorante**: coloro che, dopo essersi autenticati, risultano essere dei gestori del ristorante all'applicazione web (coloro che generalmente creano i menù e vi aggiungono i piatti).

3.2.2 Utenti non autenticati

Di seguito viene raffigurato uno schema che rappresenta i principali casi d'uso per gli utenti non autenticati che utilizzano l'applicazione.

Figura 3.1: Schema di un utente non autenticato



Di seguito vengono elencati i casi d'uso dell'applicazione da parte degli attori.

Registrazione

L'utente inserisce i campi email e password per registrarsi, se l'email è già presente nel database, il sistema, informerà l'utente che l'email è già in uso, altrimenti lo inserirà come nuovo utente registrato.

Autenticazione

L'utente si autentica inserendo le credenziali, se una di esse o entrambi sono errate, il sistema informerà l'utente che le credenziali non sono valide, altrimenti procederà facendo loggare l'utente.

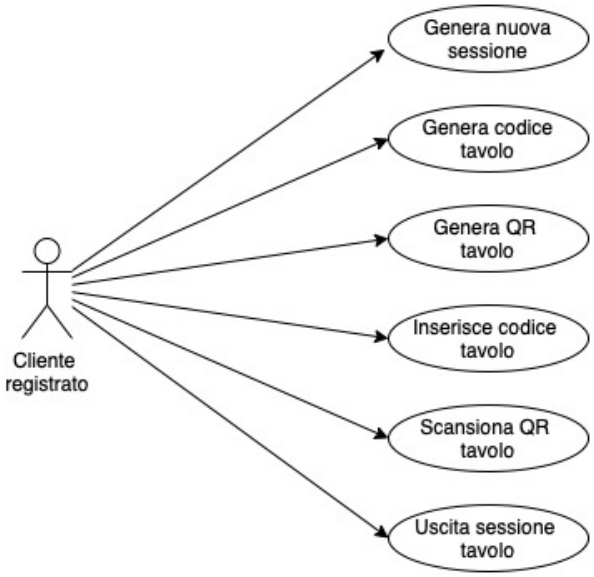
Recupero password

L'utente che non ricorda la password associata al suo account può procedere con il recupero della password, che consiste nell'inserimento della propria email, e successivamente, nell'inserimento di un codice di conferma inviato all'email inserita. Una volta inserito il codice, l'applicazione chiederà all'utente di inserire e confermare la nuova password.

3.2.3 Clienti del ristorante

Di seguito viene raffigurato uno schema che rappresenta i principali casi d'uso, legati alla sessione del tavolo, per i clienti autenticati che utilizzano l'applicazione.

Figura 3.2: Schema di un cliente che gestisce le sessioni



Genera nuova sessione

Il cliente, tramite applicazione, genera una nuova sessione (un tavolo) con la quale può aggiungere altre persone e ordinare piatti dal menù.

Genera codice tavolo

Una volta creata la sessione, il cliente può decidere di generare il codice del tavolo tramite il quale altri clienti possono unirsi alla sessione.

Genera QR tavolo

Il cliente, in alternativa al codice del tavolo, può generare un codice QR che potrà essere scansionato (tramite fotocamera) dagli altri clienti per unirsi alla sessione.

Inserisci codice tavolo

Un cliente inserendo il codice del tavolo si può unire alla sessione già creata di un altro utente.

Scansiona QR tavolo

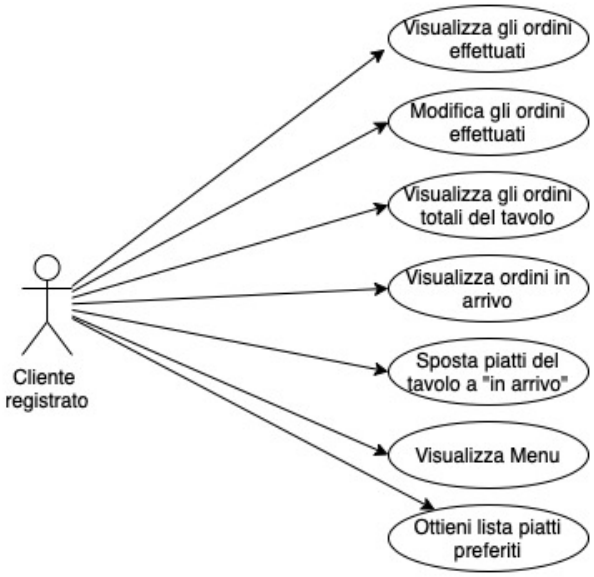
Un cliente, scansionando il codice QR del tavolo generato da un altro cliente, si può unire alla sessione.

Uscita sessione tavolo

Un cliente che è dentro una sessione può decidere di chiuderla, eliminando di conseguenza gli ordini del tavolo.

Di seguito viene raffigurato uno schema che rappresenta i principali casi d'uso, legati alla gestione degli ordini dei clienti.

Figura 3.3: Schema di un cliente che gestisce gli ordini



Visualizza gli ordini effettuati

Vengono visualizzati gli ordini effettuati fino a quel momento dal cliente.

Modifica gli ordini effettuati

Il cliente può aggiungere, rimuovere o modificare i piatti ordinati.

Visualizza gli ordini totali del tavolo

Il cliente può visualizzare gli ordini totali del tavolo: quelli ordinati da lui e da tutti gli altri presenti nella stessa sessione.

Visualizza ordini in arrivo

Il cliente può visualizzare i piatti ordinati in arrivo.

Sposta i piatti del tavolo a "in arrivo"

Il cliente può decidere di spostare tutti i piatti del tavolo in "in arrivo".

Modifica black list

Il cliente può modificare la black list: lista degli ingredienti a cui il cliente è allergico.

Visualizza menù

Il cliente può visualizzare il menù del ristorante, filtrato per gli allergeni che il cliente ha impostato precedentemente. Se un piatto contiene almeno un allergene presente tra quelli che il cliente ha indicato, il piatto non viene fatto visualizzare al cliente.

Ottieni lista piatti preferiti

Il cliente può visualizzare la lista dei piatti che ha precedentemente contrassegnato come preferiti.

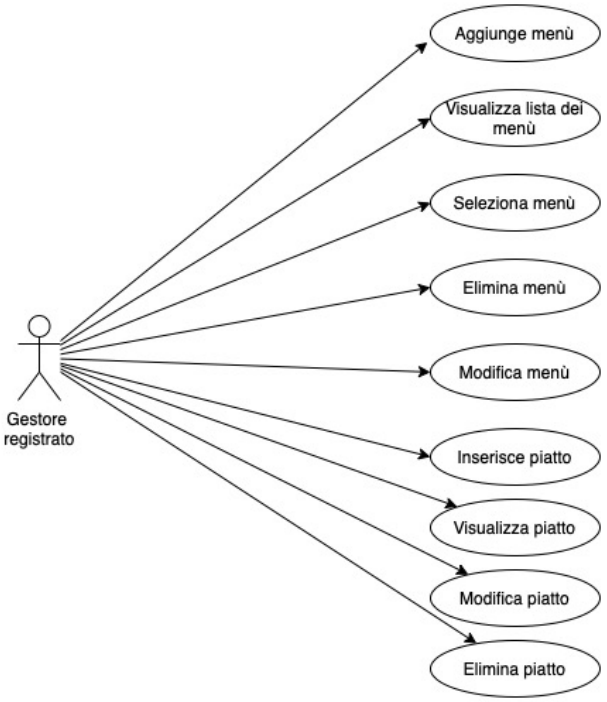
Modifica valutazione di un piatto

Il cliente può assegnare una valutazione a un piatto compresa tra 0 e 5.

3.2.4 Gestori del ristorante

Di seguito viene raffigurato uno schema che rappresenta i principali casi d'uso, legati alla gestione del ristorante, da parte dei gestori che utilizzano l'applicazione.

Figura 3.4: Schema di un gestore che gestisce il ristorante



Aggiungi menù

Il gestore può creare un nuovo menù con un proprio nome. Un menù è composto da una lista di piatti, ciascuno dei quali appartiene a una determinata sezione (antipasti, primi, secondi), e ognuno dei quali ha un proprio prezzo. Prima di creare un menù con i piatti, questi ultimi devono essere già stati creati.

Visualizza lista menù

Il gestore può visualizzare la lista completa di tutti i menù che ha creato precedentemente.

Seleziona menù

Il gestore può selezionare un menù per modificarlo o semplicemente visualizzarlo.

Elimina menù

Il gestore può eliminare un menù.

Modifica menù

Il gestore può apportare modifiche a un menù precedentemente creato.

Inserisci piatto

Il gestore può creare un nuovo piatto specificandone il nome, gli ingredienti, il prezzo, la foto ecc.

Visualizza piatto

Il gestore può visualizzare un piatto precedentemente creato.

Modifica piatto

Il gestore può modificare un piatto precedentemente creato.

Elimina piatto

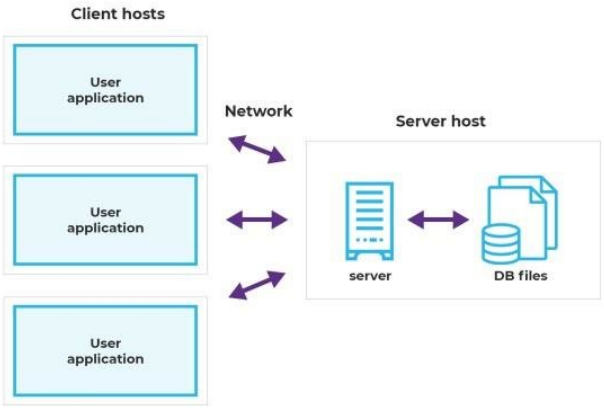
Il gestore può eliminare un piatto precedentemente creato.

Capitolo 4

Analisi delle richieste HTTP

In questo capitolo verranno analizzate tutte le richieste HTTP che il server deve gestire per poter garantire un corretto funzionamento dell'applicazione.

Figura 4.1: Schema del funzionamento della comunicazione client - server



Esempio di una richiesta inviata dal client al server per effettuare un login:

POST http://localhost:3000/login

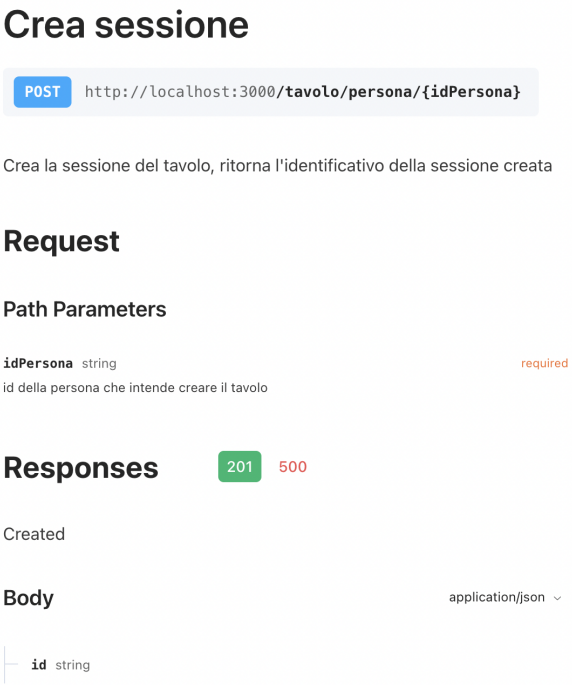
{ "email" : "example@email.it",

"password" : "password1234" }

4.1 Lista delle richieste da gestire

Il programma stoplight serve a creare uno schema delle richieste che vengono effettuate, con i dettagli su come vengono fatte (l'url, il metodo, le variabili passate, il contenuto del body) e sulle risposte che ci si aspetta (lo stato della risposta HTTP e il contenuto del body).

Figura 4.2: Interfaccia di stoplight



Come si può notare nella figura 4.2, la richiesta utilizza un metodo di tipo **POST**, viene passata nell'url la variabile **idPersona,** e le risposte che ci si aspetta di ricevere sono: **201 Created** , con un body di tipo *application/json* contentente un *id* di tipo *string* o **500 Internal Server Error**.

Di seguito verranno elencate tutte le richieste HTTP e le relative risposte che durante il tirocinio sono state sviluppate.

Crea sessione

Richiesta

Url: POST http://localhost:3000/tavolo/persona/idPersona

Metodo: POST.

Variabili del path:

- idPersona: identifica la persona che crea la sessione.

Body: nessuno.

Risposta

Stati http:

- 201 Created: quando viene creata la sessione del tavolo correttamente.

- 500 Internal Server Error: in caso di errori.

Body:

- id: string

Ottieni sessione

Richiesta

Url: GET http://localhost:3000/tavolo/idTavolo

Metodo: GET.

Variabili del path:

- idTavolo: identifica la sessione cercata.

Body: nessuno.

Risposta

Stati http:

- 200 OK: quando viene ritornata la sessione.
- 204 No Content: quando la variabile idTavolo non corrisponde a un id valido.

Body:

tavolo: integer

menu: MenuCompatto

id: integer

nome: string

piatti: array[integer]

Chiudi sessione

Richiesta

Url: DELETE http://localhost:3000/tavolo/idTavolo

Metodo: DELETE.

Variabili del path:

- idTavolo: identifica la sessione da eliminare.

Body: nessuno.

Risposta

Stati http:

- 200 OK: quando viene eliminata correttamente la sessione.
- 405 Method Not Allowed: quando la variabile idTavolo non corrisponde a un id valido.

Body: nessuno

Ottieni gli ordini di una persona

Richiesta

Url: GET http://localhost:3000/tavolo/idTavolo/persona/idPersona

Metodo: GET.

Variabili del path:

- idTavolo: identifica la sessione a cui il cliente è partecipante.
- idPersona: identifica la persona che richiede i propri ordini.

Body: nessuno.

Risposta

Stati http:

- 200 OK: quando vengono ritornati correttamente gli ordini del cliente.
- 405 Method Not Allowed: quando le variabili idTavolo o idPersona non corrispondono a un id valido.

Body:

personali: array[OrdineDettaglio]
piatto: Piatto
id: integer
numero: integer
variante: string
nome: string
prezzo: integer
allergeni: array[string]
ingredienti: array[string]
limite: integer

valutazioneMedia: integer

valutazioneUtente: integer

preferito: boolean

ultimoOrdine: string

popolare: boolean

consigliato: boolean

immagine: string

alt: string

molteplicità: integer

note: array[string]

Modifica gli ordini di una persona

Richiesta

Url: POST http://localhost:3000/tavolo/idTavolo/persona/idPersona

Metodo: GET.

Variabili del path:

- idTavolo: identifica la sessione a cui il cliente è partecipante.
- idPersona: identifica la persona che modifica i propri ordini.

Body:

ordini: array[Ordine]

idPiatto: integer

count: integer

note: string

Risposta

Stati http:

- 200 OK: quando vengono modificati correttamente gli ordini del cliente.
- 405 Method Not Allowed: quando le variabili idTavolo o idPersona non corrispondono a un id valido.

Body: nessuno.

Ottieni gli ordini di un tavolo

Richiesta

Url: GET http://localhost:3000/tavolo/idTavolo/persona/idPersona/ordini

Metodo: GET.

Variabili del path:

- idTavolo: identifica la sessione a cui il cliente è partecipante.
- idPersona: identifica la persona che richiede gli ordini del tavolo.

Body: nessuno.

Risposta

Stati http:

- 200 OK: quando vengono ritornati correttamente gli ordini del tavolo.
- 405 Method Not Allowed: quando le variabili idTavolo o idPersona non corrispondono a un id valido.

Body:

ordini: array[OrdineDettaglio]
piatto: Piatto
id: integer
numero: integer
variante: string
nome: string
prezzo: integer
allergeni: array[string]
ingredienti: array[string]
limite: integer
valutazioneMedia: integer
valutazioneUtente: integer
preferito: boolean
ultimoOrdine: string
popolare: boolean
consigliato: boolean
immagine: string
alt: string

molteplicità: integer

note: array[string]

Ottieni gli ordini in arrivo di una persona

Richiesta

Url: GET http://localhost:3000/tavolo/idTavolo/inarrivo/idPersona

Metodo: GET.

Variabili del path:

- idTavolo: identifica la sessione a cui il cliente è partecipante.
- idPersona: identifica la persona che richiede gli ordini in arrivo.

Body: nessuno.

Risposta

Stati http:

- 200 OK: quando vengono ritornati correttamente gli ordini in arrivo della persona.
- 405 Method Not Allowed: quando le variabili idTavolo o idPersona non corrispondono a un id valido.

Body:

inarrivo: array[OrdineDettaglio]

piatto: Piatto

id: integer

numero: integer

variante: string

nome: string

prezzo: integer

allergeni: array[string]

ingredienti: array[string]

limite: integer

valutazioneMedia: integer

valutazioneUtente: integer

preferito: boolean

ultimoOrdine: string

popolare: boolean

consigliato: boolean

immagine: string

alt: string

molteplicità: integer

note: array[string]

Sposta gli ordini ad in arrivo

Richiesta

Url: POST http://localhost:3000/tavolo/idTavolo/inarrivo/idPersona

Metodo: POST.

Variabili del path:

- idTavolo: identifica la sessione a cui il cliente è partecipante.
- idPersona: identifica la persona che sposta gli ordini del tavolo a in arrivo.

Body: nessuno.

Risposta

Stati http:

- 200 OK: quando vengono spostati correttamente gli ordini del tavolo a in arrivo.
- 405 Method Not Allowed: quando le variabili idTavolo o idPersona non corrispondono a un id valido.

Body: nessuno.

Mergia 2 sessioni di tavolo

Richiesta

Url: POST http://localhost:3000/tavolo/merge/idTavolo

Metodo: POST.

Variabili del path:

- idTavolo: identifica la sessione a cui ci si vuole unire.

Body:

ordini: array[object]

idPiatto: int

count: int

note: string

idPersona: string

Risposta

Stati http:

- 200 OK: quando vengono aggiunti correttamente gli ordini al tavolo indicato.

Body: nessuno

Ottieni menù

Richiesta

Url: GET http://localhost:3000/menu/idMenu/persona/idPersona

Metodo: GET.

Variabili del path:

- idMenu: identifica il menù che si vuole ricevere.
- idPersona: identifica la persona che richiede il menù.

Body: nessuno.

Risposta

Stati http:

- 200 OK: quando viene ritornato correttamente il menù.
- 204 No Content: quando le variabili idTavolo o idPersona non corrispondono a un id valido.

Body:

menu: array[Sezione]

nome: string

piatti: array[Piatto]

id: integer

numero: integer

variante: string

nome: string

prezzo: integer

allergeni: array[string]

ingredienti: array[string]

limite: integer

valutazioneMedia: integer

valutazioneUtente: integer

preferito: boolean

ultimoOrdine: string

popolare: boolean

consigliato: boolean

immagine: string

alt: string

Ottieni la lista della fasce di validità

Richiesta

Url: GET http://localhost:3000/menu/idMenu/fasce

Metodo: GET.

Variabili del path:

- idMenu: identifica il menù.

Body: nessuno.

Risposta

Stati http:

- 200 OK: quando viene ritornata correttamente la lista delle fasce di validità del menù indicato.
- 405 Method Not Allowed: quando la variabile idTavolo non corrisponde a un id valido.

Body:

fasce: array[FasciaOraria]

giorno: string

oraInizio: integer

oraFine: integer

Ottieni lista preferiti

Richiesta

Url: GET http://localhost:3000/menu/idMenu/persona/idPersona/preferiti

Metodo: GET.

Variabili del path:

- idMenu: identifica il menù dal quale si vuole ricevere la lista dei piatti preferiti dall'utente.
- idPersona: identifica la persona che richiede i piatti preferiti.

Body: nessuno.

Risposta

Stati http:

- 200 OK: quando vengono ritornati correttamente i piatti preferiti al cliente.
- 405 Method Not Allowed: quando le variabili idTavolo o idPersona non corrispondono a un id valido.

Body:

preferiti: array[Piatto]
id: integer
numero: integer
variante: string
nome: string
prezzo: integer
allergeni: array[string]
ingredienti: array[string]
limite: integer
valutazioneMedia: integer
valutazioneUtente: integer
preferito: boolean
ultimoOrdine: string
popolare: boolean
consigliato: boolean
immagine: string
alt: string

Ottieni il tema del menù

Richiesta

Url: GET http://localhost:3000/menu/idMenu/stile

Metodo: GET.

Variabili del path:

- idMenu: identifica il menù dal quale si vuole ricevere lo stile.

Body: nessuno.

Risposta

Stati http:

- 200 OK: quando viene ritornato correttamente lo stile del menù.
- 204 No Content: quando la variabile idTavolo non corrisponde a un id valido.

Body:

nome: string

tema: string

Ottieni le informazioni dell'utente

Richiesta

Url: GET http://localhost:3000/utente/idPersona

Metodo: GET.

Variabili del path:

- idPersona: identifica la persona di cui si vuole conoscere le informazioni.

Body: nessuno.

Risposta

Stati http:

- 200 OK: quando vengono ritornate correttamente le informazioni dell'utente.
- 401 Unauthorized: quando la variabile idPersona non corrisponde a un id valido.

Body:

email: string

isGestore: boolean

password: string

Registra utente

Richiesta

Url: POST http://localhost:3000/utente

Metodo: POST.

Variabili del path: nessuna.

Body:

email: string

isGestore: boolean

password: string

Risposta

Stati http:

- 200 OK: quando viene registrato correttamente l'utente.
- 409 Conflict: quando l'email è già presente nel sistema.

Body: nessuno

Modifica stato preferiti

Richiesta

Url: POST http://localhost:3000/utente/idPersona/favorito/idPiatto

Metodo: POST.

Variabili del path:

- idPiatto: identifica il piatto al quale si vuole modificare lo stato preferito.
- idPersona: identifica la persona che intende modificare lo stato preferito del piatto.

Body:

fav: boolean

Risposta

Stati http:

- 200 OK: quando viene modificato correttamente lo stato preferito di un piatto.
- 405 Method Not Allowed: quando le variabili idPiatto o idPersona non corrispondono a un id valido.

Body: nessuno.

Modifica valutazione

Richiesta

Url: POST http://localhost:3000/utente/idPersona/rate/idPiatto

Metodo: POST.

Variabili del path:

- idPiatto: identifica il piatto al quale si vuole modificare la valutazione dell'utente.
- idPersona: identifica la persona che intende modificare la valutazione del piatto.

Body:

rate: integer

Risposta

Stati http:

- 200 OK: quando viene modificata correttamente la valutazione del piatto da parte dell'utente.
- 405 Method Not Allowed: quando le variabili idPiatto o idPersona non corrispondono a un id valido.

Body: nessuno.

Invia email per il recupero della password

Richiesta

Url: POST http://localhost:3000/utente/code

Metodo: POST.

Variabili del path: nessuna

Body:

email: string

Risposta

Stati http:

- 200 OK: quando viene inviata correttamente l'email con il codice di verifica.
- 404 Not Found: quando l'email inserita non è registrata nel sistema.

Body: nessuno.

Verifica codice recupero password

Richiesta

Url: POST http://localhost:3000/utente/verifyCode

Metodo: POST.

Variabili del path: nessuna

Body:

email: string

code: string

Risposta

Stati http:

- 200 OK: quando l'email e il codice inviati dall'utente sono validi.
- 404 Not Found: quando l'email o il codice inviati dall'utente non sono validi.

Body: nessuno.

Reimposta la password dimenticata

Richiesta

Url: POST http://localhost:3000/utente/pass

Metodo: POST.

Variabili del path: nessuna.

Body:

email: string

newpass: string

Risposta

Stati http:

- 200 OK: quando viene modificata correttamente la password dell'utente.
- 406 Not Acceptable: quando la password non è valida.

Body: nessuno.

Esegui il login

Richiesta

Url: POST http://localhost:3000/login

Metodo: POST.

Variabili del path: nessuna.

Body:

email: string

password: string

Risposta

Stati http:

- 200 OK: quando le credenziali dell'utente corrispondono a quelle nel sistema.
- 403 Forbidden: quando la password o l'email sono errati.

Body:

expiresIn: integer

idToken: string

Ottieni la blacklist

Richiesta

Url: GET http://localhost:3000/utente/idPersona/blacklist

Metodo: GET.

Variabili del path:

idPersona: identifica la persona che vuole ottenere la propria blacklist.

Body: nessuno.

Risposta

Stati http:

- 200 OK: quando viene ritornata correttamente la blacklist dell'utente.
- 404 Not Found: quando idPersona non corrisponde a un id valido.

Body:

ingredienti: array[string]

Aggiorna blacklist

Richiesta

Url: POST http://localhost:3000/utente/idPersona/blacklist

Metodo: POST.

Variabili del path:

idPersona: identifica la persona che vuole modificare la propria blacklist.

Body:

ingredienti: array[string]

Risposta

Stati http:

- 200 OK: quando viene modificata correttamente la blacklist dell'utente.
- 405 Method Not Allowed: quando idPersona non è un id valido.

Body: nessuno.

Carica un immagine piatto

Richiesta

Url: POST http://localhost:3000/gestore/idPersona/immagine

Metodo: POST.

Variabili del path:

idPersona: identifica il gestore che intende aggiungere un immagine.

Body: Non implementato

Risposta

Stati http:

- 200 OK: quando viene caricata correttamente l'immagine.
- 403 Forbidden: quando idPersona non identifica un gestore.

Body: Non implementato.

Ottieni la lista piatti

Richiesta

Url: GET http://localhost:3000/gestore/idPersona/piatti

Metodo: GET.

Variabili del path:

idPersona: identifica il gestore che intende ricevere la lista dei piatti.

Body: nessuno.

Risposta

Stati http:

- 200 OK: quando viene ritornata correttamente la lista dei piatti al gestore.
- 403 Forbidden: quando idPersona non identifica un gestore.

Body:

piatti: array[PiattoPreview].

numero: integer

variante: string

numero: integer

nome: string

id: integer

consigliato: boolean

limite: integer

Nuovo piatto

Richiesta

Url: POST http://localhost:3000/gestore/idPersona/piatto

Metodo: POST.

Variabili del path:

idPersona: identifica il gestore che intende aggiungere un nuovo piatto.

Body:

numero: integer

variante: string

nome: string

prezzo: float

immagine: string

alt: string

allergeni: array[string]

ingredienti: array[string]

Risposta

Stati http:

- 200 OK: quando viene aggiunto correttamente il nuovo piatto.
- 403 Forbidden: quando idPersona non identifica un gestore.
- 400 Bad Request: quando i parametri passati non sono validi.

Body: nessuno.

Ottieni piatto

Richiesta

Url: GET http://localhost:3000/gestore/idPersona/piatto/idPiatto

Metodo: GET.

Variabili del path:

idPersona: identifica il gestore che intende ricevere un piatto.

idPiatto: identifica il piatto da ritornare.

Body: nessuno.

Risposta

Stati http:

- 200 OK: quando viene ritornato correttamente il piatto.
- 204 No Content: quando idPiatto non si riferisce ad alcun piatto.

Body:

numero: integer

variante: string

nome: string

prezzo: float

immagine: string

alt: string

allergeni: array[string]

ingredienti: array[string]

Elimina piatto

Richiesta

Url: DELETE http://localhost:3000/gestore/idPersona/piatto/idPiatto

Metodo: DELETE.

Variabili del path:

idPersona: identifica il gestore che intende eliminare un piatto.

idPiatto: identifica il piatto da eliminare.

Body: nessuno.

Risposta

Stati http:

- 200 OK: quando viene eliminato correttamente il piatto.
- 403 Forbidden: quando idPersona o idPiatto non sono validi.

Body: nessuno.

Aggiorna piatto

Richiesta

Url: POST http://localhost:3000/gestore/idPersona/piatto/idPiatto

Metodo: POST.

Variabili del path:

idPersona: identifica il gestore che intende modificare un piatto.

idPiatto: identifica il piatto da modificare.

Body:

numero: integer

variante: string

nome: string

prezzo: float

immagine: string

alt: string

allergeni: array[string]

ingredienti: array[string]

Risposta

Stati http:

- 200 OK: quando viene modificato correttamente il piatto.
- 403 Forbidden: quando idPersona o idPiatto non sono validi.

Body: nessuno.

Ottieni lista menù

Richiesta

Url: POST http://localhost:3000/gestore/idPersona/menu

Metodo: POST.

Variabili del path:

idPersona: identifica il gestore che intende ottenere la lista dei menù.

Body: nessuno.

Risposta

Stati http:

- 200 OK: quando viene ritornata correttamente la lista dei menù.
- 403 Not Acceptable: quando idPersona non identifica un gestore.

Body:

list: array[object]

nome: string

id: integer

Nuovo menù

Richiesta

Url: POST http://localhost:3000/gestore/idPersona/menu

Metodo: POST.

Variabili del path:

idPersona: identifica il gestore che intende creare un nuovo menù.

Body:

name: string

Risposta

Stati http:

- 200 OK: quando viene creato correttamente un nuovo menù.
- 403 Forbidden: quando idPersona non identifica un gestore.

Body:

id: integer

Ottieni menù

Richiesta

Url: GET http://localhost:3000/gestore/idPersona/menu/idMenu

Metodo: GET.

Variabili del path:

idPersona: identifica il gestore che intende ricevere il menù.

idMenu: identifica il menù che si intende ricevere.

Body: nessuno.

Risposta

Stati http:

- 200 OK: quando viene ritornato correttamente il menù.
- 204 No Content: quando idMenu non corrisponde a un menù esistente.
- 403 Forbidden: quando idPersona non identifica un gestore.

Body:

nome: string

menu: array[SezionePreview]

fasce: array[FasciaOraria]

Aggiorna menù

Richiesta

Url: POST http://localhost:3000/gestore/idPersona/menu/idMenu

Metodo: POST.

Variabili del path:

idPersona: identifica il gestore che intende aggiornare il menù.

idMenu: identifica il menù che si intende aggiornare.

Body:

nome: string

menu: array[SezionePreview]

nome: string

piatti: array[PiattoPreview]

numero: integer

variante: string

numero: integer

nome: string

id: integer

consigliato: boolean

limite: integer

fasce: array[FasciaOraria]

giorno: string

oraInizio: integer

oraFine: integer

Risposta

Stati http:

- 200 OK: quando viene aggiornato correttamente il menù.
- 403 Forbidden: quando idPersona non identifica un gestore o idMenu non identifica un menù.

Body: nessuno.

Elimina menù

Richiesta

Url: DELETE http://localhost:3000/gestore/idPersona/menu/idMenu

Metodo: DELETE.

Variabili del path:

idPersona: identifica il gestore che intende eliminare il menù.

idMenu: identifica il menù che si intende eliminare.

Body: nessuno

Risposta

Stati http:

- 200 OK: quando viene eliminato correttamente il menù.
- 404 Not Found: quando idMenu o idPersona non sono identificativi validi.

Body: nessuno.

Capitolo 5

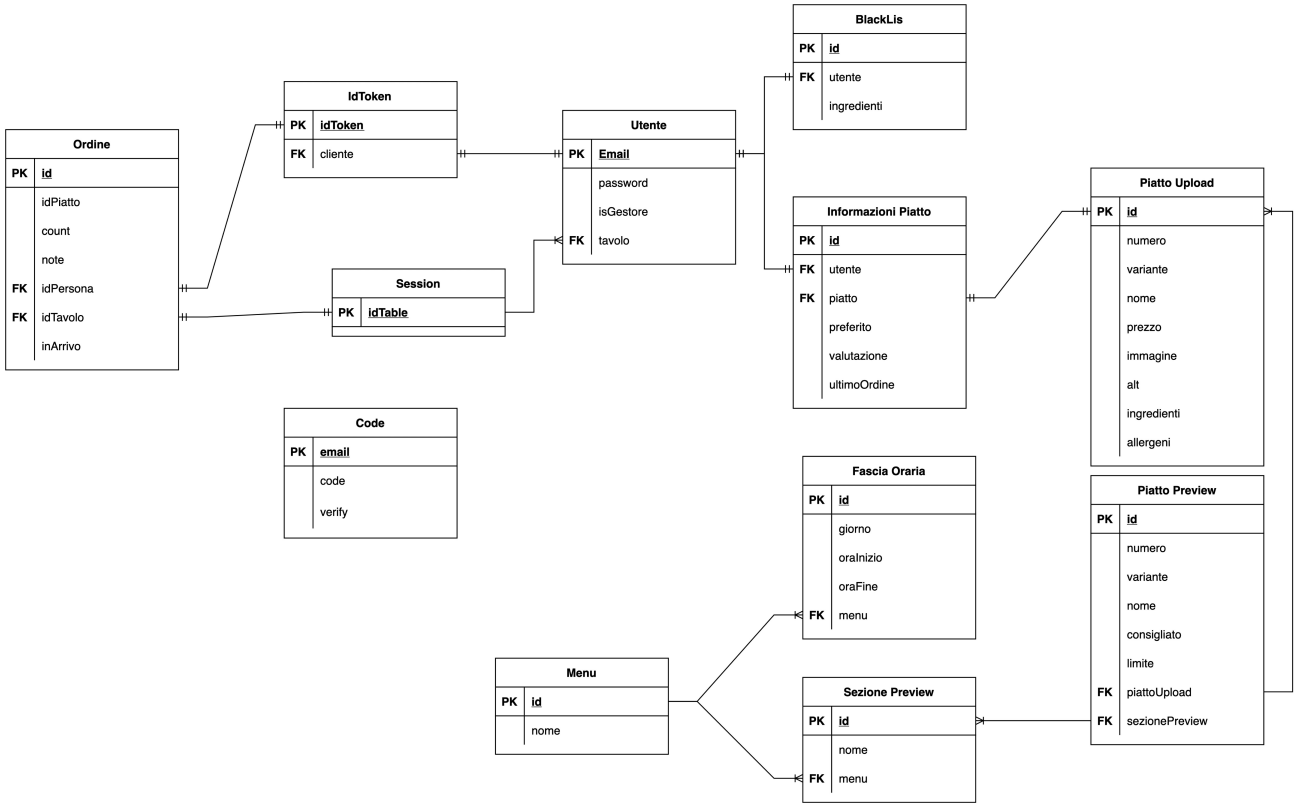
Il Database

L'utilizzo di un database in un progetto back-end è fondamentale per salvare e recuperare i dati dell'applicazione. Nel caso di SushiLab, è risultato fondamentale il salvataggio di dati quali i piatti del ristorante (assieme alle loro informazioni), i menù del ristorante, i dati degli utenti ecc.

5.1 Schema del database

Di seguito viene raffigurato uno schema delle entità che compongono il database e delle relazioni tra loro.

Figura 5.1: Schema ER del database



5.2 Entità del database

Di seguito verranno elencate le entità del database e verrà spiegata la loro utilità nel programma:

Utente

L'utente è caratterizzato da:

- email (String): dev'essere univoca nel database (non possono esistere più utenti registrati con la stessa email).
- password (String).
- isGestore (boolean): identifica se l'utente è un cliente (false) o un gestore (true).
- tavolo (Session, OneToOne): identifica la sessione a cui il cliente si è unito.

Piatto Upload

L'entità piatto upload rappresenta i piatti che il gestore crea per il suo ristorante, ed è caratterizzata da:

- id (Integer): un numero identificativo generato automaticamente.
- numero (Integer).
- variante (String).
- nome (Session).
- prezzo (Float).
- immagine (String): l'url dell'immagine.
- alt (String).
- ingredienti (List⟨String⟩)
- allergeni (List⟨String⟩)

Session

La session è caratterizzata da:

- idTable (Integer): un numero identificativo generato automaticamente.

Menu

Il menù è caratterizzato da:

- id (Integer): un numero identificativo generato automaticamente.
- nome (String).

Fascia Oraria

La fascia oraria si applica a un menù, ma a un menù possono essere applicate più fasce orarie, e indica il/i giorno/i in cui il menù è valido. È caratterizzata da:

- id (Integer): un numero identificativo generato automaticamente.
- giorno (String): può assumere i valori: "Lun", "Mar", "Mer", "Gio", "Ven", "Sab", "Dom", "All", "Week", "End".
- oraInizio (Integer): in minuti dalla mezzanotte.
- oraFine (Integer): in minuti dalla mezzanotte.
- menu (Menu): il menù a cui si applica la fascia oraria.

Sezione Preview

La sezione preview viene creata in automatico alla creazione di un menù e rappresenta la categoria di appartenenza dei piatti (es: antipasti, primi ecc.). È caratterizzata da:

- id (Integer): un numero identificativo generato automaticamente.
- nome (String).
- menu (Menu): il menù a cui fa riferimento la sezione.

Piatto Preview

L'entità piatto preview rappresenta i piatti che il gestore assegna a una sezione:

- id (Integer): un numero identificativo generato automaticamente.
- numero (Integer).
- variante (String).
- nome (String).
- consigliato (boolean).
- limite (Integer).
- piattoUpload (Piatto Upload).
- piattoPreview (Piatto Preview).

Informazioni Piatto

L'entità informazioni piatto serve a salvare le preferenze degli utenti riguardo i piatti ed è caratterizzata da:

- id (Integer): un numero identificativo generato automaticamente.
- utente (Utente).
- piatto (Piatto Upload).
- preferito (boolean).
- valutazione (Integer): valori ammessi da 0 a 5.
- ultimoOrdine (String).

Blacklist

L'entità blacklist serve a salvare gli allergeni degli utenti:

- id (Integer): un numero identificativo generato automaticamente.
- utente (Utente).
- ingredienti (List⟨String⟩).

IdToken

L'idtoken di una persona è un numero generato casualmente che identifica un utente che ha effettuato l'accesso all'applicazione. È caratterizzato da:

- idToken (Integer): un numero identificativo generato casualmente.
- utente (Utente).

Ordine

L'entità ordine serve a salvare gli ordini effettuati dagli utenti. È caratterizzata da:

- id (Integer): un numero identificativo generato casualmente.
- idPiatto (Integer): id del piatto ordinato.
- count (Integer): numero di molteplicità del piatto.
- note (String).
- idPersona (IdToken).
- idTavolo (Session).
- inArrivo (boolean).

Code

L'entità code serve a salvare i codici generati per il recupero della password degli utenti. È caratterizzata da:

- email (String): email dell'utente.
- code (String): codice generato casualmente.
- verify (boolean): true se l'utente ha verificato il codice, false altrimenti.

Capitolo 6

Implementazione del software

In questo capitolo verranno illustrati i principali approcci utilizzati nello sviluppo dell'applicazione.

6.1 Gestione richieste HTTP

Nel seguente esempio verrà mostrata una classe di esempio che gestisce le richieste che arrivano al server con un particolare url. L'url dev'essere composto dall'indirizzo del server (es: localhost:8080) e da un path che inizi con un certo valore (nel codice questo valore sarà dettato dal parametro value). Per esempio vogliamo creare una classe che gestisca le seguenti richieste HTTP il cui path inizia con *clienti*:

- POST localhost:8080/clienti/registrazione
- GET localhost:8080/clienti/lista
- GET localhost:8080/clienti/idCliente

Da notare che l'ultima richiesta passa come parametro nell'url una variabile idCliente che dovrà essere prelevata dal controller.

```
1 @RestController // Indica che la classe gestisce le richieste web.
2 @RequestMapping(value = "clienti") // Le richieste che ascolta hanno path
     che inizia per "utente".
3 public class Controller {
4     @PostMapping(path = "registrazione") // Invoca la funzione se riceve
     una richiesta POST localhost:8080/clienti/registrazione
5     public ResponseEntity<String> registraUtente(){...}
```

```
6     @GetMapping(path = "lista") // Metodo GET e url: localhost:8080/clienti
    /lista
7     public ResponseEntity<String> ottieniListaUtenti(){...}
8     @GetMapping(path = "{idCliente}") // Metodo GET e url: localhost:8080/
    clienti/{idCliente}
9     public ResponseEntity<String> ottieniListaUtenti(@PathVariable
    idCliente){...}
10 }
```

6.2 Gestione risposte HTTP

Dopo aver indirizzato le richieste nei metodi opportuni, è necessario ritornare una risposta al client. Per far ciò, nei metodi chiamati dalle richieste, bisogna ritornare un oggetto di tipo ResponseEntity⟨String⟩. ResponseEntity ha molteplici costruttori, nello sviluppo del software sono stati utilizzati i seguenti:

- ResponseEntity(String body, HttpStatus state): viene ritornata una risposta HTTP contenente lo stato e il body (nel caso dell'applicazione in formato JSON).
- ResponseEntity(HttpStatus state): non viene ritornato il body ma solo lo stato della risposta.

Capitolo 7

Problemi affrontati e sviluppi futuri

Durante il percorso di stage sono emerse diverse problematiche che se non risolte avrebbero rallentato o fermato lo sviluppo del software.

7.1 Stoplight non adatto

Problema

Lo stoplight che era stato generato dai membri della parte front-end del progetto presentava alcune problematiche quando ho dovuto implementare le richieste nella parte back-end del software. Nello specifico, quando un utente voleva effettuare una richiesta al server, esso non mandava alcuna informazione su chi fosse stato a effettuare la richiesta.

Soluzione

Riprogettare lo stoplight di SushiLab modificando le richieste vecchie che non permettevano l'identificazione degli utenti.

Figura 7.1: Esempio di una richiesta vecchia

Figura 7.2: Nuova richiesta con identificativo dell'utente

Facendo inviare all'utente il proprio id nell'url, il server è in grado di riconoscere l'utente e ritornare per esempio i suoi dati o modificare gli ordini ecc.

7.2 Architettura a micro servizi

Problema

Sviluppare un applicazione strutturata a micro servizi richiederebbe un'organizzazione del software complesso e un tempo di sviluppo più lungo rispetto a quanto avuto per il periodo di stage.

Soluzione

Insieme al tutor aziendale si è deciso, poiché il progetto non era di grandi dimensioni, di procedere allo sviluppo dell'applicazione come un unico servizio. Quindi di non sviluppare più micro servizi nonostante la seconda scelta sia la più vantaggiosa in termini di manutenibilità del progetto.

7.3 Sviluppi futuri

Il progetto dovrà affrontare diversi sviluppi e modifiche per essere considerato pubblicabile. Di seguito verranno elencate alcune delle modifiche che secondo me dovrebbero essere apportate:

- Modifica del front-end: In seguito alle modifiche che ho apportato allo stoplight del progetto (quindi al formato delle richieste che il client deve inviare), bisognerà modificare la parte di front-end in modo che implementi le richieste modificate.
- Implementazione delle funzionalità mancanti sia dal front-end che dal back-end: alcune funzionalità presenti nello stoplight non sono state trattate dal back-end (come per esempio la richiesta del tema del menù, o l'aggiunta di un immagine a un piatto), in quanto non ancora chiara la metodologia per implementarle.

- Implementare la sicurezza: al momento l'applicazione non garantisce alcuna integrità e riservatezza dei dati. Un modulo di spring che risolve questo problema è Spring Security.

Capitolo 8

Conclusioni

8.1 Raggiungimento degli obbiettivi

Buona parte degli obbiettivi prefissati all'inizio dello stage sono stati raggiunti con successo. Non è stato possibile testare l'applicazione con il front-end a causa della modifica delle richieste ricevute dal server, ma il lavoro svolto è risultato funzionante ai test effettuati in locale con Postman.

8.2 Analisi del lavoro svolto

Le misure adottate per lavorare in modalità smart working si sono rilevate essere ugualmente efficaci per lo svolgimento dello stage, senza alcuna mancanza di assistenza in quanto ogni dubbio veniva chiarito tramite strumenti di comunicazione come Discord. Il risultato del lavoro svolto in azienda è risultato essere adeguato alle aspettative e, in seguito agli sviluppi futuri, l'applicazione potrà essere resa disponibile ai clienti.

8.3 Valutazione personale

L'esperienza di stage svolta presso SyncLab è stata molto costruttiva e stimolante in quanto ho appreso conoscenza di alcuni strumenti utili per lo sviluppo di applicazioni web, e ho avuto modo di affrontare per la prima volta una realtà lavorativa nell'ambito informatico. I concetti che ho appreso durante l'università si sono rivelati essere utili per il lavoro che ho svolto, senza alcuni di questi non sarei stato in grado di concludere il progetto in tempo.

In conclusione, grazie a questa esperienza ho avuto modo di apprendere concetti nuovi, di rendermi autosufficiente nella gestione del tempo lavorativo, e di introdurmi nel mondo del lavoro.

Glossario

API Application programming interface: un insieme di procedure atte a risolvere uno specifico problema di comunicazione tra diversi computer o tra diversi software o tra diversi componenti di software. 3

applicazione server Software che viene eseguito su un server. 2

back-end Lo sviluppo web back-end è lo sviluppo del software che permette l'effettivo funzionamento delle interazioni con i client. iii, 2, 3, 8, 41, 49, 50

back-end developer Sviluppatore di applicazioni lato back-end. 2

body Il body è il contenitore all'interno del quale si trova la parte visibile di una pagina web. 3, 17, 18, 48

client In una rete informatica ogni computer collegato al server e in grado di scambiare dati con esso. 3, 4, 17, 48, 50

database In informatica con database si indica un insieme di dati strutturati ovvero omogeneo per contenuti e formato memorizzati in un computer. 3, 4, 11, 41, 42

framework Un'architettura logica di supporto sulla quale un software può essere progettato e realizzato spesso facilitandone lo sviluppo da parte del programmatore. 2, 3

front-end Lo sviluppo web front-end è lo sviluppo dell'interfaccia utente grafica di un sito web attraverso l'uso di HTML CSS e JavaScript in modo che gli utenti possano visualizzare e interagire con quel sito web. 2, 3, 49, 50, 53

GitHub GitHub è un servizio di hosting per progetti software. 3

HTTP HyperText Transfer Protocol. 2–4, 8, 17, 18, 47, 48

IDE Un ambiente di sviluppo integrato: un software che in fase di programmazione supporta i programmatori nello sviluppo e debugging del codice sorgente di un programma. 2

Java In informatica è un linguaggio di programmazione ad alto livello orientato agli oggetti e a tipizzazione statica. 2, 7

JSON JavaScript Object Notation: è un formato adatto all'interscambio di dati fra applicazioni client/server. 3, 8, 48

maven In informatica Apache Maven è uno strumento di gestione di progetti software basati su Java e build automation. 8

open source Il software open source è un software per computer rilasciato con una licenza in cui il titolare del copyright concede agli utenti i diritti di utilizzare studiare modificare e distribuire il software e il relativo codice sorgente a chiunque e per qualsiasi scopo. 2, 4

server In informatica computer di elevate prestazioni che in una rete fornisce un servizio agli altri elaboratori collegati detti client. 2, 3, 17, 47, 49, 50, 53

server web In informatica un server web è un'applicazione software che in esecuzione su un server è in grado di gestire le richieste di trasferimento di pagine web di un client tipicamente un web browser. 3

software In informatica si intendono tali il semplice dato o informazione oppure più propriamente le istruzioni di un programma codificate in linguaggio macchina o in linguaggio di programmazione (codice sorgente) memorizzate su uno o più supporti fisici sotto forma di codice eseguibile. 2, 3, 48–50

Bibliografia

[1] *SyncLab*: https://www.synclab.it

[2] *Front-end e back-end*: https://it.wikipedia.org/wiki/Front-end_e_back-end

[3] *Spring framework*: https://www.youtube.com/watch?v=gq4S-ovWVlM

[4] *Spring Data JPA*: https://www.youtube.com/watch?v=8SGI_XS5OPw&t=2689s

[5] *Architettura a microservizi e monolitica*: https://blog.mia-platform.eu/it/da-monolite-a-microservizi-come-far-evolvere-unapplicazione-legacy

[6] *Strumento per creazione diagrammi*: https://www.draw.io/index.html

[7] *Stoplight del progetto*: https://michelezadro.stoplight.io

[8] *Codici di stato HTTP*: https://it.wikipedia.org/wiki/Codici_di_stato_HTTP