Università degli Studi di Padova

DIPARTIMENTO DI MATEMATICA "TULLIO LEVI-CIVITA"

CORSO DI LAUREA IN INFORMATICA

Progettazione e realizzazione di un'applicazione web

Design and realization of a web application

Relatore
Prof. Paolo Baldan

Laureando
Pietro Macrì

Sommario

Questo documento descrive il percorso attraversato dallo studente durante lo stage avvenuto in collaborazione con l'azienda Net Logos. Tale stage, durato poco più di trecento ore, è consistito nelle progettazione e realizzazione di un sito web di un generico ristorante.

L'applicazione web permette all'utente di effettuare prenotazioni dei tavoli del ristorante, ed è incorporata nel sito di cui sopra. Tale sito mostra inoltre le caratteristiche del locale, i piatti, i vini e tutte le peculiarità che possano invogliare il potenziale cliente a recarsi nel posto e consumare un pasto.

Data l'elevata semplicità del lato logico dell'applicazione, grande attenzione è stata dedicata al migliorare il più possibile la visita dell'utente all'interno del sito.

Ringraziamenti

Un sentito grazie da parte mia a tutti coloro che mi sono stati vicini durante questo triennio e che mi hanno supportato, sia in maniera diretta sia indiretta.

Un grazie anche a Net Logos per avermi dato l'opportunità di affrontare un percorso formativo con loro. Sempre in Net Logos ringrazio anche i colleghi, per aver mostrato una disponibilità nei miei riguardi che mai mi sarei aspettato di simile portata.

Indice

Capitolo 1

Introduzione

1.1 Descrizione dell'azienda ospitante

Net Logos nasce nell'anno 2000 come centro di formazione incentrata nell'ambito dell'informatica.

Dal 2003 l'azienda ottiene l'accreditamento della Regione Umbria per l'erogazione di corsi che rilasciano certificazione e qualifica regionale L.845/78 per le macroaree della formazione superiore, continua e permanente.

Nel 2009 Net Logos affianca alla formazione una serie di nuove attività, tra le quali:

- consulenza informatica nel settore Internet;
- sviluppo di nuove piattaforme per la gestione aziendale;
- upgrade di strutture preesistenti;
- installazione di server Windows e Linux;
- creazione di reti Intranet aziendali;
- sviluppo di applicazioni per il web su qualunque piattaforma.

L'azienda è inoltre Testing Center autorizzato Microsoft per il rilascio delle certificazioni M.O.S. (Microsoft Office Specialist) e M.E.C. (Microsoft Educational Center), per i corsi di apprendimento del pacchetto Microsoft Office con docenti certificati e courseware esclusivi Microsoft.

Nel corso del tempo la società ha stretto importanti rapporti di collaborazione, tra i quali è possibile evidenziare – oltre alla già citata Microsoft – l'Università degli Studi di Perugia, in particolar modo nella Facoltà di Economia, e il Polo di Mantenimento delle Armi Leggere, appartenente al Comando Logistico dell'Esercito.

1.2 Sintesi del progetto ed elenco degli obiettivi

Il progetto consiste nell'implementazione di un'applicazione web per la prenotazione di posti per un generico ristorante. Nello specifico, viene chiesto al tirocinante di sviluppare una versione tale che essa possa fungere successivamente da template per una realizzazione più facilitata di applicazioni per eventuali ristoranti clienti futuri.

A seguito di varie discussioni con il tutor aziendale, l'elenco degli obiettivi prefissi per il tirocinio ha subìto un lieve cambiamento rispetto a quanto previsto nel piano di tirocinio. Segue la lista finale degli obiettivi:

- Obbligatori
    - Studio e pratica delle tecnologie e degli strumenti da utilizzare
    - Stesura dell'analisi dei requisiti
    - Progettazione dell'applicazione
    - Implementazione dell'applicazione
- Desiderabili
    - Implementazione di una sezione admin
- Facoltativi
    - Implementazione di una gestione real-time, tramite notifiche push
    - Affiancamento al team operativo aziendale nell'analisi e sviluppo di prodotti più complessi

Il progetto ha come scopo primario quello di avvicinare lo studente al mondo dello sviluppo web, un settore dell'informatica che gode di un'elevata domanda da parte della clientela, e di fornirgli le competenze pratiche necessarie in questo tipo di lavoro, competenze che ovviamente nascono dalle loro corrispettive teoriche apprese nel corso degli studi universitari. Nello specifico il progetto risulta funzionale nell'abituare lo studente alle meccaniche di cooperazione ma allo stesso tempo imporgli una determinata autonomia necessaria per un procedimento che non si ripercuota sulla tabella di marcia; capacità di cooperazione ed autonomia, sebbene siano due pregi agli antipodi fra loro, risultano infatti ugualmente importanti in questo e in altri settori dell'informatica.

Data la limitata funzionalità del prodotto, l'analisi dei casi d'uso è stata di grande facilità. Essi si possono infatti raggruppare in tre macro-categorie: login, prenotazione e operazioni admin. Il progetto, tuttavia, ha compensato una tale semplicità nel lato funzionale dell'applicazione con una maggior cura e dedizione al lato qualitativo del prodotto; questa maggior attenzione alla qualità è facilmente verificabile nella evidente differenza tra le cardinalità dei gruppi di requisiti funzionali e obbligatori, gruppi analizzati più dettagliatamente nel paragrafo §2.3 di questo documento.

Di altissima importanza per il sito è la figura dell'utente: il prodotto è stato infatti strutturato di modo tale da migliorare il più possibile l'approccio dell'utente al sito, dimodoché si favorisca di conseguenza un'eventuale visita futura a quest'ultimo. Queste attenzioni verso l'utente si concretizzano generalmente nello studio della User Experience e della User Interface. Tali aspetti, che verranno analizzati con più cura nel paragrafo §3.3, erano totalmente nuovi allo studente nelle fasi iniziali del progetto; una così scarsa familiarità con essi ha portato ad un'obbligatoria formazione in merito e ad un più lento studio di essi in relazione al prodotto, entrambe attività che hanno contribuito a dilatare la progettazione del front-end in termini di tempo.

Sempre rimanendo nel relativo all'approccio dell'utente, degno di nota è anche il fattore accessibilità. Tale fattore non è stato richiesto direttamente dall'azienda, ma il percorso formativo dello studente ha portato quest'ultimo ad integrarlo tra le analisi da effettuare nello sviluppo, al fine di generare un prodotto qualitativamente valido. La medesima sorte è stata destinata alla SEO, non prevista dall'azienda ma aggiunta dallo studente. L'azienda ha invece esplicitamente richiesto la responsiveness del prodotto; essa è stata gestita interamente grazie a Bootstrap, e ovviamente alle media query del css. Questi tre fattori vengono analizzati con maggiore dettaglio nelle sezioni §4.3, §4.4 e §4.5.

L'azienda ha effettuato la scelta – a mio modesto parere non condivisibile – di non fornire il back-end del prodotto allo studente, ma di accompagnare quest'ultimo nella sua progettazione e nel suo sviluppo. Fortunatamente la gerarchia, la quale conta nove classi, si è rivelata di alta semplicità; tuttavia, nonostante questa elevata semplicità, è stato necessario applicare determinati accorgimenti durante lo sviluppo per migliorarne la manutenibilità.

Il prodotto non è un sito relativo ad un ristorante specifico, ma risulta più una sorta di versione base per un ristorante generico. In questo modo è possibile, quando necessario, applicare poche modifiche per ottenere un sito applicabile a qualsiasi ristorante cliente. Infatti, sebbene quasi sicuramente saranno presenti delle richieste e personalizzazioni specifiche del locale, il prodotto attuale risulta uno scheletro degli eventuali prodotti completi, scheletro che sarà possibile mostrare nell'immediato al cliente per fornirgli un'idea generale del risultato. Inoltre è anche una base di partenza su cui i futuri developer incaricati potranno lavorare.

1.3 Tecnologie e strumenti utilizzati

Segue la lista delle tecnologie e degli strumenti che sono stati utilizzati durante il percorso del tirocinio:

- **TypeScript**: linguaggio di programmazione open source sviluppato di Microsoft; si tratta di un'estensione di JavaScript che prevede un sistema di tipizzazione abbastanza rigido (ma pur sempre aggirabile tramite il tipo generico *any*), che consente un maggior controllo durante la compilazione;
- **HTML5**: linguaggio di markup in cui il developer definisce il contenuto della pagina web; la versione 5 di HTML (acronimo di HyperText Markup Language) è concepita per definire standard funzionali e API;
- **Sass**: linguaggio utilizzato per definire lo stile degli elementi di una pagina web; Sass (che sta per Syntactically Awesome Style Sheets) è un'estensione del linguaggio CSS (acronimo di Cascading Style Sheets) a cui aggiunge, tra le altre, regole di annidamento e un più fluido controllo dei componenti;
- **Browser**: i browser web utilizzati per verificare di passo in passo i progressi del lavoro sono stati Goggle Chrome, Microsoft Edge e Mozilla Firefox; saltuariamente è stato controllato il lavoro anche applicando il tema scuro di Chrome, per evitare che utenti che lo utilizzano possano trovarsi a disagio con i contrasti;
- **Figma**: editor di grafica vettoriale utilizzato per la creazione del prototipo del sito e per l'analisi della User Experience (UX) e della User Interface (UI);
- **Visual Studio Code**: l'IDE utilizzato maggiormente per questo progetto, nonché uno dei più diffusi; l'intero lato front-end è stato sviluppato su questo editor;
- **StarUML**: strumento che permette di disegnare diagrammi che seguano i paradigmi dell'ingegneria software, usato nel corso del tirocinio per la creazione del diagramma di classi relativo al back-end;
- **IntelliJ IDEA**: IDE apposito per la programmazione in Java; dato che il back-end dell'applicazione è stato sviluppato in questo linguaggio, si è optato per tale editor per favorire la stesura del codice;

- **PostgreSQL**: sistema di gestione di database strutturato ad oggetti; sebbene la mole di dati da gestire fosse esigua, si è preferito utilizzare questo DBMS (DataBase Management System) per simulare maggiormente una situazione reale, dove la quantità di dati immagazzinati è nettamente superiore;
- **pgAdmin**: applicazione C++ open source che consente di amministrare con semplicità database di PostgreSQL;
- **Git**: sistema di versionamento utilizzato per condividere file e progressi fra tirocinante e colleghi dell'azienda;
- **GitHub**: servizio di hosting per progetti software che permette un uso più intuitivo rispetto all'utilizzo di comandi di linea per il caricamento in git e una maggiore visibilità dei file o delle cartelle caricate;
- **Bootstrap**: framework che facilita lo sviluppo di un layout reponsivo delle pagine web;
- **Angular**: framework open source per lo sviluppo di applicazioni web; il linguaggio di programmazione usato è TypeScript, a differenza del JavaScript della precedente versione AngularJS;
- **Angular Material**: toolkit che velocizza il processo di sviluppo fornendo componenti già pronti e che rispettano i principi del material design (ovvero ogni elemento visualizzato deve essere ben visibile, raffigurativo e finalizzato);
- **Angular CLI**: strumento a linea di comando che permette di creare con rapidità moduli, servizi e componenti;
- **Postman**: piattaforma che permette progettazione, creazione e test di API.

1.4 Breve descrizione del risultato

Il risultato ottenuto al termine del tirocinio è un sito che consta di cinque pagine web, quattro di pubblico accesso e una riservata agli amministratori, che simulano il potenziale sito internet di un generico ristorante. Le pagine pubbliche sono curate sotto il punto di vista di stile, di UX e di UI; esse presentano inoltre un header e un footer in comune, i quali permettono rispettivamente una navigazione rapida e chiara e la comunicazione all'utente di informazioni utili (tra cui contatti e orari del ristorante).

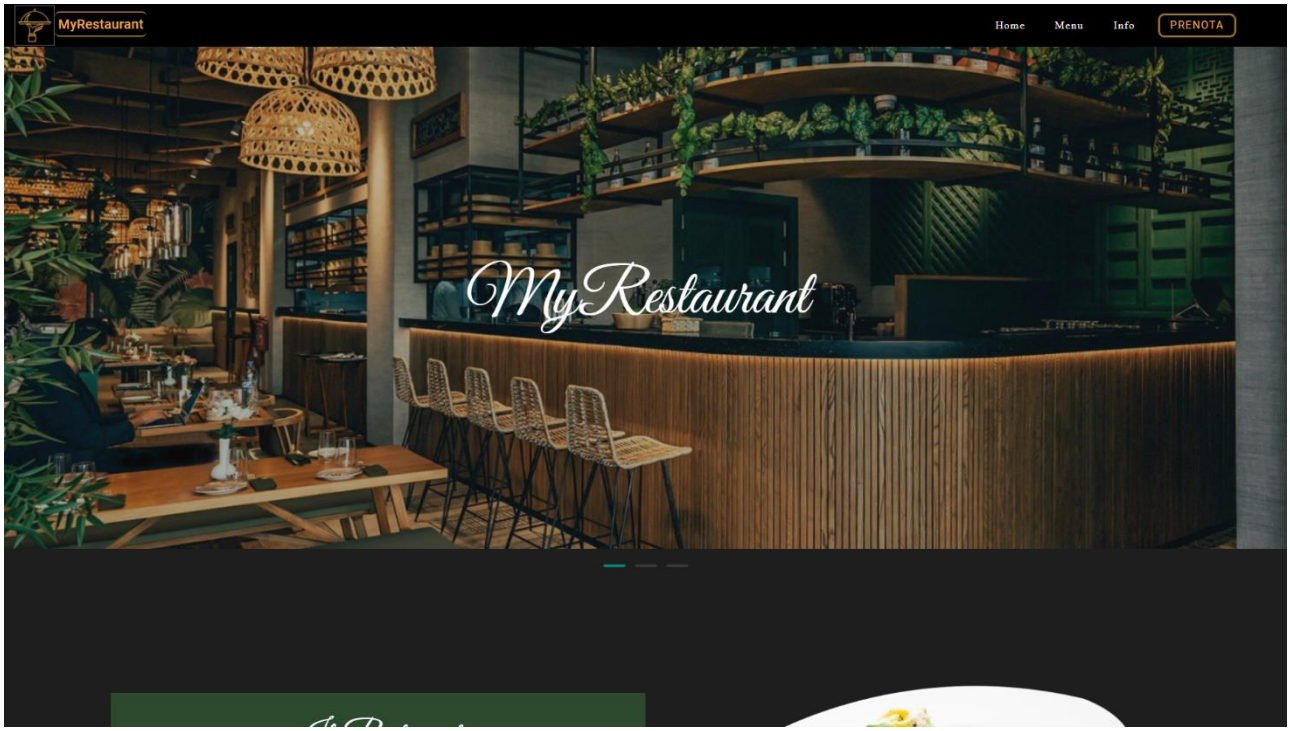

Immagine 1: homepage del sito

La pagina riservata all'amministrazione del sito, invece, è più basilare: non è curata sotto l'aspetto UX, UI o stile, perché l'amministratore non presenta tali necessità. Gli unici campi di elevata importanza per l'amministratore sono la comodità e la velocità di inserimento, modifica o cancellazione dei dati, a cui è stata dunque prestata la dovuta attenzione: la pagina è strutturata affinché una qualsiasi operazione sui dati memorizzati sia rapida e diretta, minimizzando il numero di iterazioni che l'amministratore deve effettuare per compiere le operazioni.

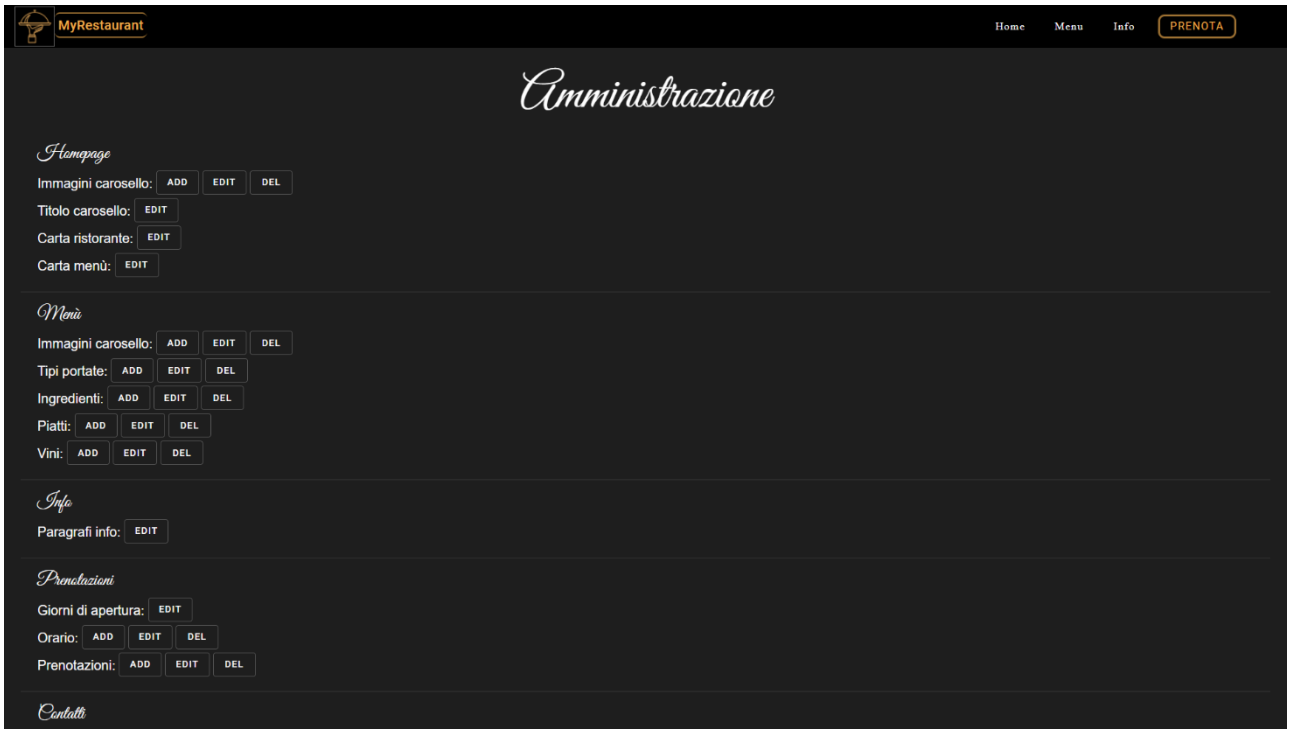

Immagine 2: pagina amministrazione

1.5 Struttura del documento di relazione

Al termine di questo capitolo di introduzione inizia l'effettiva parte di relazione del tirocinio. Il presente documento, il cui scopo è quello di riportare con la massima chiarezza possibile suddetta relazione, è stato strutturato in modo tale che segua cronologicamente le fasi del percorso di tirocinio del laureando.

L'ordine dei capitoli del documento di relazione sarà dunque il seguente:

2. **Analisi dei requisiti**: questo capitolo riassume i passi di accordo tra tirocinante e tutor aziendale circa ciò che il prodotto deve o non deve presentare, simulazione di un accordo tra l'azienda e un eventuale cliente; tali passi sono le fasi primarie del percorso, nelle quali vengono stabilite univocamente le richieste del tutor e i compiti del tirocinante, parallelismo con le richieste del cliente e i compiti del team di sviluppo;
3. **Progettazione**: questo capitolo analizza le fasi di lavoro che precedono la stesura effettiva del codice, ma che sono comunque fondamentali per dare uno scheletro al prodotto, scheletro sul quale poi il team di sviluppo va a fare riferimento per la creazione del prodotto effettivo;
4. **Sviluppo**: questo capitolo tratta la fase vera e propria di sviluppo del prodotto tramite la stesura del codice ad esso relativo;
5. **Conclusioni**: questo ultimo capitolo sintetizza il risultato del percorso di tirocinio e le opinioni del tirocinante.

L'ultima pagina del documento è invece dedicata all'elenco delle fonti.

Capitolo 2

Analisi dei requisiti

2.1 Funzionalità dell'applicazione

L'utenza che l'applicazione prevede è rappresentabile tramite i seguenti gruppi:

- **utente generale**: rappresenta un generico utente che visita il sito;
- **amministratore**: rappresenta il ristoratore o chi per egli si occupa della gestione del sito web del locale; è un sottogruppo dell'utente generale.

L'utente generale può navigare liberamente tra le pagine del sito, fatta eccezione per quella riservata all'amministrazione, ed effettuare prenotazioni di tavoli. L'amministratore può effettuare tutte le operazioni disponibili all'utente, dato che è di fatto un tipo di utente, ed ha inoltre accesso alla pagina di amministrazione, tramite la quale può gestire con comodità le prenotazioni e i tipi di dati mostrati all'utenza (ad esempio i piatti presenti nel menù, l'orario di apertura del ristorante, eccetera).

Venga da qui chiamato per comodità "utente cliente" qualsiasi elemento del gruppo utente generale che non faccia parte del di esso sottogruppo di amministratori.

È bene notare che la funzione di login non è necessaria per l'utente cliente. Si è dunque valutato di non mostrare bottoni o link che portino alla pagina di login, così da accedere unicamente tramite aggiunta all'URL del sito della desinenza "/login". Per istruire il gruppo di amministratori – il cui numero, in linea di massima, è presumibilmente assai inferiore a quello degli utenti clienti – su come effettuare il login, procedura che non è intuitiva data la mancanza di elementi nel sito che lo rappresenti, è stato valutato sufficiente inserire nel manuale d'uso che verrà fornito al ristorante le istruzioni necessarie.

Similmente al login, non è stata considerata necessaria l'implementazione del logout, in quanto questa sarebbe accessibile eventualmente solo all'amministratore, il quale non presenta nessun bisogno di cambiare account e, di conseguenza, di effettuare il logout. Al termine della sessione avviene un logout automatico.

L’eliminazione di login e logout è stata una scelta improntata su un’analisi di pro e contro, ma va sottolineato che non è una decisione definitiva: non appena si desidererà estendere tali funzioni anche agli utenti clienti, sarà sufficiente aggiungere pochi elementi nelle pagine che indirizzino ad esso. Ovviamente sarà però necessaria anche una nuova analisi dei casi d’uso.

Segue la lista delle operazioni eseguibili nella sezione di amministrazione, divise in base al contesto:

- **Homepage**:
  - aggiunta, modifica e cancellazione di immagini al carosello principale;
  - modifica del titolo del carosello;
  - modifica della card relativa al ristorante (titolo e testo);
  - modifica della card relativa al menù (titolo e testo).
- **Menù**:
  - aggiunta, modifica e cancellazione di immagini al carosello;
  - aggiunta, modifica e cancellazione di tipi di portate;
  - aggiunta, modifica e cancellazione di ingredienti;
  - aggiunta, modifica e cancellazione di piatti;
  - aggiunta, modifica e cancellazione di vini.
- **Info**:
  - modifica del contenuto della pagina info.
- **Prenotazioni**:
  - modifica dei giorni d’apertura del ristorante;
  - aggiunta, modifica e cancellazione degli orari a cui si può prenotare[I];
  - aggiunta, modifica e cancellazione di prenotazioni.
- **Contatti**:
  - modifica del cellulare del ristorante;
  - modifica del telefono del ristorante;
  - modifica della partita IVA del ristorante;
  - modifica dell’indirizzo del ristorante;
  - modifica dell’indirizzo Facebook del ristorante;
  - modifica dell’indirizzo Instagram del ristorante.

- **Orari**:
  - modifica dell'orario di apertura relativo al pranzo;
  - modifica dell'orario di chiusura relativo al pranzo;
  - modifica dell'orario di apertura relativo alla cena;
  - modifica dell'orario di chiusura relativo alla cena.

(I): La prenotazione è stata strutturata in modo che l'utente abbia determinati orari disponibili per la prenotazione fissi all'interno di un certo range. Questo per uniformare le prenotazioni agli orari maggiormente richiesti in generale dai clienti, ovvero ad intervalli di quarti d'ora. Di default gli orari sono dunque impostati a ":00", ":15", ":30" e ":45", con l'aggiunta inoltre di un range di 90 minuti di distanza tra l'ultimo orario disponibile per la prenotazione e l'orario di chiusura del ristorante, ma tali impostazioni sono modificabili secondo le richieste del locale.

2.2 Use case

Di fondamentale importanza per il progetto è stata la stesura della lista di casi d'uso del sito. Segue ora l'elenco degli use case e ad essi relative immagini.

2.2.1 UC1: Login

Attori primari: utente cliente.

Descrizione: l'utente cliente inserisce le credenziali per accedere come amministratore.

Precondizioni: l'utente cliente si interfaccia con il dialog per il login.

Scenario principale:

1. l'utente inserisce lo username;
2. l'utente inserisce la password;
3. l'utente clicca sul tasto di conferma.

Scenario alternativo: l'utente inserisce credenziali non valide (UC1.1: Login fallito).

Postcondizioni: l'utente entra nella pagina di amministrazione.

2.2.2 UC1.1: Login fallito

Attori primari: utente cliente.

Descrizione: l'utente cliente inserisce credenziali non valide durante il login.

Precondizioni: l'utente cliente si inserisce credenziali non valide durante il login.

Scenario principale: viene visualizzato un messaggio di errore.

Postcondizioni: l'utente è conscio del mancato login e del motivo che l'ha causato.

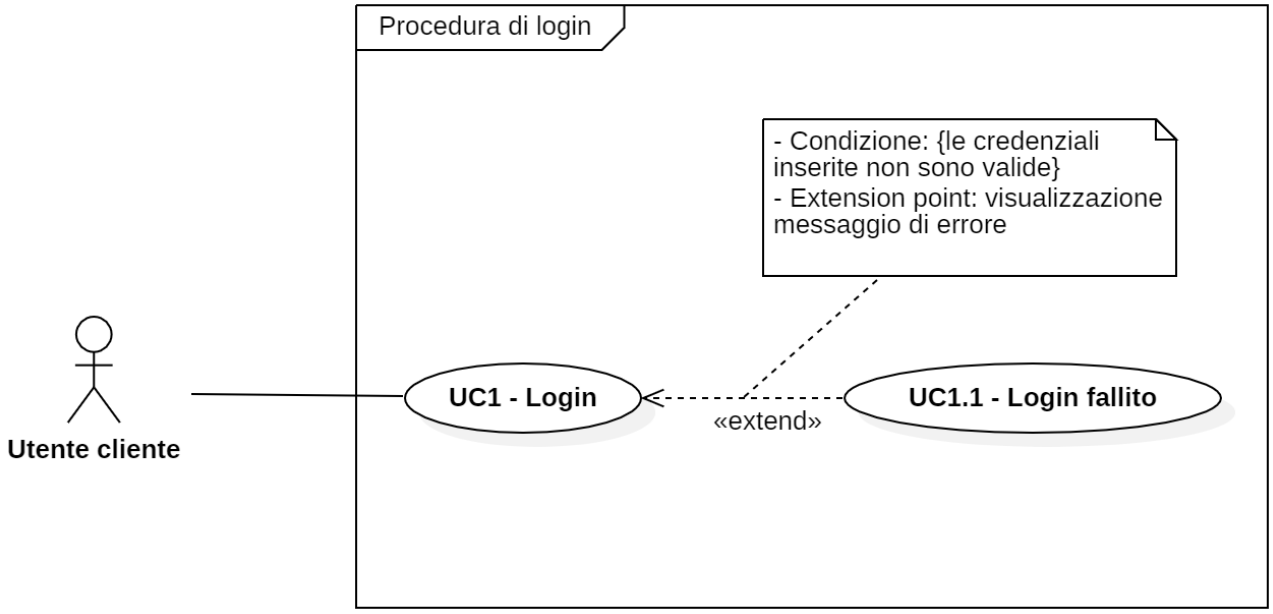

Immagine 3: UC1 e relativi

2.2.3 UC2: Prenotazione

Attori primari: utente generale.

Descrizione: l'utente tenta di effettuare una prenotazione.

Precondizioni: l'utente si interfaccia con il form di prenotazione.

Scenario principale:

1. l'utente inserisce la data di prenotazione;
2. l'utente inserisce l'orario di prenotazione;
3. l'utente inserisce il nome;
4. l'utente inserisce il cognome;
5. l'utente inserisce un recapito telefonico;
6. l'utente inserisce il numero di posti al tavolo;
7. l'utente clicca sul tasto di conferma.

Scenari alternativi:

- l'utente non inserisce tutti i campi o i campi inseriti non sono validi (UC2.1: Form prenotazione non valido);
- la prenotazione non va a buon fine (UC2.2: Prenotazione fallita).

Postcondizioni: l'utente effettua la prenotazione, che viene registrata nel sistema, e riceve una notifica di successo della prenotazione.

2.2.4 UC2.1: Form prenotazione non valido

Attori primari: utente generale.

Descrizione: l'utente tenta di effettuare una prenotazione ma non compila correttamente il form.

Precondizioni: il form di prenotazione non è valido e l'utente ha cliccato sul tasto di conferma del form.

Scenario principale: i campi non validi assumono una colorazione rossa.

Postcondizioni: l'utente intuisce quali campi hanno impedito la prenotazione.

2.2.5 UC2.2: Prenotazione fallita

Attori primari: utente generale.

Descrizione: l'utente tenta di effettuare una prenotazione, compila correttamente il form ma la chiamata non va a termine.

Precondizioni: il form di prenotazione è valido, l'utente ha cliccato sul tasto di conferma del form e la chiamata non va a termine.

Scenario principale: un messaggio di errore viene mostrato all'utente.

Postcondizioni: l'utente sa che la prenotazione non è avvenuta e sa che l'errore non è dovuto ad una sua colpa.

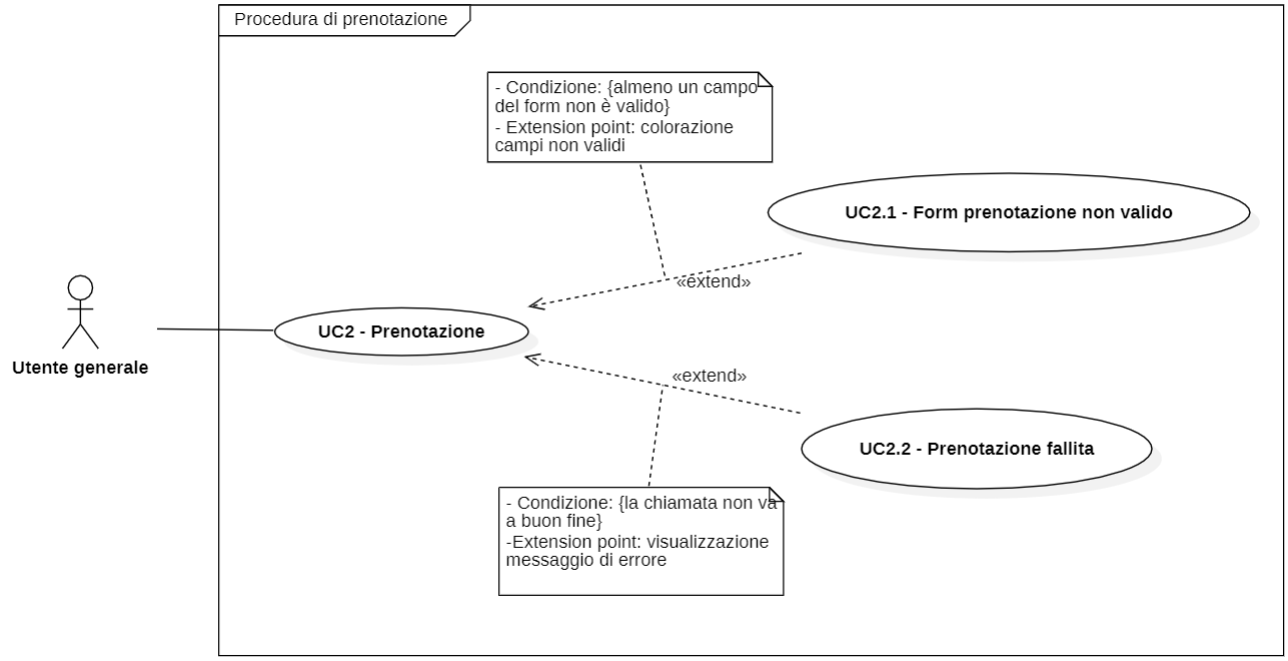

Immagine 4: UC2 e relativi

2.2.6 UC3: Operazione admin

Attori primari: admin.

Descrizione: l'admin esegue un'operazione nella pagina di amministrazione.

Precondizioni: l'admin si interfaccia con il form che desidera compilare.

Scenario principale:

1. l'admin inserisce tutti i campi obbligatori richiesti ed eventuali facoltativi;
2. l'admin clicca sul tasto di conferma.

Scenari alternativi:

- l'admin non inserisce tutti i campi o i campi inseriti non sono validi (UC3.1: Form admin non valido);
- l'operazione non va a buon fine (UC3.2: Operazione fallita).

Casi figli:

- UC3.3: Aggiunta;
- UC3.4: Modifica;
- UC3.5: Eliminazione.

Postcondizioni: l'admin effettua l'operazione, che modifica i dati memorizzati in database, e riceve una notifica di successo dell'operazione.

2.2.7 UC3.1: Form admin non valido

Attori primari: admin.

Descrizione: l'admin tenta di effettuare un'operazione nella pagina di amministrazione, ma non compila correttamente il form.

Precondizioni: il form non è valido e l'admin ha cliccato sul tasto di conferma relativo.

Scenario principale: i campi non validi assumono una colorazione rossa.

Postcondizioni: l'admin intuisce quali campi hanno impedito la prenotazione.

2.2.8 UC3.2: Operazione fallita

Attori primari: admin.

Descrizione: l'admin tenta di effettuare un'operazione nella pagina di amministrazione, compila correttamente il form ma la chiamata non va a termine.

Precondizioni: il form è valido, l'admin ha cliccato sul tasto di conferma e la chiamata non va a termine.

Scenario principale: un messaggio di errore viene mostrato all'admin.

Postcondizioni: l'admin sa che l'operazione non è avvenuta e sa che l'errore non è dovuto ad una sua colpa.

2.2.9 UC3.3: Aggiunta

Attori primari: admin.

Descrizione: l'admin esegue un'aggiunta nella pagina di amministrazione.

Precondizioni: l'admin si interfaccia con il form di aggiunta che desidera compilare.

Scenario principale:

1. l'admin inserisce tutti i campi obbligatori richiesti ed eventuali facoltativi;
2. l'admin clicca sul tasto di conferma.

Scenari alternativi: gli scenari alternativi sono i medesimi del caso padre UC3: Operazione admin.

Postcondizioni: l'admin effettua l'inserimento del nuovo dato, che viene memorizzato in database, e riceve una notifica di successo dell'operazione.

2.2.10 UC3.4: Modifica

Attori primari: admin.

Descrizione: l'admin esegue un'operazione di modifica nella pagina di amministrazione.

Precondizioni: l'admin si interfaccia con il form di modifica che desidera compilare.

Scenario principale:

1. l'admin inserisce un identificativo del dato che vuole cambiare (posizione in una lista, nome, ID numerico o qualsiasi altro valore che identifichi un elemento dai suoi simili);
2. l'admin compila almeno un altro dei campi del form;
3. l'admin clicca sul tasto di conferma.

Scenari alternativi:

- l'admin non compila correttamente nessuno dei campi non-identificativi nel form (UC3.4.1: Campi modifica non validi);
- il campo identificativo non trova riscontro nel database (UC3.4.2: ID modifica non trovato).

Postcondizioni: l'admin effettua la modifica del dato, che viene modificato in database, e riceve una notifica di successo dell'operazione.

2.2.11 UC3.4.1: Campi modifica non validi

Attori primari: admin.

Descrizione: l'admin indica quale elemento modificare ma non la modifica da apportare.

Precondizioni: l'admin clicca sul tasto di conferma di un form di modifica senza aver inserito correttamente un campo non identificativo.

Scenario principale: tutti i campi non identificativi assumono una colorazione rossa.

Postcondizioni: l'admin capisce di non aver inserito alcun campo di modifica del dato.

2.2.12 UC3.4.2: ID modifica non trovato

Attori primari: admin.

Descrizione: il campo identificativo del form di modifica non trova alcun riscontro nel database.

Precondizioni: l'admin clicca compila correttamente il form di modifica, clicca il tasto di conferma e il valore inserito nel campo identificativo non è presente nel database.

Scenario principale: un messaggio di errore avvisa l'admin che nessun dato nel database ha l'identificativo uguale a quello inserito nel form.

Postcondizioni: l'admin è conscio della mancata modifica e della mancata presenza del dato nel database.

2.2.13 UC3.5: Eliminazione

Attori primari: admin.

Descrizione: l'admin esegue un'operazione di eliminazione nella pagina di amministrazione.

Precondizioni: l'admin si interfaccia con il form di eliminazione che desidera compilare.

Scenario principale:

1. l'admin inserisce un identificativo del dato che vuole eliminare;
2. l'admin clicca sul tasto di conferma.

Scenari alternativi: gli scenari alternativi sono i medesimi del caso padre UC3: Operazione admin.

Postcondizioni: l'admin effettua la modifica del dato, che viene modificato in database, e riceve una notifica di successo dell'operazione.

2.2.14 UC3.5.1: ID eliminazione non trovato

Attori primari: admin.

Descrizione: il campo identificativo del form di eliminazione non trova alcun riscontro nel database.

Precondizioni: l'admin clicca compila correttamente il form di eliminazione, clicca il tasto di conferma e il valore inserito nel campo identificativo non è presente nel database.

Scenario principale: un messaggio di errore avvisa l'admin che nessun dato nel database ha l'identificativo uguale a quello inserito nel form.

Postcondizioni: l'admin è conscio della mancata eliminazione e della mancata presenza del dato nel database.

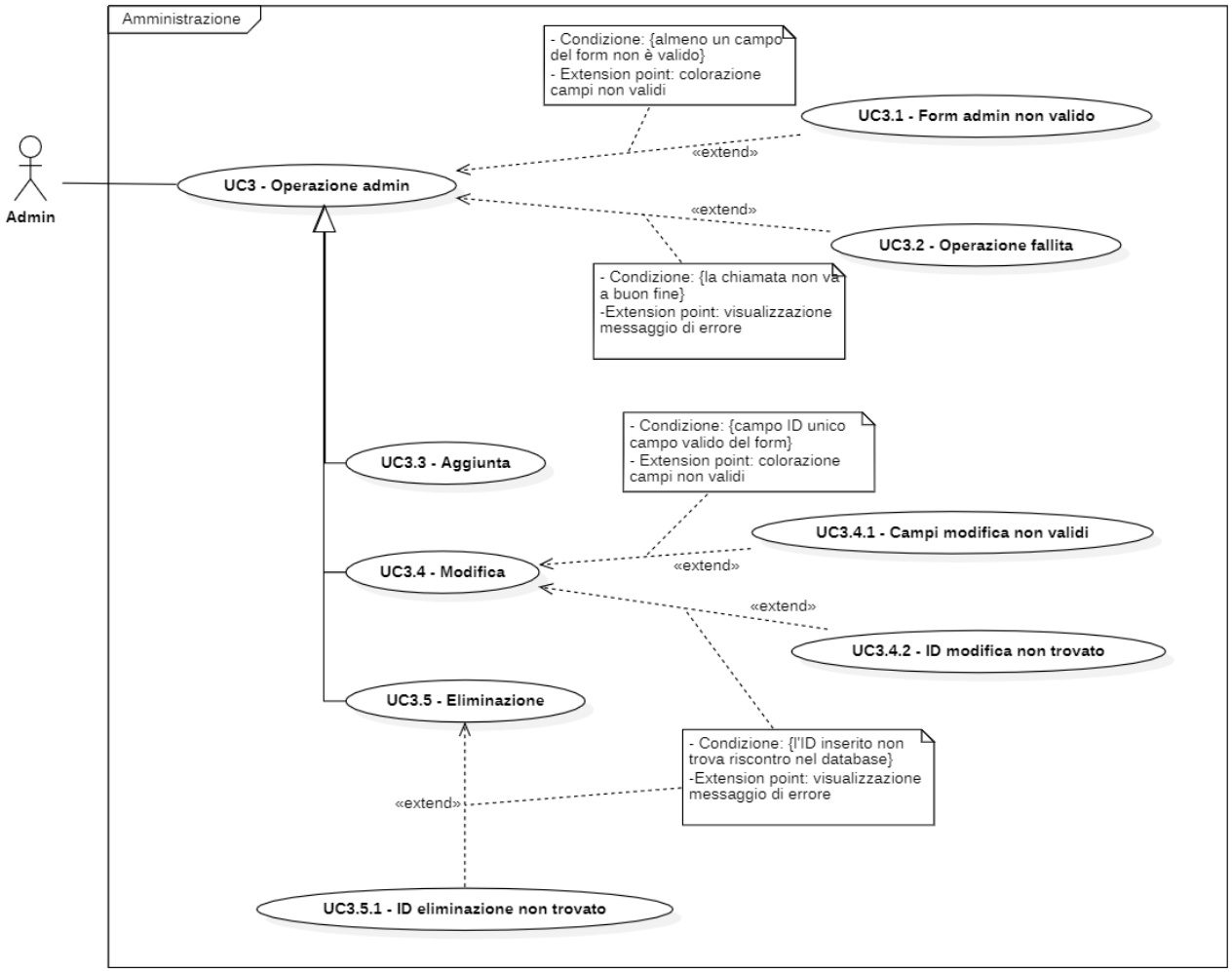

Immagine 5: UC3 e relativi

2.2.15 Parentesi sul numero di use case identificati

Si noti che è possibile identificare numerosi casi figli degli use case di aggiunta, modifica ed eliminazione (rispettivamente UC3.3, UC3.4 e UC3.5), in quanto i form nella pagina admin sono altrettanto numerosi. Tuttavia un'analisi di tali casi figli non porterebbero alcun valore aggiunto significativo, ragion per cui si è deciso di non approfondirli. Le relative funzionalità sono comunque elencate nella sezione del documento che analizza le funzionalità del sito, ovvero in §2.1.

2.3 Tracciamento requisiti

A seguito delle analisi dei requisiti effettuata con l'azienda e analisi degli use case previsti per il progetto sono stati tracciati i requisiti con le relative importanze e fonti.

I requisiti possono assumere diversa importanza. In particolare si possono individuare i seguenti livelli (in ordine decrescente di importanza):

- Obbligatorio: requisito fondamentale e necessario per la completezza del prodotto;
- Desiderabile: requisito non obbligatorio ma la cui presenza comporterebbe un notevole valore aggiunto al prodotto;
- Facoltativo: requisito non obbligatorio e la cui presenza comporta un valore aggiunto lieve.

I requisiti possono inoltre essere classificati in base alla loro tipologia, ovvero:

- Funzionali: requisiti legati alle funzionalità del sistema;
- Qualitativi: requisiti legati a metriche e standard atti a migliorare la qualità del prodotto;
- Prestazionali: requisiti legati alle prestazioni del prodotto (non presenti in questo progetto);
- Vincolo: requisiti legati a fattori legali o a vincoli imposti dal cliente.

Nei seguenti paragrafi si tracciano i requisiti.

2.3.1 Requisiti funzionali

ID	Importanza	Descrizione	Fonte
RF1	Desiderabile	Un utente cliente deve poter accedere al ruolo di admin se in possesso delle credenziali	UC1
RF2	Obbligatorio	Un utente generale deve poter effettuare una prenotazione nella sezione apposita	UC2
RF3	Obbligatorio	Un eventuale errore nella compilazione di un form da parte di un utente generale deve essere indicato in maniera chiara ed evidente	UC2.1, UC2.2, UC3.1, UC3.2, UC3.4.1, UC3.4.2, UC3.5.1

RF4	Desiderabile	Un admin deve poter interagire con tutti i dati presenti in database	UC3 e ad esso relativi
RF5	Facoltativo	Al momento di un'avvenuta prenotazione, una notifica viene inviata al cellulare indicato nel processo di prenotazione	Azienda

2.3.2 Requisiti qualitativi

ID	Importanza	Descrizione	Fonte
RQ1	Desiderabile	Un utente generale deve poter arrivare alla pagina che desidera visitare in massimo 3 click	Azienda
RQ2	Obbligatorio	Su ogni pagina non amministrativa deve essere presente un elemento ben visibile che permetta di tornare immediatamente alla homepage del sito	Azienda
RQ3	Obbligatorio	Il codice deve essere caricato in una repository di GitHub (deve dunque essere versionabile via Git)	Azienda
RQ4	Obbligatorio	Per lo sviluppo del front-end del prodotto si deve usare il framework Angular (e dunque il codice va scritto in TypeScript)	Azienda
RQ7	Obbligatorio	Per la creazione di componenti Angular si deve utilizzare quelli presenti in Angular Material	Azienda
RQ6	Desiderabile	Per le comunicazioni con il back-end del prodotto si usi il linguaggio Java	Azienda
RQ7	Desiderabile	Per la gestione del database si usi PostgreSQL	Azienda
RQ8	Obbligatorio	Ogni pagina del sito deve essere responsiva	Azienda
RQ9	Obbligatorio	Ogni elemento iterativo presente in una qualsiasi pagina del sito deve: essere ben visibile, indicare chiaramente ciò che fa, eseguire il suo scopo correttamente	Azienda
RQ10	Desiderabile	Il sito deve essere ben presentabile su tutti i browser accordati in precedenza	Azienda
RQ11	Facoltativo	La sezione admin deve essere sufficientemente curata, ma possibilmente evitando animazioni	Azienda

2.3.3 Requisiti di vincolo

ID	Importanza	Descrizione	Fonte
RV1	Obbligatorio	I contenuti del sito devono essere salvati in database	Azienda
RV2	Obbligatorio	Le prenotazioni effettuate dagli utenti devono essere salvate in database	Azienda

Capitolo 3

Progettazione

3.1 Distribuzione ore prevista

Segue un resoconto della distribuzione delle ore di lavoro prevista nel programma redatto precedentemente all'inizio del tirocinio.

Durata (in ore)	Descrizione attività
70	Gruppo 1: Formazione iniziale sulle tecnologie
15	Introduzione all'azienda e definizione degli obiettivi
15	Stesura della documentazione
40	Training sugli strumenti di sviluppo
70	Gruppo 2: Teoria di base del framework
20	Single page application
20	Two-way data bindings
30	Styling
80	Gruppo 3: Applicazione web per l'integrazione CRUD dei dati di prenotazione
10	Tipi di base di dati
50	Sviluppo applicazione front-end
10	Test unitari
10	Stesura manuale utente e documentazione
80	Gruppo 4: Applicazione web per l'interrogazione via API per la prenotazione
10	Stesura dello swagger
50	Sviluppo applicazione front-end per interrogazione servizi REST
10	Test unitari e di integrazione
10	Stesura manuale utente e documentazione servizi
TOT: 300	

3.2 Studio strumenti e framework

3.2.1 Angular

Angular 2+, meglio noto semplicemente come Angular, è un framework open source ideato per lo sviluppo di applicazioni web. Dispone di licenza MIT (una licenza di software libero creata dal Massachusetts Institute of Technology)[(1)]. È stato sviluppato principalmente da Google e il suo primo rilascio è avvenuto nel 2016[(2)].

Angular è stato totalmente riscritto rispetto a AngularJS, il che ha portato ad una non compatibilità fra le due versioni. Come già accennato in precendeza, Angular usa TypeScript come linguaggio di programmazione, una versione tipizzata e più rigida del JavaScript usato dal corrispettivo AngularJS (da cui, intuitivamente, il nome)[(3)].

Angular offre al team di programmazione una serie di componenti, nota come Angular Material, che velocizza la creazione di pagine web. Angular può inoltre lavorare in cooperazione con Bootsrap, il quale è più incentrato sul layout degli elementi, per rendere la pagina responsiva con facilità; ciononostante si noti che è in fase di sviluppo[(4)].

Il vantaggio di questo framework è che permette di non renderizzare elementi non visibili nel sito. Gli altri framework, infatti, renderizzano indistintamente tutti gli elementi del sito e solo successivamente valutano se mostrarli o meno a schermo in base alla logica del codice. Ciò comporta un ritardo nella generazione della pagina, per quanto lieve esso possa essere. Con un numero successivamente alto di elementi tale ritardo può diventare significativo e inficiare negativamente sull'esperienza di navigazione del sito da parte dell'utente.

A seguire cenni sull'architettura di Angular, ricavati dalla guida ufficiale di Angular[(5)]. I blocchi base di Angular sono i componenti, i quali sono organizzati in moduli: i moduli NgModules raggruppano frammenti di codice collegati logicamente in gruppi funzionali, e un insieme di moduli forma un'applicazione. Un'applicazione presenta sempre un modulo radice, necessario per l'avvio della stessa, e solitamente molteplici moduli aggiuntivi. I componenti definiscono le viste, le quali sono gruppi di elementi visibili a schermo che Angular può selezionare e modificare in base alla logica e ai dati del programma. I componenti usano servizi, i quali offrono specifiche funzionalità non collegate direttamente alle viste. Moduli, componenti e servizi sono classi che usano decoratori, similmente ad alcuni elementi HTML ma in maniera più dinamica; ciò permette sia una gestione più

dinamica dei dati tra i vari elementi di Angular, sia una dependency injection facilitata. Angular offre inoltre un servizio di routing, il quale definisce i percorsi di navigazione tramite le viste.

3.2.2 Bootstrap

Bootstrap è un framework open source ideato per la gestione del layout di elementi all'interno di una pagina web. È stato sviluppato presso Twitter in modo tale che uniformasse i vari componenti che ne realizzavano l'interfaccia web[(6)].

Dalla versione 2.0, Bootstrap permette di creare strutture di layout altamente responsive, andando dunque ad estendere con facilità l'accesso all'applicazione anche a dispositivi di dimensioni diverse rispetto al classico 1920x1080 px.

La completezza e chiarezza della documentazione di Bootstrap[(7)] sono sufficientemente famose nel mondo dello sviluppo web.

3.2.3 Back-end

Il back-end è stato strutturato in linguaggio Java, un linguaggio di programmazione orientato agli oggetti e di alto livello. È stato scelto questo linguaggio nello specifico perché è appositamente progettato per essere il più indipendente possibile dall'hardware che lo esegue: ciò è possibile tramite una doppia fase di esecuzione, prima tramite compilazione in bytecode (un linguaggio intermedio tra il linguaggio macchina e un linguaggio di programmazione) e poi tramite interpretazione da parte di una Java Virtual Machine[(8)]. Questa peculiarità porta a prestazioni leggermente inferiori rispetto ad altri linguaggi object-oriented; tuttavia, non essendoci vincoli di prestazione ed essendo la differenza prestazionale praticamente insignificante a causa delle ridotte dimensioni dell'applicazione, si è ugualmente optato per tale linguaggio.

3.2.4 Database

Il database è stato creato tramite PostgreSQL, un sistema di gestione di dati open source. Non vi è una ragione specifica per la quale sia stato scelto questo sistema rispetto ad altri; semplicemente l'azienda utilizza con maggiore frequenza tale DBMS, il che porta i colleghi ad essere più esaustivi nelle risposte ad eventuali chiarimenti richiesti dal laureando.

Per l'applicazione, l'accesso ai dati del database è possibile grazie ad API appositamente create. Tali API sono state progettate, create e testate grazie a Postman.

3.3 Analisi User Experience e User Interface

3.3.1 Definizioni di UX e UI

User Experience (UX) e User Interface (UI) sono due concetti di elevatissima importanza nell'ambito dello sviluppo web. Sebbene siano simili tra loro, e spesso erroneamente confusi l'uno con l'altro, sono diversi: la User Interface è l'interfaccia stessa visibile all'utente, l'insieme di tutti gli elementi visualizzati in una pagina e di tutte le relazioni (specialmente di tipo spaziale) presenti fra essi; per User Experience si intende invece l'esperienza che un utente ha mentre visita il sito, ovvero il grado di naturalezza e intuitività che l'utente sperimenta nel corso della navigazione.

È bene dunque che il team di programmazione affronti un'analisi per ognuna di queste caratteristiche prima di procedere con la stesura del codice; la prassi prevede che le due analisi vengano effettuate in parallelo, così da evitare che le falle individuate da una influiscano eccessivamente sull'altra, ma ciò che è fondamentale è il completamento di entrambe.

3.3.2 Risultati analisi UX e UI sul sito

Seguono le liste degli accorgimenti apportati al sito a seguito dello studio di User Experience e User Interface effettuato dallo studente con l'accompagnamento di una figura specializzata.

UX:

- header e footer presenti in ogni pagina;
- tutti i link presenti nell'header;
- collegamenti importanti evidenziati;
- contrasto non troppo forte e piacevole;
- font semplici e diretti, solo i titoli possono essere decorati;
- funzionalità ed espressività di ogni elemento visualizzato in pagina;
- animazioni brevi, semplici e possibilmente non lineari (tipicamente cubiche di Bezier);
- form chiari ed espliciti;
- eventuali errori ben evidenti;
- presenza di immagini in quanto supporto al testo e non sostituzione di esso;
- evidente differenza tra elementi interattivi e non.

UI:

- sufficiente distanza fra gli elementi;
- chiara indicazione di fine pagina;
- visibilità di ogni elemento visualizzato in pagina;
- testi ben leggibili;
- elementi disposti secondo uno schema logico;
- chiara indicazione della presenza di elementi non visualizzati (e.g. nei caroselli);
- allineamento dei testi centrale o giustificato;
- disposizione degli elementi in modo tale da evitare la generazione da parte del browser della barra di scorrimento orizzontale;
- eventuali separazioni logiche ben marcate.

Simili accorgimenti favoriscono la navigazione dell'utente nel sito e non lo scoraggiano dal visitare nuovamente il sito.

3.4 Studio di una gerarchia di classi per il back-end

Per quanto riguarda i contenuti salvati in database, si può facilmente notare che intercorrono relazioni fra essi; un esempio piuttosto ovvio è la relazione tra menù e piatti, come anche tra piatto e ingredienti. È stato dunque necessario progettare una gerarchia delle classi per dare al back-end una struttura logica e semplice e, soprattutto, più manutenibile.

Segue una lista di tutte le classi individuate e lo schema della gerarchia:

- **Ristorante**: questa classe rappresenta ogni locale appartenente ad un'eventuale catena di ristoranti; contiene i contatti e gli indirizzi del locale in sé; il sito prevede di default l'esistenza di un unico ristorante;
- **Info**: rappresenta ogni paragrafo delle informazioni che riassumono il relativo ristorante; nel loro insieme formano informazioni che possano descrivere al meglio il ristorante al lettore, in caso ciò sia necessario;
- **Immagine**: classe dall'elevata semplicità, composta unicamente da un URL necessario al sito per individuare l'immagine e da una caption che ne fornisce una descrizione per aumentare l'accessibilità al sito alle persone non vedenti o ipovedenti;
- **DateEOrari**: rappresenta le date e gli orari di apertura del ristorante;
- **Orario**: rappresenta un generico intervallo tra due orari;
- **Menu**: rappresenta il menù del ristorante; di default il sito prevede un unico menù per ogni ristorante, ma ciò può essere modificato in caso di bisogno (e.g. se qualche evento prevede un menù fisso a tema);
- **Vino**: rappresenta ogni tipo di vino presente nel menù; ne indica tutti i dati di maggiore importanza, come nome, regione di provenienza, annata, prezzo, eccetera;
- **Piatto**: rappresenta ogni piatto presente nel menù; ne indica tutti i dati di maggiore importanza, similmente al vino, in particolare gli ingredienti;
- **Ingrediente**: rappresenta ogni ingrediente presente in un piatto; classe molto semplice che indica unicamente il nome dell'ingrediente e la sua eventuale appartenenza agli allergeni

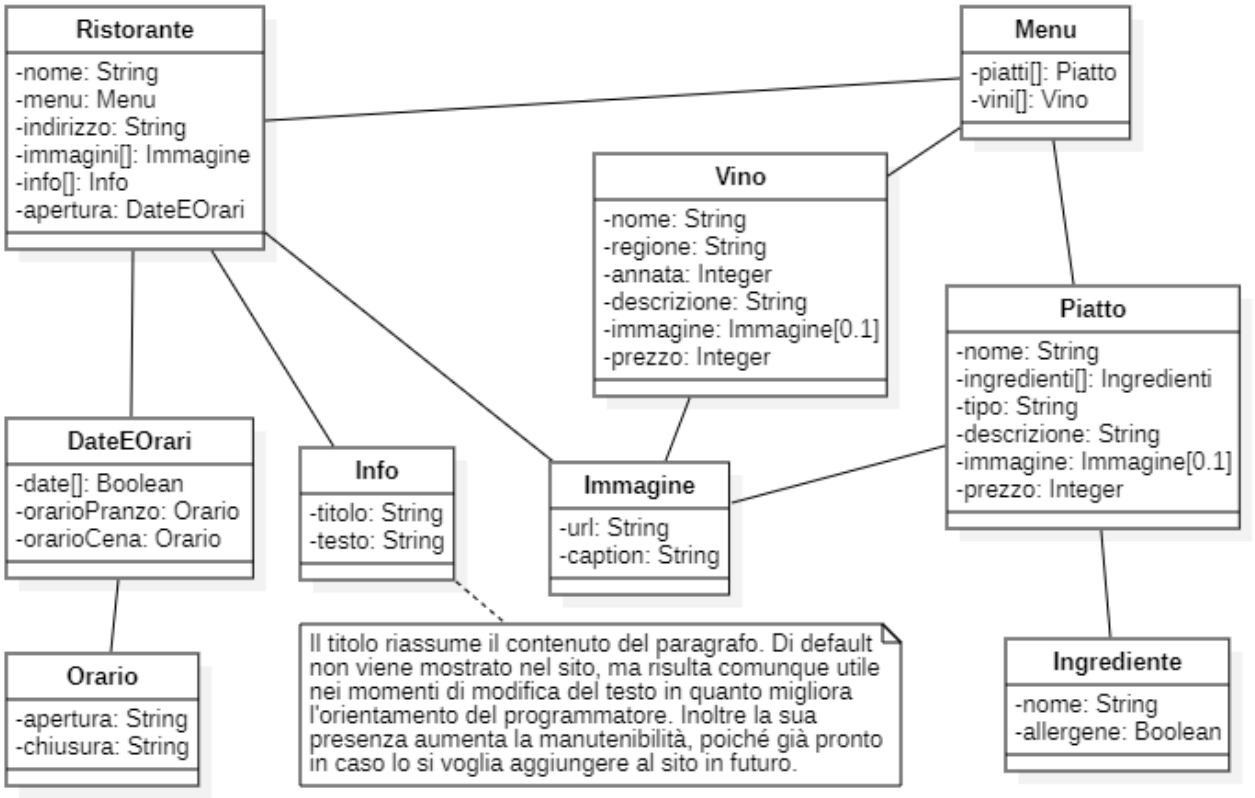

Immagine 6: prima versione della gerarchia

3.5 Le best practices da seguire

Nei paesi anglofoni, il linguaggio comune attribuisce ai termini "best practices" il significato di "a way of doing something that is seen as a very good example of how it should be done and can be copied by other companies or organizations"[9], ovverosia "un modo di fare qualcosa che è considerato un ottimo esempio di come dovrebbe essere fatto e che può essere copiato da altre compagnie o organizzazioni". Un esempio di elevate semplicità ed efficacia è il controllare da entrambi i lati della strada prima di attraversarla.

Nel mondo dell'informatica il significato dei termini non si discosta un granché dalla soprastante definizione, ma assume ovviamente una connotazione in un contesto più specifico. Ivi diventano infatti una sorta di convenzioni che l'informatico segue per una serie di motivi, tra i quali il fornire una migliore comprensione del codice, una migliore performance del prodotto o qualsivoglia altra ragione.

3.5.1 Best practices nel back-end

Partendo con un'analisi delle best practices applicate nel back-end, che compongono una lista ben più esigua rispetto alle corrispettive del front-end, si nota che Java non ne richiede specificatamente alcune. È buona norma comunque seguire le best practices di programmazione "classiche" e, dato che si gestisce qui una gerarchia di classi, quelle relative alla programmazione improntata agli oggetti.

Sono perciò state individuate le seguenti best practices:

- Relative alla programmazione di base:
    - seguire le convenzioni di attribuzione nomi;
    - evitare inizializzazioni non necessarie;
    - utilizzare cicli senza contatore ove possibile;
    - gestire correttamente le eccezioni;
    - preferire float a double ove non è necessaria una precisione elevata;
    - creare un'adeguata e logica struttura di cartelle;
    - commentare adeguatamente il codice.

- Relative alla programmazione ad oggetti:
    - usare interfacce e classi già presenti ove possibile;
    - evitare classi eccessivamente popolose in termini di argomenti o metodi;
    - evitare troppe linee di codice per le definizioni dei metodi;
    - rendere private, o eventualmente protected, gli attributi delle classi ove possibile;
    - assicurarsi che ogni classe abbia una e una sola responsabilità (single responsibility principle)
    - assicurarsi che le sottoclassi siano un tipo specifico delle classi genitore.

3.5.2 Best practices nel front-end

Passando ora all'analisi relativa al front-end si sottolinea che, oltre quelle relative alla programmazione di base già evidenziate nel punto §3.5.1, sono previste le best practices relative ad Angular.

Le principali sono le seguenti:

- usare lo strumento Angular CLI per la creazione di servizi, componenti, eccetera;
- usare elementi caratteristici di ECMAScript 6 (un sottostandard di linguaggio di JavaScript);
- accompagnare alla direttiva **ngFor* il relativo *trackBy*[(II)];
- sfruttare il lazy loading[(III)];
- usare il file *Index.ts* per raggruppare i file tra loro correlati;
- evitare la tipizzazione con *any*;
- dichiarare nuovi tipi per limitare i typo;
- configurare secondo i propri standard il file tslint[(IV)];
- evitare l'annidamento delle subscriptions;
- dividere componenti grandi in componenti più piccoli e riutilizzabili;
- nominare i file [feature].[type].[desinence] (ad esempio home.component.ts);
- assicurarsi che ogni file di typescript abbia una e una sola responsabilità (single responsibility principle, similmente alla programmazione ad oggetti);
- creare ed usare interfacce quando necessario.

(II): trackBy è una direttiva di Angular che permette di identificare inequivocabilmente ogni elemento in una lista di elementi del DOM, gestendoli in maniera simile agli array; incrementa la velocità di caricamento e facilita le interazioni con i singoli elementi.

(III): il lazy loading è una strategia di programmazione web che consiste nel posticipare il più possibile il caricamento di file pesanti, come immagini, video o audio; caricando tali file solo quando richiesto, si distribuiscono meglio i tempi di attesa, e si evita anche di caricare file non richiesti; incrementa la velocità del sito.

(IV): tslint è un file creato automaticamente da Angular CLI al momento dell'esecuzione del comando di creazione di un nuovo progetto; contiene una lista di regole che il programmatore può auto-imporsi al fine di mantenere il codice sorgente pulito, ordinato e attinente alle convenzioni; le regole del tslint sono personalizzabili, ma è buona norma attenersi a quelle convenzionali; il mancato superamento corretto di queste regole causa un arresto nella compilazione del codice.

Capitolo 4

Sviluppo

4.1 Componenti, moduli e servizi per il front-end

Per svolgere al meglio il loro compito, i siti web devono essere, oltre che correttamente funzionanti, accattivanti e semplici nella loro navigazione; l'utente, infatti, deve essere invogliato a rimanere o tornare in un futuro nel sito. Non esiste sfortunatamente, ma per certi versi anche fortunatamente, alcun elemento o stile che garantisca ciò. Tuttavia sono presenti determinate caratteristiche del sito che agevolano questo invogliamento: una struttura semplice ed intuitiva, la capacità di mantenere la concentrazione dell'utente ma senza sovrastimolarlo, la corrispondenza tra risposta che l'utente si aspetta e risposta generata sono alcune di esse.

Per questi motivi, e a fronte di una relativa semplicità del lato applicativo, il progetto prevedeva un'attenta cura del front-end.

Si procede a breve ad elencare tutti i file di codice generati nel corso del progetto. La lista stilata nel paragrafo successivo a questo tratterà dunque dei file presenti nella sezione src (source) del prodotto, andando ad ignorare file e librerie esterne ad essa – generati o importati in maniera automatica da Angular nel processo di creazione di un nuovo progetto – per ragioni di praticità e compattezza di questo documento. Al fine di fornire una maggiore chiarezza della struttura del raggruppamento in cartelle, la lista mostra anche il path dell'elemento analizzato dando per scontato il prefisso "src/". Inoltre, per sinteticità, si considerino le cartelle seguite da un * come folder contenenti i seguenti file:

- *[folderName].[elementType].html*: definisce la struttura e il layout dell'elemento;
- *[folderName].[elementType].scss*: definisce lo stile dell'elemento;
- *[folderName].[elementType].ts*: definisce la logica di comportamento dell'elemento;
- *[folderName].[elementType].spec.ts*: definisce i test unitari applicabili all'elemento.

Tali file verranno dunque dati per scontati e non verranno mostrati nella lista. Inoltre si sottolinea che ogni punto della lista che non presenta una desinenza nel nome indica una cartella.

La lista delle entità create durante il progetto è la seguente:

- **index.html**: html principale; il suo campo head contiene tutti i tag link e meta di uso generale e il tag title della pagina; il campo body contiene unicamente il tag radice di app;
- **main.ts**: comportamento dell'applicazione, necessaria in questo progetto unicamente per effettuare il bootstrap del modulo principale;
- **style.css**: istruzioni di stile generali e applicate di base ovunque;
- **assets**: contiene le personalizzazioni non relative allo stile (quest'ultime infatti già gestite dai file .scss);
- assets/**fonts**: contiene le istruzioni che permettono al linguaggio di usare font non comuni nei browser; i file dentro questa cartella sono di inutile elencazione, in quanto direttamente importati dal web;
- assets/**images**: contiene le immagini che non è necessario salvare in database per leggerezza e bassissimo cambiamento, come il logo del ristorante o eventuali immagini relative alla locazione del posto; anche in questo caso non risulta necessario approfondire nella presente lista i documenti appartenenti a questa cartella;
- **app***: macro-cartella contenente tutti gli elementi non appartenenti agli assets; i documenti relativi alla notazione * sono in questo caso inclusi nella sottocartella features;
- app/**features**: contiene tutti i componenti che rappresentano un'intera pagina del sito (raggrupparli sotto questa cartella è una delle best practices di Angular);
- app/features/**admin***: componente relativo alla pagina di amministrazione;
- app/features/**booking***: componente relativo alla pagina di prenotazione;
- app/features/**home***: componente relativo alla homepage;
- app/features/**info***: componente relativo alla pagina di informazioni;
- app/features/**menu***: componente relativo alla pagina del menù;
- app/features/**app-routing.module.ts**: file per la gestione del routing;
- app/features/**app.module.ts**: file per le dichiarazioni e le importazioni di tutti i moduli, indica inoltre i providers e quali componenti avviare al bootstrap;
- app/**components**: contiene tutti i componenti che non rientrano nelle features (altra best practice di Angular);
- app/components/**carousel***: componente relativo al carosello della homepage;

- app/components/carousel/**carouselImg.ts**: definisce ed esporta l'interfaccia della classe CarouselImg, ovvero di tutte le immagini da mostrare nel carosello;
- app/components/carousel/**myCarouselImg.ts**: definisce ed esporta l'interfaccia della classe MyCarouselImg, che permette un eventuale approccio alternativo alla gestione delle immagini del carosello; la classe non viene implementata insieme a CarouselImg nel file carouselImg.ts perché la separazione della dichiarazione di interfacce esportanti rientra tra le best practices di Angular;
- app/components/**bookingDialog***: dialog di conferma della prenotazione;
- app/components/**loginDialog***: dialog per l'inserimento delle credenziali di accesso al profilo di amministratore;
- app/components/**info-group***: card relativa al riassunto delle informazioni e immagini di contorno ad essa;
- app/components/**menu-group***: card relativa ad un'anticipazione del menù e immagini di contorno ad essa;
- app/components/**navbar***: l'header delle pagine, contiene bottoni per navigare nel sito;
- app/components/**footer***: il footer delle pagine, contiene le informazioni importanti quali orari e contatti;
- app/**services***: contiene i file di tipo modulo atti a dichiarare ed esportare i path per il routing; poiché contiene documenti per un servizio – il quale non necessita di una struttura né, tantomeno, di uno stile – non presenta al suo interno i file .html e .css relativi alla notazione *;
- app/**shared***: contiene i file di tipo modulo atti a raccogliere classi e risorse che sono usate in più di un modulo dinamico; tipicamente sono comodi per gestire i moduli di form; analogamente alla cartella services, non presenta file .html e .css.

Non appartenente al gruppo di documenti interni a src ma comunque degno di nota è il file tsconfig.json. Questo file, generato automaticamente al termine della creazione di un nuovo progetto, contiene una serie di regole di scrittura di codice, le quali impediscono la compilazione nel caso non vengano rispettate. Tali regole sono create di default con determinati parametri e condizioni, ma sono tutte modificabili ed eliminabili. Il grande vantaggio di questa procedura è che il programmatore può imporsi regole personalizzate più o meno rigide, così da affrontare la stesura del codice in maniera più uniforme e, di conseguenza, di maggiore comprensione nel futuro.

4.2 Versione finale della gerarchia di back-end

Nel corso dello sviluppo del prodotto, la gerarchia delle classi gestite dal back-end ha subìto alcune correzioni. Tali modifiche hanno portato ad un guadagno in termini di manutenibilità del prodotto a fronte di un appesantimento della gerarchia che risulta essere di fatto esiguo.

Segue una lista delle riflessioni effettuate con il personale dell'azienda relative ad eventuali modifiche alla struttura della gerarchia.

Nella classe Ristorante si può notare che è presente un solo menù. Ciò non risulta propriamente corretto, in quanto un unico ristorante può offrire molteplici menù (si pensi a particolari eventi o festività che possano essere accompagnate da un menù a tema, oppure ai ristoranti all-you-can-eat che possono scegliere di offrire determinati piatti solo nel menù "alla carta"). Poiché questa caratteristica della classe Ristorante è un errore logico nella struttura di essa, la modifica risulta una correzione necessaria.

Una lunga discussione è stata effettuata riguardo al mantenimento della classe DateEOrari. La sua esistenza infatti offre un vantaggio e uno svantaggio entrambi di lieve impatto: il vantaggio consiste nel rendere la classe Ristorante più semplice e leggibile andando a ridurre il numero dei suoi campi, e risulta lieve perché il numero di campi ridotto è basso; lo svantaggio è che appesantisce e rende di più difficile comprensione la gerarchia delle classi, e risulta lieve poiché aggiunge un'unica classe e un'unica dipendenza. Si è alla fine scelto di mantenerla poiché va a ridurre il grado di responsabilità della classe Ristorante.

Similmente all'analisi di DateEOrari, e con lo stesso esito, è stata svolta un'analisi sulla classe Orario. Si è scelto di mantenere anch'essa perché va a ridurre il grado di responsabilità della classe Ristorante.

Si è notato che la classe Immagine non presenta un titolo. Questo perché al momento della creazione della gerarchia non risultava necessario collegare un titolo alle immagini, in quanto il sito non lo richiede. Tuttavia non vi è garanzia che tale situazione non possa cambiare in futuro. È bene dunque organizzare la struttura affinché essa sia versatile e possa assecondare i cambiamenti richiesti senza bisogno di eccessivi investimenti in termini di lavoro ed energie. Si è concluso quindi che l'aggiunta di un campo facoltativo relativo al titolo sia una scelta corretta.

Al momento i paragrafi contenenti le informazioni non presentano limiti di caratteri ed il loro contenuto è a discrezione del cliente. Ma dato che successivamente egli può voler verificare che i testi inseriti non superino una determinata lunghezza, è meglio fare in modo che ci sia già da subito un attributo facoltativo che imponga il valore entro cui limitarsi. Similmente al ragionamento effettuato con l'aggiunta di titoli facoltativi alle immagini, si è valutata positivamente l'aggiunta di un limite di caratteri facoltativo nella classe Info.

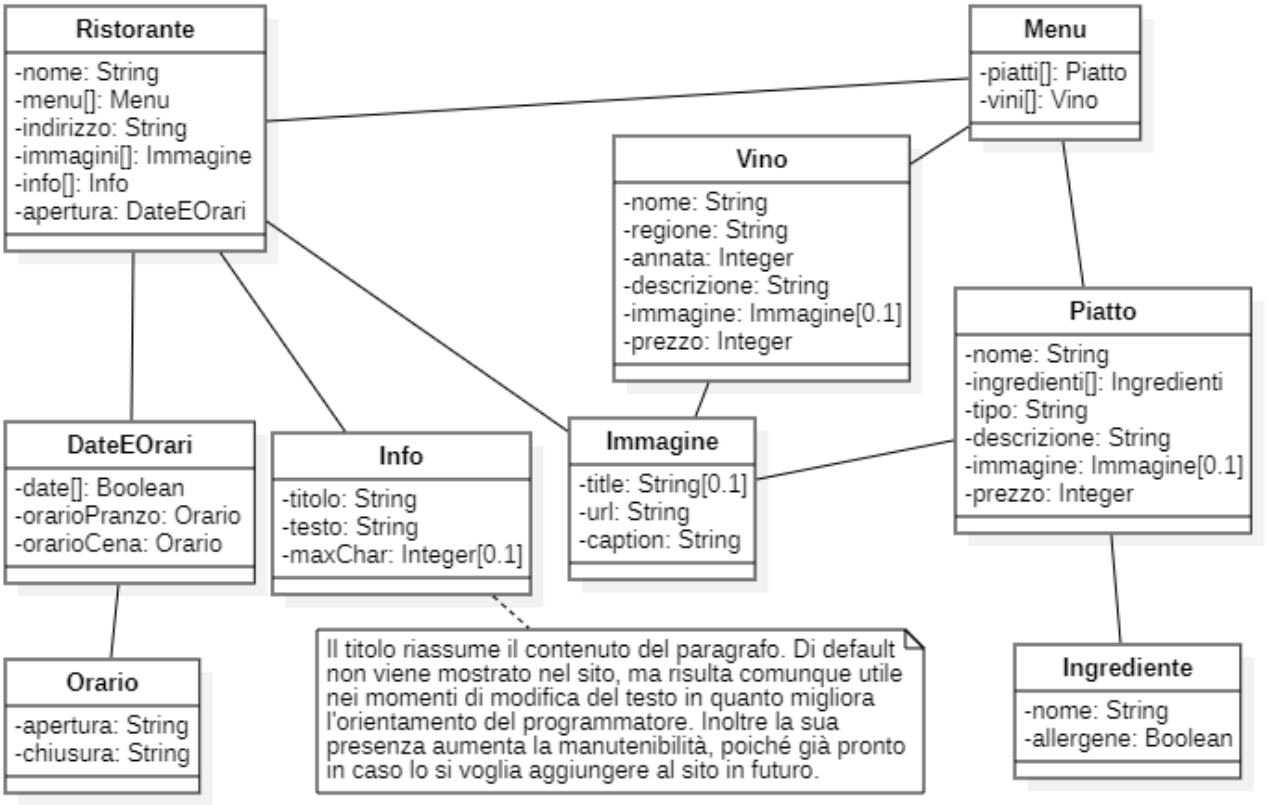

Immagine 7: versione finale della gerarchia

4.3 Analisi sull'accessibilità

Caratteristica fondamentale per i siti è la loro accessibilità. Essa si definisce come il grado di facilità con la quale ogni tipologia di utente riesce a raggiungere, entrare e navigare all'interno del sito in esame. La necessità di rendere un sito accessibile a tutti è dovuta non solo al quasi sempre correlato incremento del ranking nel campo del SEO – di cui si andrà ad analizzare a breve – ma anche al permettere a persone con svantaggi di qualsivoglia tipo di poter usufruire del servizio alla pari degli altri utenti.

Per come è stato strutturata la versione di default del sito, non sono presenti elementi in formato video o audio. Non è dunque necessario uno studio per garantire l'accessibilità a persone con deficit uditivi, in quanto ogni elemento del sito è ben visibile a schermo. Se un ristorante cliente volesse successivamente aggiungere qualche elemento video o audio al sito, ad esempio un classico video per pubblicizzare il posto, allora sarà necessario provvedere all'aggiunta di sottotitoli o mezzi simili per garantire a persone con disabilità di tipo uditivo di comprendere le informazioni comunicate.

Dato il grande quantitativo di informazioni comunicate visivamente invece, sia tramite testo che tramite immagini, il discorso è diverso per le persone con deficit alla vista.
Per facilitare le persone che necessitano di uno screen reader per la navigazione, ogni immagine presente nel sito è accompagnata da un relativo attributo "alt": tale attributo nasce come alternativa (da cui il nome) da mostrare a schermo in caso il caricamento dell'immagine non vada a buon fine, ma è anche ciò che viene letto dallo screen reader all'utente; fornendo all'attributo alt una stringa sufficientemente descrittiva dell'immagine, si può dunque trasmettere l'informazione anche agli utenti che tale immagine non possono vederla. Tutto il resto delle informazioni viene trasmesso dal sito tramite testo, elemento con cui lo screen reader si interfaccia perfettamente; si noti anche che anche i bottoni vengono identificati come tali, o come link, e comunicati all'utente da ogni tipo di screen reader presente[10]. È bene notare che tutti i possibili link sono presenti (anche o solo) nell'header: in questo modo gli utenti che necessitano di screen reader hanno accesso nell'immediato a queste informazioni di maggiore importanza, in quanto questo strumento di assistenza legge gli elementi seguendo l'ordine in cui sono posizionati nel codice sorgente.

Per quanto riguarda le persone che non necessitano invece di dispositivi per la lettura assistita dello schermo ma che presentano comunque alcune difficoltà, si sono presi determinati accorgimenti. Il font usato per il testo è infatti di grandezza maggiore o uguale a 16px in ogni elemento di testo; inoltre è sempre senza grazie per favorire la lettura, eccezion fatta per i titoli dove la grandezza permette una lettura sufficientemente scorrevole nonostante il font qui appartenga alla famiglia degli handwriting. I colori scelti permettono sempre un contrasto elevato tra testo e sfondo e permettono anche di identificare intuitivamente gli elementi di maggiore importanza dal resto.

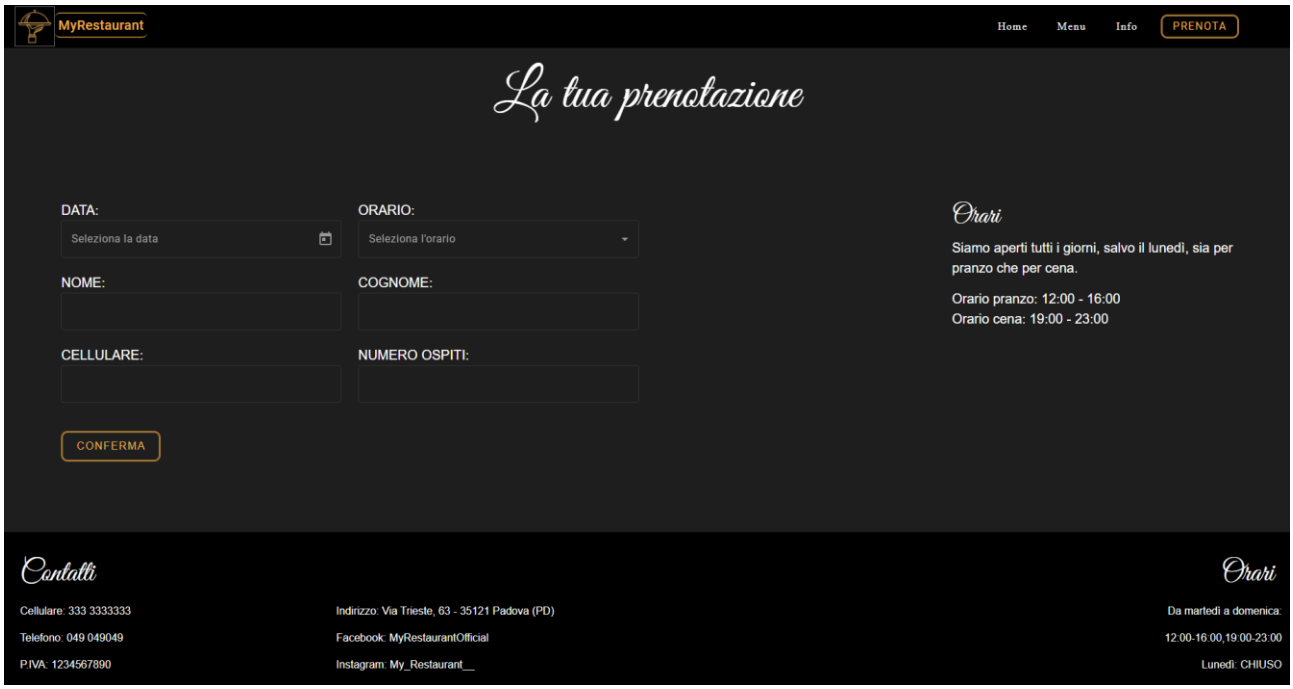

Immagine 8: pagina di prenotazione

L'immagine 8 qui riportata, come anche le altre immagini relative al sito precedentemente aggiunte al documento, permette di dare un'idea della leggibilità dei testi presenti.

4.4 Search Engine Optimization

Secondo la guida introduttiva all'ottimizzazione per i motori di ricerca di Google[11], il SEO – che sta per Search Enging Optimization, per l'appunto ottimizzazione per i motori di ricerca – è una serie di modifiche e tecniche che facilitano i motori di ricerca nello scansionare, indicizzare e comprendere il contenuto di un sito web. Tali tecniche offrono un incredibile vantaggio al sito, poiché una loro adeguata applicazione porta i motori di ricerca a posizionare il sito più avanti degli altri; e ovviamente un posizionamento migliore porta a maggiori visite e, di conseguenza, ad un maggiore profitto (nel caso in esame dovuto ad una maggiore pubblicità del locale).

Tra le tecniche utilizzate durante la creazione del prodotto, si possono individuare le seguenti:

- utilizzare correttamente i tag di intestazione (h1 … h6): un uso moderato di tali tag e una loro limitata grandezza lunghezza fanno assumere loro maggiore importanza;
- indicare la parole chiave il prima possibile;
- ottimizzare il tempo di caricamento del sito (possibile grazie al lazy loading);
- fornire stringhe adeguate negli attributi alt;
- usare link preferibilmente interni;
- migliorare la User Experience: i motori di ricerca analizzano anche quanto gli utenti trascorrono all'interno di un sito e se, terminata la navigazione, tornano indietro e aprono un altro sito esposto tra i risultati di ricerca (procedura nota come "pogo sticking");
- fornire un tag meta ai documenti.

4.5 Responsiveness

La responsiveness è il grado di "adattabilità" del sito ai diversi dispositivi che vi accedono. Un sito altamente responsive è in grado di adattare la propria struttura e i propri contenuti in base alle dimensioni del device che vi fa accesso.

Un uso adeguato di Bootstrap permette una gestione tanto semplice quanto efficace della responsiveness del sito. I seguenti dati su Bootstrap sono presi dalla relativa pagina ufficiale[12].

Le griglie di Bootstrap sono composte da una serie di righe, le quali sono divise a loro volta in colonne. La peculiarità di Bootstrap è però il fornire la possibilità di assegnare alla medesima colonna valori diversi per intervalli di dimensione dello schermo diversi. È possibile infatti individuare quattro breakpoints (small a 576px, medium a 768px, large a 992px ed extralarge a 1200px) e relativi prefissi di classe, così da assegnare alla colonna valori diversi (ma sempre in dodicesimi della riga) in base alla dimensione dello schermo.

Per facilitare la comprensione, si fornisce un esempio: assegnare ad un tag l'attributo class="col-xl-3 col-md-4 col-12" comporta una larghezza responsiva dell'elemento corrispondente pari a:

- un quarto della riga in caso di schermi larghi 1200px o più;
- un terzo della riga in caso di schermi larghi tra 768px e 1199px;
- l'intera riga in caso di schermi larghi 767px o meno.

In questo modo è stato possibile soddisfare il requisito RQ8 relativo alla responsività di ogni pagina del sito.

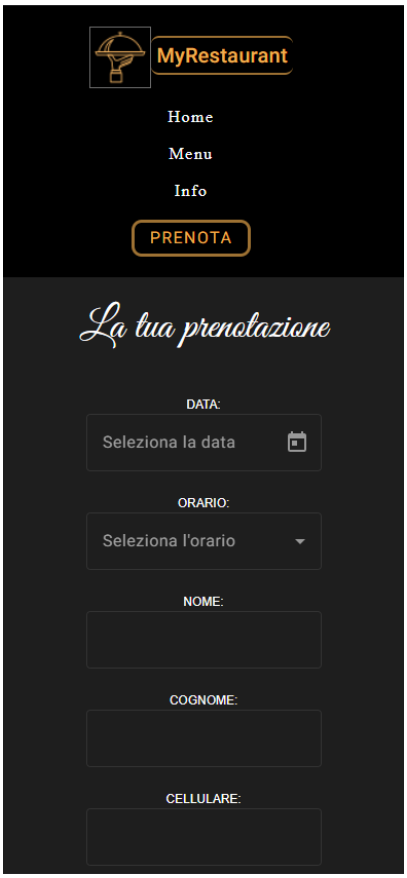

Immagine 9: la pagina di prenotazione vista da un iPhone 12 Pro

Come ben visibile dall'immagine 9 precedentemente riportata, la struttura del sito appare assai diversa se visualizzata da schermi di piccole dimensioni.

4.6 Distribuzione orario definitiva

Assai di rado la distribuzione delle ore di lavoro stimata all'inizio di un progetto coincide perfettamente con la distribuzione effettiva; la previsione ad inizio progetto serve infatti solo a dare un'idea generale di quanto occuperanno le tempistiche di ogni singolo sprint, o di ogni singola attività specifica.

Sebbene sia tollerata una discreta rielaborazione della distribuzione, sarebbe comunque bene per l'andamento del progetto, e per la conseguente qualità del prodotto, adeguarsi alla distribuzione iniziale. Se infatti i cambiamenti risultano eccessivi, lo studio preventivo della distribuzione perde totalmente di utilità e diventa una perdita di tempo.

Una volta redatta la prima distribuzione generica delle ore è bene dunque seguirla il più possibile, ovviamente con i dovuti margini di cambiamento collegati alla sempre presente imprevedibilità dello sviluppo. Ciò non è avvenuto nel caso del tirocinio di cui questo documento è relazione; infatti, sebbene la struttura della distribuzione sia simile a quella prevista, la quantità di ore dedicate alle singole attività e le stesse attività si allontanano eccessivamente dal preventivo iniziale.

Seguono una lista dei più importanti cambiamenti avvenuti e una tabella con la distribuzione finale dell'orario.

- A fronte di un quantitativo di ore per la formazione stimato a 120 ore, essa è durata circa 70 ore; ciò grazie alla velocizzazione dei filmati di spiegazione e ad una sottostimata preparazione dello studente fornita dal corso di studi dell'Ateneo.
- La stesura della documentazione è stata totalmente abbandonata, eccezion fatta per l'analisi dei requisiti e la stesura degli use case; tuttavia tali studi non sono stati effettuati con lo stesso approfondimento mostrato in questo documento, ma in maniera più superficiale. A fronte di una stima di 35 ore dedicate in totale alla documentazione, durante il tirocinio se ne sono effettivamente utilizzate 4.
- I test unitari previsti non sono stati implementati. Questo perché alla creazione di un nuovo componente, Angular CLI genera in automatico un file di test unitario. È risultato dunque sufficiente effettuare tali test tramite il comando *ng test*. Per quanto riguarda i test di integrazione, invece, se n'è occupata l'azienda, contrariamente a quanto previsto. È difficile fornire una stima di quanto l'attività di testing abbia impattato sulla distribuzione delle ore di

lavoro, in quanto ogni elemento veniva testato per conto proprio al termine del suo sviluppo, ma di certo questa quantità non supera l'ora.

- Nonostante fosse previsto diversamente, l'azienda non ha fornito il back-end dell'applicazione, ma ha accompagnato lo studente nella creazione di esso. Ciò ovviamente ha sottratto il tempo ad altre attività, in quanto non era presente nel preventivo. Tra studio della gerarchia delle classi ed implementazione effettiva della logica, il back-end ha richiesto un ammontare di 80 ore precise.
- La stesura dello swagger dell'applicazione non è stata effettuata; l'azienda ha infatti preferito che il laureando si dedicasse maggiormente al sito.
- Lo sviluppo del front-end, sia per quanto riguarda l'approccio CRUD sia quello alle API, ha ricevuto molte più attenzioni di quanto preventivato.

La seguente tabella delle ore viene raggruppata per macro argomenti, disposti in ordine cronologico. Le durate sono indicate in ore.

Descrizione attività	**Durata prevista**	**Durata effettiva**
Introduzione all'azienda e definizione degli obiettivi	15	4
Stesura della documentazione	35	4
Formazione generale (strumenti di sviluppo, single page application, two-way data bindings, styling, tipi di base di dati)	120	72
Sviluppo front-end CRUD	50	80
Sviluppo back-end	0	80
Sviluppo front-end API	50	64
Testing	20	0
Stesura swagger	10	0
TOTALE	300	304

Capitolo 5

Conclusioni

5.1 Raggiungimento obiettivi e soddisfacimento requisiti

Nella sezione §1.2 di questo documento si sono analizzati gli obiettivi prefissati per il tirocinio. Se ne indica ora invece il loro raggiungimento grazie alla seguente tabella.

Descrizione	Importanza	Stato
Studio e pratica delle tecnologie e degli strumenti da utilizzare	Obbligatorio	Raggiunto
Stesura dell'analisi dei requisiti	Obbligatorio	Raggiunto
Progettazione dell'applicazione	Obbligatorio	Raggiunto
Sviluppo dell'applicazione	Obbligatorio	Raggiunto
Implementazione di una sezione admin	Desiderabile	Raggiunto
Implementazione di una gestione real-time, tramite notifiche push	Facoltativo	Non raggiunto
Affiancamento al team operativo aziendale nell'analisi e sviluppo di prodotti più complessi	Facoltativo	Raggiunto

Per quanto riguarda i requisiti invece, quest'ultimi elencati nella sezione §2.3, si può sintetizzare per praticità che sono stati tutti soddisfatti.

5.2 Possibilità sull'utilizzo del prodotto

Allo stato finale, il prodotto è un sito web che dispone anche di un lato applicativo. Sebbene non rappresenti un ristorante preciso (tutti i dati mostrati sono o di pura fantasia o relativi al Dipartimento di Matematica "Tullio Levi-Civita" dell'ateneo patavino) può di fatto trasformarsi in un suo sito assai velocemente: è sufficiente inserire infatti i dati del ristorante scelto – locazione, contatti, piatti disponibili, orari, immagini, eccetera – per ottenere un sito già pronto che permette, oltre che di essere un elemento di pubblicità digitale, di effettuare prenotazioni di posti.

Inutile dire che un sito ben fatto presenta generalmente molte più funzionalità di quelle supportate dal prodotto, ma possiamo considerare quest'ultimo come una sorta di scheletro su cui successivamente aggiungere eventuali altre features richieste dal ristoratore. Il prodotto è infatti un sito base, i cui elementi difficilmente non vengono richiesti; ulteriori elementi aggiunti non necessariamente porterebbero ad un valore aggiunto del sito ad essi collegato, poiché potrebbero essere non richiesti dal ristorante e risultare dunque unicamente un disturbo per il ristoratore e per i clienti che ci potrebbero navigare.

Poiché la progettazione e lo sviluppo sono avvenuti seguendo le best practices, eventuali aggiunte di elementi al codice saranno semplici.

5.3 Valutazioni complessive

Queste 300 ore di tirocinio si sono rivelate, contro ogni aspettativa, di elevati interesse. Il mondo del lavoro era una novità per il sottoscritto, ma ciononostante le giornate sono trascorse con rapidità; ciò comprova un forte coinvolgimento negli argomenti trattati e nell'approccio stesso alla progettazione e alla creazione del sito.

Il tirocinio si è rivelato non solo una prima esperienza nel settore lavorativo, ma anche una valutazione delle abilità e conoscenze acquisite durante il triennio di studi.

Gli strumenti utilizzati sono stati altamente intuitivi: assai raramente è capitato di dover chiedere a colleghi aiuto per la comprensione di essi. Anche i linguaggi appresi, in particolare Angular, si sono rivelati di semplice comprensione, nonostante abbiano le fondamenta su una logica complicata.

L'autonomia lasciata da parte dell'azienda è stata assai elevata. Un'eccessiva autonomia degli sviluppatori può risultare un'arma a doppio taglio, ma fortunatamente questo non è stato il caso.

L'ambiente di lavoro è stato molto accogliente e i colleghi si sono mostrati altamente comprensivi, competenti e cordiali. A inizio di ogni giornata avveniva una riunione di circa mezz'ora per aggiornarsi sugli sviluppi raggiunti e sui problemi riscontrati, e ognuno aveva l'occasione di aiutare qualcun altro. Personalmente ho avuto un impatto molto positivo riguardo a tale riunione, un giudizio che non mi aspettavo minimamente si sbilanciasse verso questa direzione.

In sintesi la valutazione personale riguardo quest'esperienza è largamente positiva, con l'ovvia presenza tuttavia di alcune piccole critiche che verranno analizzate nel capitolo successivo.

5.4 Critiche

Alcuni fattori riscontrati durante queste (circa) otto settimane di lavoro non hanno avuto un riscontro positivo sulla mia personale valutazione. Come è stato giusto elencare gli aspetti positivi di questa mia esperienza, reputo corretto elencare anche quelli che altrettanto positivi non sono.

In primis si può notare la mancata adesione al piano di progetto. L'analisi dei requisiti redatta successivamente ad esso non ha infatti seguito a sufficienza, perlomeno a mio parere, tale documento, mancando di assegnargli la giusta importanza che merita.

Altro aspetto negativo del prodotto è anche la sua scarsissima documentazione. Non sono stati prodotti alcun manuale utente né alcuna documentazione esterna al riguardo: le uniche tracce di documentazione si possono trovare unicamente nel codice sorgente. Ciò abbassa notevolmente il grado di manutenibilità che l'adesione alla best practices e la stesura ordinata del codice hanno apportato.

Fortunatamente si nota che gli aspetti positivi superano di gran lunga quelli negativi rilevati, il che è un ottimo indice di qualità dell'esperienza. Tuttavia è bene non adagiarsi su un simile risultato positivo, in quanto è sempre possibile migliorare. Sarebbe dunque corretto in eventuali progetti futuri seguire questi e i prossimi spunti, in modo da mantenere un graduale e costante miglioramento nel corso del tempo.

Riferimenti esterni

(1): GitHub di Angular. Link:
https://github.com/angular/angular

(2): WayBack Machine, archivio internet. Link:
https://web.archive.org/web/20180118181923/http://angularjs.blogspot.co.uk/2016/09/angular2-final.html

(3): Angular Minds, azienda specializzata in Angular. Link:
https://www.angularminds.com/blog/article/comparison-difference-between-angular1-vs-angular2-vs-angular4.html

(4): GitHub di Flex Layout. Link:
https://github.com/angular/flex-layout

(5): Guida di Angular. Link:
https://angular.io/guide/architecture

(6): Link:
https://markdotto.com/2012/01/17/bootstrap-in-a-list-apart-342/

(7): Documentazione ufficiale Bootstrap. Link:
https://getbootstrap.com/docs/5.3/getting-started/introduction/

(8): Sezione riguardo a Java di Oracle. Link:
https://www.oracle.com/java/technologies/#334

(9): Definizione di Oxford. Link:
https://www.oxfordlearnersdictionaries.com/definition/american_english/best-practice#:~:text=a%20way%20of%20doing%20something,best%20practices%20among%20smaller%20companies

(10): Università di Washington. Link:
https://www.washington.edu/accessibility/websites/links-buttons/

(11): Link:
https://developers.google.com/search/docs/fundamentals/seo-starter-guide

(12): Link:
https://getbootstrap.com/docs/4.0/layout/grid/