



Bagni X Booking System
Applicazioni e Servizi Web

Riccardo Maldini
riccardo.maldini2@studio.unibo.it

Thomas Angelini
thomas.angelini@studio.unibo.it

Francesco Gorini
francesco.gorini@studio.unibo.it

Ottobre 2020

Indice

1	Introduzione	3
2	Requisiti	4
2.1	Requisiti utente customer	4
2.2	Requisiti utente admin	5
3	Design	7
3.1	Schermate libere	8
3.2	Area riservata del customer	10
3.3	Accesso all'area riservata admin	11
3.4	Area riservata admin	12
3.5	Sezioni in modifica admin	14
4	Tecnologie	15
4.1	Stack MEAN	15
4.1.1	MongoDB	15
4.1.2	Node.js	15
4.1.3	ExpressJs	16
4.1.4	Angular	16
4.2	Metodologie	16
4.3	Tecnologie per la sicurezza	17
4.3.1	Middleware di autenticazione custom	17
4.3.2	JWT	18
4.3.3	Modulo Sanitizer	19
4.3.4	BcryptJs	19
4.3.5	Guard Angular	20
4.4	Tecnologie di supporto	20
4.4.1	Docker	20
4.4.2	GitHub, Actions e Pages	20
4.4.3	Postman	21
4.4.4	Swagger	21
4.4.5	Adobe XD	21

5 Codice	22
5.1 Struttura della repository	22
5.2 Server	22
5.3 Client	23
6 Test	25
6.1 Continuous Integration	25
6.2 Testing del server	25
6.3 Testing del client	26
6.4 Test di integrazione	26
6.5 Interazione con utenti esterni	26
7 Deployment	27
7.1 Installazione e messa in funzione	27
7.2 Demo su GitHub Pages	28
8 Conclusioni	29

Capitolo 1

Introduzione

Nell'ultimo periodo, le preoccupazioni legate al sorgere dell'emergenza Covid-19 hanno portato, quale effetto collaterale, ad un'importante spinta alla digitalizzazione di determinati settori. Il settore balneare si è rivelato uno di quelli influenzati in maniera particolare da questa tendenza, a causa soprattutto dei timori legati ai contagi in spiaggia. Questi hanno portato allo sviluppo di diverse app e soluzioni per la prenotazione dei posti in spiaggia. Il contesto emergenziale ha portato a un proliferare di soluzioni "artigianali": molte di queste, a causa del contesto emergenziale nel quale sono sorte, sono ben lontane dalla soluzione ottima, e presentano una scarsa cura nei dettagli e negli aspetti di usabilità e UX.

Il progetto in questione si propone di portare una soluzione, potenzialmente utilizzabile fin da subito, per la disposizione di un sistema di prenotazione online solido, moderno, costruito attorno alle esigenze del settore degli stabilimenti balneari.

Più nello specifico, il progetto si propone di realizzare una web app per un generico stabilimento balneare "Bagni X". Lo scopo principale è la gestione delle prenotazioni:

- Sia **lato cliente**, il quale assume la possibilità di personalizzare interamente le proprie prenotazioni, e visualizzare una panoramica completa dei servizi offerti dallo stabilimento;
- Sia **lato proprietario/bagnino**, i quali possono gestire le prenotazioni e personalizzare in toto la schermata visualizzata dai clienti, tramite un CMS sviluppato su misura.

Il progetto sfrutta le tecnologie allo stato dell'arte per lo sviluppo di web app, ed è pensato come un modello base applicabile a un qualunque stabilimento balneare o attività similare nello stesso settore (come piscine, o camping).

L'emergenza Covid-19 ha rappresentato di fatto il trampolino di lancio per la digitalizzazione di un settore storicamente sotto digitalizzato. La web app mira dunque a inserirsi in un settore in piena espansione negli anni a venire.

Capitolo 2

Requisiti

Il sistema deve presentarsi all’utente sotto forma di una web app, accessibile all’esterno come una Single Page Application. Questa si configura quindi come un rich client, in grado di effettuare richieste AJAX in direzione di vari endpoint, tali da comporre una API. I dettagli dell’implementazione vengono approfonditi nei capitoli successivi.

Il client consente la visualizzazione di informazioni da parte di due tipologie di utenti:

- L’utente **customer** detiene il ruolo di cliente dello stabilimento. Questo presenta quindi necessità legate all’effettuare di prenotazioni, consultare informazioni e aggiornamenti sullo stabilimento;
- L’utente **admin** detiene il ruolo di proprietario, o lavoratore all’interno dello stabilimento balneare. Presenta la necessità di gestire le prenotazioni, i servizi offerti, e la personalizzazione della pagina visualizzata da utenti visitatori.

In linea di massima, ogni tipologia di utente porta con sé un **set di permessi gerarchico** sulle operazioni che possono essere svolte, ovvero tale per cui ogni raggruppamento può svolgere le operazioni di quella sottostante: le operazioni del customer possono anche essere effettuate dall’utente admin, e così via.

Si ricorda inoltre che lo scopo primario del progetto è quello di realizzare un sistema di prenotazioni, ma **non di pagamento**. Ciò esula dagli scopi del progetto, e presenta inoltre delle problematiche pratiche di difficile gestione, seppur ciò rappresenti un’importante caratteristica da aggiungere eventualmente in futuro.

2.1 Requisiti utente customer

L’utente customer rappresenta un potenziale cliente dello stabilimento. In assenza di registrazione, un cliente visitatore ha la possibilità di:

- **Consultare la home page:** la home page racchiude tutte le informazioni sullo stabilimento, oltre ai listini prezzi completi per i servizi offerti;
- **Consultare le news dello stabilimento:** la sezione news permette di fornire informazioni giornaliere, alla stregua di una sorta di blog;
- **Registrarsi all'applicazione, o effettuare il login:** delle schermate specifiche permettono di effettuare l'accesso all'area personale.

La registrazione al sito permette al customer di accedere ai servizi di prenotazione. L'accesso avviene tramite una combinazione di email e password. In particolare, i servizi offerti sono i seguenti:

- **Effettuare nuove prenotazioni:** un'apposita schermata permette di creare la propria prenotazione, personalizzandone aspetti quali l'ombrellone, i servizi aggiuntivi, e il periodo, sulla base della disponibilità dello stabilimento;
- **Gestire le prenotazioni pregresse:** le prenotazioni effettuate possono essere visualizzate in un'apposita schermata, atta anche a consultarne lo stato. Egli può anche annullare la prenotazione, a patto che l'annullamento sia effettuato almeno due giorni prima dell'erogazione del servizio;
- **Gestire il proprio profilo:** una schermata profilo consente la modifica dei dati personali, della password, l'effettuazione di logout e rimozione dell'utente.

2.2 Requisiti utente admin

L'utente admin ha un ruolo di gestione e supervisione sul sistema. Dall'interfaccia web può accedere alla propria area riservata, che si configura in maniera similare ai classici CMS per siti web e blog, come Wordpress. L'accesso all'area riservata viene disciplinato da un sistema di autenticazione basato sulla combinazione di username e password, gestito internamente del server.

La tipologia di utente admin può essere inoltre suddivisa in due sottotipologie, o livelli di permessi:

- Il livello **standard** permette di effettuare le operazioni comuni di gestione e personalizzazione;
- Il livello **root** permette, in aggiunta alle operazioni standard, di aggiungere o rimuovere gli altri utenti, tramite l'apposita interfaccia web. Idealmente, nel sistema è presente un solo utente root.

Un generico admin può quindi effettuare dall'interfaccia web le seguenti operazioni:

- **Personalizzare l'appearance lo stabilimento:** le possibilità di personalizzazione riguardano lo stabilire l' inizio e la fine della stagione, il logo dello stabilimento, il contenuto e la disposizione delle schede della home page visualizzata dai clienti;
- **Personalizzare i servizi in vendita:** lo stabilimento detiene di fatto un catalogo di servizi, composto da varie categorie di ombrelloni, servizi aggiuntivi, e degli sconti applicabili su determinate categorie di ombrelloni, in determinati periodi. È tra le possibilità dell'utente admin la personalizzazione di questi aspetti;
- **Gestire le prenotazioni dei clienti:** le prenotazioni effettuate dai clienti possono essere consultate dall'admin, confermate o eventualmente rifiutate. È possibile inoltre aggiungere delle prenotazioni direttamente da parte dell'amministratore, nel caso in cui i clienti non siano registrati all'applicazione;
- **Personalizzare la sezione news:** possibilità da parte dell'admin di modificare il blog dello stabilimento;
- **Visualizzare informazioni sui clienti:** al momento della registrazione, il cliente può decidere di fornire dei recapiti, allo scopo di poter essere più facilmente contattati. L'amministratore deve poter accedervi facilmente, oltre alle prenotazioni pregresse dello specifico cliente;
- **Visualizzare statistiche a supporto delle decisioni:** una specifica sezione permette, tramite grafici e rappresentazioni apposite, di visualizzare informazioni statistiche sull'andamento della stagione:
 - di tipo **statistico**, andando ad aggregare i dati raccolti nella stagione corrente o semplicemente le informazioni del giorno odierno;
 - di tipo **predittivo**, andando a stimare l'andamento futuro della stagione, in base ai dati pregressi, ai vari periodi della stagione, e ad altre informazioni di rilievo.

Capitolo 3

Design

Nell'ambito di progetto, si è posta particolare cura allo sviluppo dell'interfaccia utente. Aspetti come l'usabilità, la responsività, la coerenza grafica tra gli elementi dell'interfaccia, sono stati posti come prioritari.

Un livello di attenzione tale ha richiesto la messa in atto di un processo di design strutturato, atto a pianificare dettagliatamente il layout delle interfacce ben prima dell'implementazione concreta. Tale processo si è diramato nelle seguenti fasi:

1. **Brainstorming:** come primissima fase, sono state fatte delle ricerche sul dominio applicativo, ricercando delle applicazioni similari, ed andando ad elaborare degli schizzi di interfaccia, e della gerarchia delle schermate;
2. **Implementazione del mockup:** si è quindi andato ad implementare mockup, con le principali schermate dell'applicazione. La l'applicativo utilizzato per l'elaborazione di questo è Adobe XD.

Essendo il client sviluppato su tecnologia PWA, si è cercato sin da questa fase di emulare quanto possibile possibile il design nativo delle applicazioni per smartphone. Si sono seguite quanto più possibile le linee guida del Material Design, utilizzando anche librerie XD apposite.

Questa fase è stata utile in particolar modo per definire le linee guida di design dell'applicazione, la palette dei colori, e i principali componenti grafici, poi da implementare concretamente nel client.

Va inoltre sottolineata l'estrema versatilità dell'applicativo Adobe XD, il quale ha permesso non solo di sviluppare le schermate, ma anche di porre in atto un vero e proprio prototipo navigabile, utilizzato anche per dimostrazioni ad utenti esterni.

3. **Confronto con soggetti esterni:** una parte fondamentale del processo di sviluppo ha riguardato il confronto con dei soggetti esterni, sia da parte di soggetti interessati al progetto, che da potenziali clienti, allo scopo di raccogliere informazioni sull'usabilità del sistema “sul campo”, e su

eventuali caratteristiche da rivedere. Ciò ha permesso di correggere il tiro, sia su alcune caratteristiche dell’interfaccia, che delle intere funzionalità nate da proposte esterne (ad esempio, la possibilità di avere una rubrica dei clienti, aggiunta in seguito a un suggerimento esterno);

4. **Implementazione del client Angular:** terminato il mockup, si è proceduto con lo sviluppo concreto dell’interfaccia sul framework Angular. In questa fase, l’ausilio del mockup è stato essenziale, e ha permesso di porre in atto lo sviluppo con una fondazione di idee ben chiara.

A seguire vengono descritte le varie schermate dell’applicazione, risultato del lavoro di design.

3.1 Schermate libere

Nel momento in cui un utente non autenticato si ritrova ad esplorare il sito, questo si ritrova a poter esplorare una sezione limitata dello stesso, atto perlopiù a fornire al visitatore una panoramica generale sui servizi offerti. Tali schermate vengono descritte nelle figure 3.1 e 3.2:

- **Home page** permette di avere una panoramica generale sullo stabilimento e sui servizi offerti;
- **News** funge da blog dello stabilimento, consultabile anche dagli utenti non registrati;
- **Login** permette ai customer registrati di accedere alla propria area personale;
- **Register** permette ai customer non registrati di disporre della propria area personale, e del servizio di prenotazione.

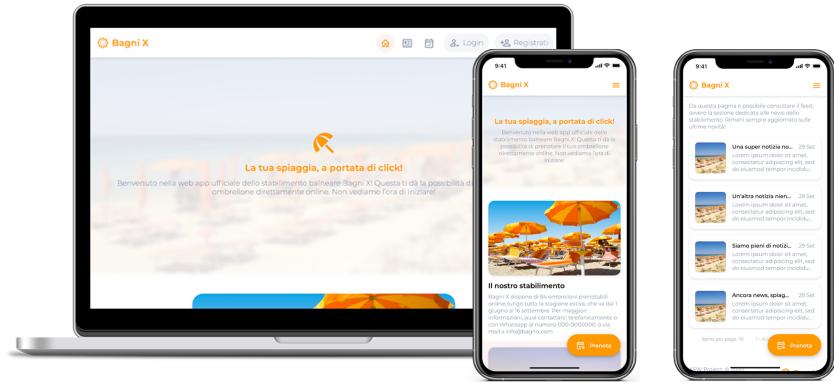


Figura 3.1: Schermate libere accedibili dal customer.

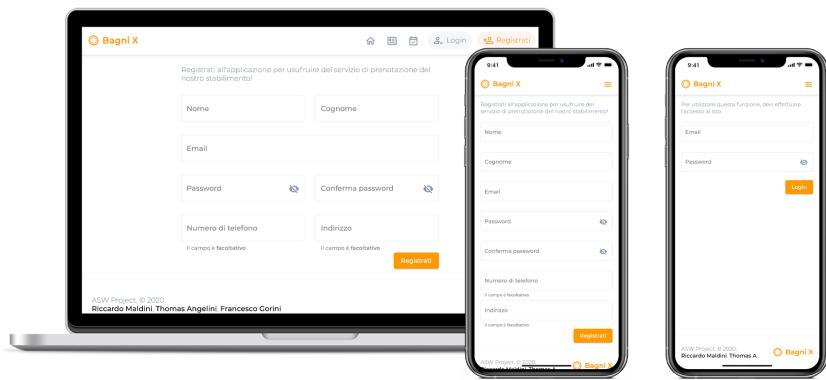


Figura 3.2: Schermate di registrazione e login customer.

3.2 Area riservata del customer

A seguito dell'accesso alla propria area personale, il customer ha la possibilità di accedere alle funzionalità di prenotazione e di gestione dell'account, mostrate nelle schermate a figura 3.3 e 3.4:

- All'interno di tutta l'area personale, tramite la pressione del FAB in basso a destra, è possibile accedere al **flusso di prenotazione**, descritto nella figura 3.4. Questo consiste in tre step successivi tra di loro, concretizzati in tre diverse schermate, che consentono di selezionare il periodo lungo il quale prenotare, personalizzare ombrelloni e servizi, e completare la prenotazione;
- **Bookings** consente di visualizzare e gestire tutte le prenotazioni effettuate fino a quel momento. Premendo su una di queste, è possibile visualizzare informazioni aggiuntive sulle stesse, ed eventualmente accedere ai comandi per l'annullamento della prenotazione, se possibile;
- **Profile** è la schermata dalla quale visualizzare e personalizzare le impostazioni del proprio account, quali i recapiti o la password. Presenta inoltre delle opzioni per eliminare l'account ed effettuare il logout;

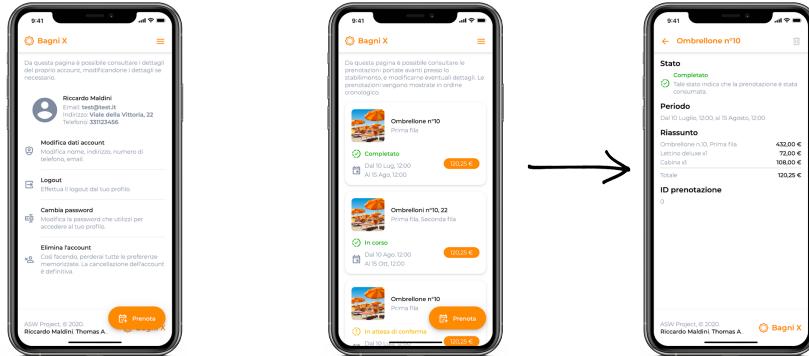


Figura 3.3: Schermate principali dell'area riservata del customer, e dettagli prenotazione.

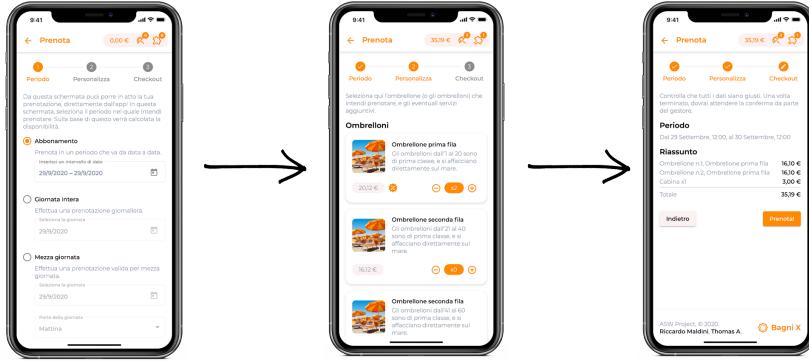


Figura 3.4: Step rappresentativi del flusso di prenotazione.

3.3 Accesso all'area riservata admin

L'accesso all'area riservata admin richiede uno step aggiuntivo di lavoro da parte dell'utente, per evitare che l'utente customer incappi accidentalmente nella schermata di login admin. Tale schermata, raffigurata nella figura 3.5, può essere visualizzata digitando nella barra degli indirizzi `http://<domainName>/admin`, o `http://<domainName>/admin/login`, in maniera similare a quanto accade ad esempio nel CMS di Wordpress.

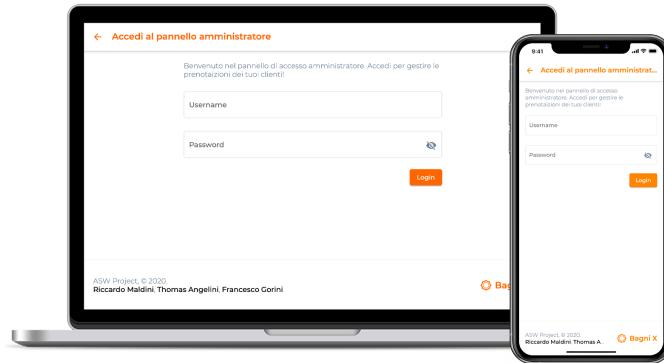


Figura 3.5: Schermate di login admin.

3.4 Area riservata admin

A seguito del login, l'utente admin ha la possibilità di navigare in una serie di schermate che permettono ad esso di gestire “da dietro le quinte” diversi aspetti della web app. Le schermate navigabili, mostrate nelle figure 3.6 e 3.7, descritte in seguito:

- La schermata di **gestione delle prenotazioni** permette di visualizzare tutte le prenotazioni effettuate dai clienti, in ordine cronologico decrescente, e di confermare o annullare queste ultime. A differenza di ciò che avviene nella corrispettiva schermata customer, l'admin ha la possibilità di eliminare la prenotazione in qualunque momento. Inoltre, tramite il FAB in basso a destra, viene data la possibilità all'admin di aggiungere manualmente delle prenotazioni, nel caso in cui si vogliano aggiungere prenotazioni di utenti non registrati all'applicazione. Il flusso di prenotazione è molto simile a quello visualizzato dal customer, con l'aggiunta di uno step preventivo per la selezione del customer;
- La schermata **statistiche** permette di visualizzare diverse informazioni statistiche e previsionali di grande utilità per i gestori dello stabilimento. Nel particolare:
 - Una prima sezione permette di visualizzare statistiche aggregate giornaliere;
 - Una seconda, delle statistiche storiche e previsionali relative alla stagione corrente. I dati previsionali sono calcolati sulla base di dati statistici immagazzinati lato server;
 - Una terza, delle informazioni storiche legate ad eventuali stagioni pregresse.
- La schermata **personalizza stabilimento** permette di modificare i dati mostrati nella welcome screen visualizzata dai visitatori del sito. Permette inoltre di modificare la finestra di prenotazione disponibile (ovvero le date di inizio e fine stagione), e il nome e logo dello stabilimento (mostrati in varie sezioni dell'applicazione);
- le schermate **categorie** e **servizi** permettono di modificare l'offerta dello stabilimento, espressa in categorie di ombrelloni e servizi accessori, aggiungendone o modificandone i dati, aggiungendo eventualmente sconti su determinati periodi;
- la schermata **rubrica** permette all'admin di visualizzare i recapiti e le prenotazioni legati a specifici clienti, e dà inoltre la possibilità all'admin di aggiungerne manualmente degli altri. In tal caso, tali customer non disporranno di un proprio account, ma potranno essere utilizzati dall'admin per aggiungere prenotazioni a loro riconducibili;

- infine, la schermata **profilo** permette all'utente admin di modificare il proprio username o password, o di effettuare il logout. L'utente admin con privilegi di root, visualizza inoltre in questa schermata delle opzioni aggiuntive, che permettono lui di aggiungere e rimuovere utenti admin.

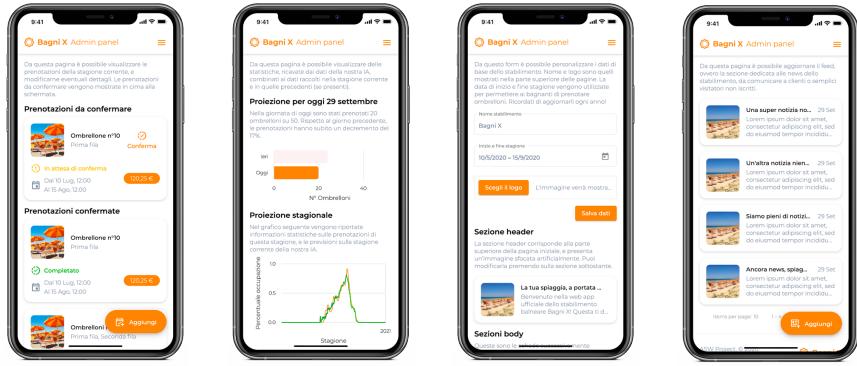


Figura 3.6: Schermate accessibili a seguito dell'avvenuta autenticazione come customer, prima parte.

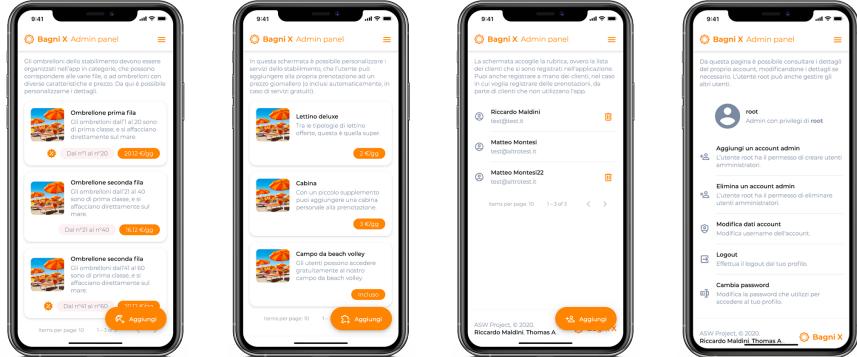


Figura 3.7: Schermate accessibili a seguito dell'avvenuta autenticazione come customer, seconda parte.

3.5 Sezioni in modifica admin

Le schermate di modifica sono quelle accessibili tramite la pressione in una delle voci delle schermate precedenti, come quella delle prenotazioni, quelle legate a categorie ombrelloni o servizi, o ai FAB per l'aggiunta di questi ultimi. Queste permettono dunque di modificarne i rispettivi dati, o generarne, tramite degli appositi form. Alcuni componenti di questi form sono stati ereditati dalla libreria Angular Material, mentre degli altri, come quelli per l'upload di immagini e saldi, sono stati creati da zero, coerentemente con lo stile grafico adottato.

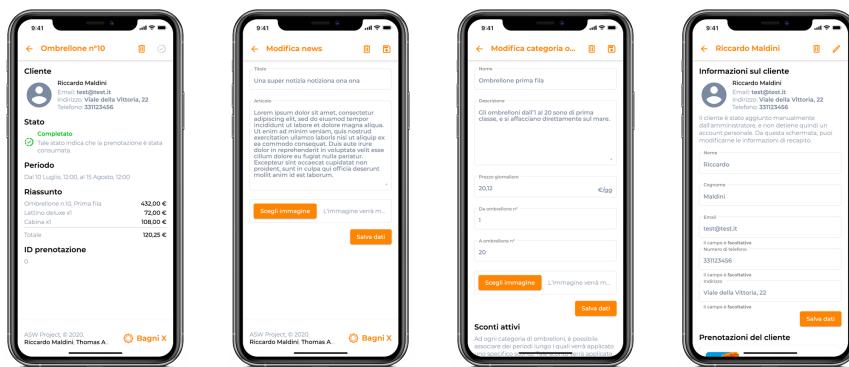


Figura 3.8: Principali schermate di modifica diretta dell'admin.

Capitolo 4

Tecnologie

In questo capitolo vengono approfondite le principali tecnologie adottate nell'e-laborato, discutendo le motivazioni che hanno portato alla loro adozione.

4.1 Stack MEAN

Come scelta progettuale, si è deciso di adottare lo **stack MEAN puro**, tramite l'adozione dei framework MongoDB per la memorizzazione delle informazioni, NodeJS ed Express utilizzati per lo sviluppo del backend, e Angular per il frontend. La scelta è stata dettata principalmente per la volontà di approfondire tali framework da parte dei membri del gruppo.

4.1.1 MongoDB

MongoDB è un software open source utilizzato per la memorizzazione dei dati, in particolare facente parte dei cosiddetti DBMS NoSQL. Esso memorizza i dati come documenti in stile JSON anziché tramite tabelle relazionali. Questa sua particolare caratteristica lo rende molto efficiente, e facile da modellare su un sistema RESTful, che interagisce tramite file JSON.

Gli stack moderni per la creazione di applicazioni web come MEAN, MEVN, o MERN si basano tutte su MongoDB in quanto garantisce un'efficienza ed una flessibilità maggiore rispetto a gli schemi relazionali.

4.1.2 Node.js

Node.js rappresenta il motore runtime JavaScript del server di progetto. I motivi della sua grande popolarità sono molteplici:

- **multipiattaforma**: questa caratteristica permette a Node di essere mandato in esecuzione su diverse infrastrutture;
- **open source**: la community che ne deriva è veramente ampia, e ciò permette di reperire molto facilmente documentazione e supporto;

- **estendibilità:** permette di integrare in maniera automatizzata ulteriori moduli, tramite il package manager NPM.

4.1.3 ExpressJs

ExpressJs è un framework open source pensato per lo sviluppo di applicazioni web, disponibile per Node tramite il package manager NPM. È ormai divenuto lo standard de facto per lo sviluppo di server framework basati su Node.

Esso facilita l'utilizzo di un'implementazione API REST, agevolando il routing presente di base in Node, mappando ad un indirizzo URL o ad una specifica porta una funzione da svolgere.

Nel progetto è stato utilizzato primariamente per l'implementazione del sistema di routing. Inoltre, grazie alla possibilità di introdurre dei middleware alle richieste, ha agevolato enormemente anche l'introduzione del sistema di gestione dei permessi, implementato per l'appunto come un middleware ExpressJs.

4.1.4 Angular

Tra i framework più utilizzati allo stato dell'arte nello sviluppo frontend di applicazioni web vi è **Angular**. Nato come proseguimento del progetto AngularJS, questo prodotto oggi è gestito da Google; ciò permette ad Angular di essere stabile e continuamente aggiornato, e di disporre di una community molto ampia.

La sua adozione nel progetto è stata dettata dalla volontà da parte del team di approfondire tale framework, a discapito delle difficoltà nel suo apprendimento.

4.2 Metodologie

Lo sviluppo è stato portato avanti secondo una metodologia **agile**. Si sono sfruttati elementi di Scrum, come il concetto di Sprint, Daily Scrum, Backlog, organizzazione del lavoro in attività. Strumenti come **Trello** hanno permesso di organizzare il flusso di lavoro e la coordinazione tra i componenti del gruppo.

Non essendo il progetto commissionato da un utente esterno, gli stessi membri del team fungono sia da committenti che da sviluppatori. L'analisi dei requisiti, infatti, si è fatta prendendo come riferimento le esigenze e le funzionalità che un ipotetico utilizzatore dell'applicazione potrebbe richiedere.

Si è poi rivelata di particolare importanza la comunicazione frequente tra i vari membri del gruppo, e l'esecuzione alternata di alcune specifiche parti del progetto in parallelo con delle parti di "programmazione di gruppo", nelle quali ogni membro potesse spiegare agli altri la specifica tecnologia approfondita.

Il workflow adottato a livello di repository è **GitHub Flow**, una versione semplificata di GitFlow per progetti di piccole-medie dimensioni. Prevede che ogni nuova feature venga sviluppata e testata su un apposito branch, poi riversato nel master al raggiungimento di una versione stabile.

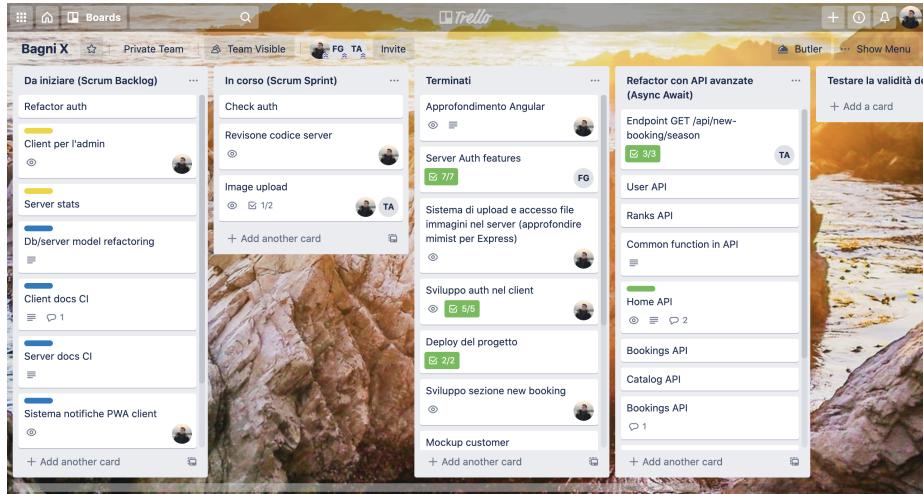


Figura 4.1: Estratto del backlog di progetto su tecnologia Trello.

4.3 Tecnologie per la sicurezza

L'applicazione prevede l'inserimento di dati sensibili da parte degli utenti, sia per quanto riguarda l'amministratore dello stabilimento balneare, sia per i clienti.

Non vi deve essere modo da parte di utenti malevoli di penetrare all'interno del sistema; inoltre, per gli utenti registrati va evitata la possibilità di “privilege escalation”, probabilmente molto nociva per il sistema.

Si è posta particolare attenzione alla protezione di questi dati. Pur non essendo il sistema un vero e proprio sistema in produzione, e avendo ignorato per praticità alcune best practices di deploy (ad esempio come la configurazione di regole di sicurezza avanzate per i vari container dell'applicazione), sono state poste in atto diverse pratiche per porre al sicuro i dati.

4.3.1 Middleware di autenticazione custom

Come precedentemente accennato, il controllo dei permessi lato API viene svolto tramite un apposito middleware, pensato per integrarsi con il framework ExpressJs.

Nella pratica, tutte le richieste ricevute dal server in direzione dell'API, vengono inoltrate al middleware d'autenticazione, e inoltrate al livello successivo (ovvero il controller) solo qualora la richiesta rispetti determinati requisiti di sicurezza. In caso contrario, il server risponde al mittente con una risposta 401 (unauthorized). Sono stati qui implementati i vari livelli di permessi citati nell'analisi di requisiti, ognuno con un proprio set di richieste autorizzate:

- **Livello libero:** alcune richieste si configurano come completamente libere, e possono essere effettuate da qualunque utente senza misure restrittive

sull'identità del mittente. Tra queste, le GET su homepage e news, o le POST per registrazione e login;

- **Livello customer:** altre richieste ancora sono possibili essere portate avanti solo se si è autenticati almeno come customer. Ad esempio, le richieste che riguardano la creazione di una prenotazione;
- **Livello customer-specific:** richieste eseguibili solo se si è uno specifico customer: ad esempio ogni customer può visualizzare solo le proprie prenotazioni;
- **Livello admin:** richieste eseguibili solo se si è un admin. La maggior parte dei dati visualizzati nell'area protetta presentano questo livello;
- **Livello admin-specific:** riguarda la possibilità di modificare i dati utente dello specifico admin, come username o password;
- **Livello root:** il livello gerarchicamente più alto. Può eseguire tutte le richieste sull'API precedenti, con in aggiunta la possibilità di creare e rimuovere admin.

4.3.2 JWT

Lo standard de-facto che si utilizza oggigiorno per la gestione dell'autenticazione all'interno di applicazioni web-based è il cosiddetto **JSON Web Token**. Questo sistema di criptazione delle informazioni permette di far comunicare due parti in maniera sicura.

Il JWT rappresenta una stringa criptata detta “token” di lunghezza variabile, generata automaticamente da vari servizi disponibili open source. La sicurezza di questa metodologia sta nella riservatezza delle chiavi con le quali si criptano le informazioni.

Nel nostro sistema è stato utilizzato il **meccanismo di criptazione a chiave simmetrica**; la chiave corrisponde ad una stringa segreta le cui metodologie per tenerla riservata esulano dagli scopi di questo progetto. Andando a criptare i dati corrispondenti ad un utente specifico, questo permette di creare un meccanismo di autenticazione sicuro ed efficace.

Il JWT contiene al suo interno varie informazioni che possono essere immesse quando viene creato; tali elementi possono essere letti solo da chi possiede la stessa chiave simmetrica con cui è stato generato il token (nel nostro caso, il server). Il JWT contiene le seguenti informazioni:

- **ID utente:** corrisponde all'identificativo dell'utente univoco all'interno del sistema. Questa informazione permette di controllare se l'utente esiste effettivamente nel database, e se il token risulta essere valido;
- **Audience:** rappresenta il campo standard utilizzato per definire il livello di permessi dell'utente. Nel nostro caso, permette di distinguere gli utenti **customer** dagli utenti **admin**, e ancora dall'utente **root**;

- **ExpiresIn:** rappresenta l'informazione relativa alla validità temporale token, in quanto risulta essere una best practice revocare il JWT dopo un certo lasso di tempo.

All'interno dell'elaborato, ogni utente esegue i seguenti step per autenticarsi ed effettuare richiesti all'API:

1. **Registrazione:** i clienti hanno la possibilità di registrarsi, tramite l'apposita interfaccia web, e di inserire i propri dati. Gli utenti admin vengono invece inseriti nel sistema direttamente dall'admin root;
2. **Ottenimento del JWT:** A seguito della registrazione, l'utente può effettuare il login al sistema, tramite l'interfaccia web, inserendo username (o email) e password. In caso di credenziali corrette, il sistema restituisce il JWT relativo allo specifico utente, con all'interno indicazioni sul proprio livello di permessi;
3. **Effettuazione delle richieste:** l'utente autenticato può quindi effettuare richieste all'API, includendo alle proprie richieste l'header **authorization**, con l'indicazione del proprio JWT. In base al livello di permessi, esso potrà effettuare solo un limitato sottoinsieme di richieste.

Il JWT risulta essere completamente invisibile all'utente che utilizza l'applicazione. La memorizzazione e la gestione del token vengono effettuate in maniera integrata attraverso il frontend Angular, che memorizza il JWT sotto forma di cookie.

4.3.3 Modulo Sanitizer

Per evitare problemi di sicurezza dovuto all'inserimento di dati erronei o nocivi all'interno dell'applicazione, è stato previsto un sistema di sanificazione dell'input. In maniera automatica, viene controllato ogni valore che viene inserito dall'utente: questo deve corrispondere al formato che l'applicazione si aspetta e inoltre deve evitare di creare potenzialmente dei danni al sistema. Tale controllo viene effettuato tramite il modulo NPM **Santizer**.

4.3.4 BcryptJs

Un'ulteriore problematica sovviene per la gestione delle password all'interno del database MongoDB. Viene considerato da tempo obbligatorio salvare le informazioni importanti utilizzando delle tecniche di cifratura ed evitando quindi che siano memorizzate in chiaro.

Tra le funzioni di hashing password più utilizzate, vi è **Bcrypt**. Basata su una cifratura Blowfish, Bcrypt permette di garantire standard di cifratura elevati, grazie alla possibilità di essere adattiva, ovvero di variare il numero di iterazioni dinamicamente, e di incorporare il valore di sale automaticamente in modo da proteggere le informazioni da possibili attacchi basati su dizionari.

Nel nostro progetto è stata utilizzato il modulo **NPMBCryptJs**, che implementa tale algoritmo. Il suo utilizzo è stato fondamentale e allo stesso tempo molto rapido, permettendo di salvare in sicurezza le password degli utenti.

4.3.5 Guard Angular

Un’ultima misura di sicurezza è stata invece implementata lato client, e consiste nell’adozione delle **guard Angular** per restringere l’accesso a determinate schermate dell’applicazione.

Le guard altro non sono dei service Angular, atti ad effettuare dei controlli preliminari all’accesso delle route client dell’applicazione, effettuando il redirect nel caso in cui l’utente non sia autorizzato ad accedere a quella rotta. Nel contesto del nostro progetto, queste sono state utilizzate per inibire l’accesso all’area protetta degli utenti customer e admin, nel caso in cui questi non disponessero dei permessi necessari. Tali permessi vengono determinati in base alla presenza o all’assenza del rispettivo JWT tra i cookie dell’applicazione.

4.4 Tecnologie di supporto

4.4.1 Docker

Docker è un progetto open source che automatizza il deployment di applicazioni all’interno di container software, fornendo un’astrazione aggiuntiva grazie alla virtualizzazione a livello di sistema operativo di Linux. I contenitori hanno la possibilità di interagire tra loro comunicando con la rete nativa di Docker.

Docker è stata selezionata in ambito di progetto come tecnologia per il deployment dei vari moduli software. Questo in quanto essa risulta essere lo standard de facto per il deploy di moduli software moderni.

4.4.2 GitHub, Actions e Pages

GitHub è uno dei più diffusi servizi di hosting per progetti software. La piattaforma è stata scelta sia in quanto ampiamente conosciuta dai membri del team, sia per le tecnologie di supporto che offre. In particolare:

- **Pages:** di fatto un servizio di hosting offerto da GitHub, che permette di visualizzare pagine statiche direttamente dalla propria repository. Il servizio è stato utilizzato sia per ospitare una **versione web della documentazione degli endpoint API**, sia per ospitare **una demo del servizio**, collegata direttamente alla repo di progetto;
- **Actions:** servizio che permette l’esecuzione di codice per l’implementazione di Continuous Integration e Delivery, utilizzato per il testing automatico del codice direttamente sulla repo, e per la delivery automatica della demo al sopracitato servizio Pages.

4.4.3 Postman

Postman è un tool multipiattaforma utile per il testing manuale delle proprie API, permettendo la realizzazione di chiamate HTTP tramite interfaccia visuale. Il software è molto diffuso, in quanto permette di testare la validità del backend senza l'utilizzo di un client completo. Il programma è aggiornato e consente persino l'utilizzo dei più innovativi linguaggi Web come GraphQL.

Nel progetto è stato utilizzato per valutare e testare le query realizzate nel server.

4.4.4 Swagger

Swagger è un linguaggio di descrizione di interfacce REST, utilizzato comunemente nella creazione di documentazione per API di RESTful.

È risultato assai utile in fase di progettazione per definire, prima ancora dello sviluppo vero e proprio, le signature e gli oggetti JSON ritornati dai vari endpoint. Il file di documentazione dell'API è poi stato posto in una repository apposita, configurata con un template Pages per la visualizzazione **tramite interfaccia web** della stessa.

4.4.5 Adobe XD

Adobe XD è una soluzione di UI/UX design per la creazione di prototipi di siti web e app mobile, disponibile anche in versione gratuita. Il software si è rivelato particolarmente utile, sia per la sua potenza e facilità di utilizzo, che per il suo supporto alla creazione di prototipi navigabili.

Questi possono essere anche visualizzati su smartphone, tramite l'apposita app. Ciò ha permesso nella prima fase di progetto di mostrare il prototipo, non ancora sviluppato, a soggetti esterni.

Capitolo 5

Codice

In questo capitolo si andrà a descrivere il codice sviluppato, trattandone gli aspetti generali relativi alla struttura utilizzata.

5.1 Struttura della repository

Il codice è liberamente consultabile all'interno della repository GitHub **Bagni X Booking System**.

Al livello root della repo sono contenuti vari file per la containerizzazione automatica della soluzione, e vari file di metadati, quali licenza, readme e quant'altro. I dettagli riguardanti build e deploy del progetto vengono trattati nell'apposito capitolo.

I sorgenti più significativi sono contenuti all'interno delle directory `/client` e `/server`, contenenti rispettivamente il codice relativo a frontend e backend. La cartella `/storage-init` contiene invece file per pre-popolare il database Mongo in fase di deploy, con dei dati di base, oltre a pre-popolare il container server con eventuali asset immagini.

5.2 Server

Il codice del server è stato realizzato interamente in linguaggio JavaScript, e si appoggia sul framework NPM. L'API del sito è stata ampiamente documentata tramite Swagger, ed è stata predisposta **una repo apposita** con una **pagina GitHub Pages dedicata** alla consultazione di questa.

Lanciando `server.js`, la libreria Express avvia un server alla porta 3000, predispone le route dell'API (identificate ll'URL `/api/<route-name>`), uno spazio statico per l'accesso agli assets (all'URL `/asssets/*`), e delega la gestione di tutte le altre route al client Angular, permettendo in questo modo l'utilizzo di deep link all'interno del client.

L'entry point del server è il file `server.js`. All'interno di questo è facile visualizzare le dinamiche sopra descritte. Il codice per la gestione delle route è organizzato in 4 principali sub-directory:

- `/src/models`: la directory contiene i vari schemi e model Mongoose, atti per l'appunto a modellare le entità utilizzate nel database vero e proprio, e ad interagire con lo stesso. Ogni file contiene schema e modello di ogni collection “di primo livello”, ovvero memorizzate direttamente nella root del database. Nella subdirectory `/freeSchemas` sono invece contenuti degli schemi utilizzati come sub-collection, in alcune delle entità di primo livello;
- `/src/controllers`: la directory contiene i vari controller, ovvero le funzioni invocate da ogni route dell'API. Queste hanno lo scopo di processare i dati, validarli, sanificare ed inserirli nel database, oltre a ritornare ai mittenti stessi le risposte desiderate;
- `/src/routes`: la directory contiene il codice responsabile della generazione delle route, ovvero il collegamento tra i vari endpoint e il codice da essi invocato, contenuto all'interno di `/src/controllers`;
- `/src/utils/`: la directory contiene vari moduli di utility, suddivisi per funzionalità, utilizzati in vari punti dell'applicazione. Di particolare importanza è il modulo `authentication.js`, responsabile del controllo degli accessi dell'API. Questo implementa un middleware Express, che intercetta le richieste prima che arrivino agli endpoint, e nega l'accesso tramite un 401 a tutte le richieste non autorizzate, tramite dei controlli sul JWT.

5.3 Client

Il client è stato sviluppato tramite il framework Angular. Il codice si compone quindi di parti in TypeScript, HTML, SCSS, e alcuni file di configurazione in JSON. Esso si propone di essere un rich client SPA, atto a mimare il più possibile le funzionalità di un'app nativa. Per questo il client è stato implementato come una PWA. L'idea iniziale di progetto prevedeva anche di implementare un sistema di notifiche di prenotazione da inviare ai vari client, sfruttando le funzionalità offerte da GCM tra le altre. L'idea è stata però accantonata, principalmente a causa delle difficoltà legate alla necessità, per i regolamenti di Google, di disporre di un server reale che comunichi in HTTPS.

Va poi sottolineato che la directory `/client` contiene solo il **sorgente** del client Angular. In modalità di produzione, la versione buildata del client viene posta all'interno del server, nella directory `/server/client`, come poi descritto nel capitolo Deployment.

Si è strutturato il client secondo le best practices Angular. La struttura interna dell'app segue quella proposta da **Mathis Garberg** e dall'Angular Style Guide, con delle piccole variazioni. In generale, l'app si suddivide in sottomoduli, che seguono la seguente struttura a directory:

- la directory `/core`, e l'omonimo modulo, sono responsabili di raggruppare service, interceptor, guard, e di tutto quello che richiede all'interno dell'applicazione, di funzionare come un Singleton;
- la directory `/pages`, e l'omonimo modulo, raggruppano i componenti che rappresentano le varie schermate dell'applicazione. Se la pagina richiede dei componenti utilizzati solamente nel proprio contesto, questa a sua volta contiene un modulo per raggruppare gli stessi;
- la directory `/shared`, e il modulo corrispondente, contengono incece componenti, directive, pipe, altri moduli e utility varie utilizzate in molteplici punti dell'applicazione. Il modulo shared viene, ad esempio, importato e utilizzato all'interno di ogni pagina. `/shared` contiene anche le le import di tutti i **Material Component** utilizzati, così da accedervi a partire da un unico modulo comune.

Dettagli più specifici, relativi al funzionamento del codice, possono essere ricavati dalla consultazione del codice stesso, il quale è correlato da un ricco contenuto di commenti.

Capitolo 6

Test

Durante lo svolgimento del progetto sono stati portati avanti vari test, atti a valutare usabilità e correttezza del codice. In varie fasi dello sviluppo sono stati poi coinvolti degli utenti esterni, allo scopo di avere indicazioni aggiuntive per lo sviluppo UX/UI.

6.1 Continuous Integration

Nella repository remota sono state predisposte delle tecniche di Continuous Integration automatizzate, tramite l'utilizzo delle funzionalità offerte da GitHub Actions. Non avendo utilizzato particolari librerie dedicate per il testing del codice, sia lato client che lato server, gli script CI hanno avuto lo scopo primario di tentare la build dei container coinvolti nel sistema, e di provare a lanciarli.

Ciò risulta particolarmente utile per il client, il quale, essendo implementato in TypeScript, deve passare per una fase di compilazione, che fallisce in caso di errori, evidenziando possibili errori all'interno del codice.

Il tentativo di build e lancio dei container permette inoltre di individuare potenziali problemi all'interno del codice del server, nella configurazione del container database, e nei loro collegamenti.

6.2 Testing del server

Sono state effettuate varie tornate di test per effettuare la validità degli endpoint dell'API.

I test sono stati effettuati in larga parte con tecniche black box, tramite l'applicativo Postman, utilizzato per sottoporre al server delle chiamate RESTful. L'applicativo è stato utile sia per mettere alla prova request in direzione di, e response emanate dal server, sia per testare il sistema di autenticazione e permessi, grazie alla possibilità di impostare facilmente un bearer token a qualsiasi richiesta.

Test di tipo white box sono stati portati avanti in alcuni passaggi particolarmente delicati, lungo tutto il processo di sviluppo, tramite l'utilizzo del debugger fornito da Node.

6.3 Testing del client

Lato client, è stato possibile focalizzarsi sullo sviluppo delle interfacce, inizialmente in maniera del tutto parallela allo sviluppo server, grazie all'introduzione di un particolare interceptor con il ruolo di **fake backend**. Questo è stato pensato per intercettare le richieste HTTP effettuate dal client, e di fornire delle risposte completamente mock. Ciò ha permesso di concentrarsi sullo sviluppo UX/UI, senza dover incorrere in problemi legati alla mancanza di un vero e proprio backend, nelle prime fasi.

In generale, test black box sono stati effettuati durante tutto il processo di sviluppo tramite server fornito da Angular tramite il comando `ng serve`, mentre per i test white box si è beneficiato delle funzionalità di debug offerte dal browser Google Chrome.

Non sono state invece utilizzate particolari librerie specifiche di test.

6.4 Test di integrazione

Un'ultima importante fase di verifica ha riguardato l'integrazione tra i due moduli inizialmente sviluppati in maniera separata, il client e il server. Nelle ultime fasi è stato infatti disabilitato il fake backend del client, e si è preceduto a dei test tali da coinvolgere entrambi i moduli, per verificare eventuali incongruenze tra gli endpoint e il codice del client atto ad effettuare le richieste. È stata pensata un'apposita procedura di build per la disposizione di tali test, tale da sfruttare un'intanza MongoDB presente in localhost, e una versione del server contenente una build client dedicata.

6.5 Interazione con utenti esterni

Durante varie fasi dello sviluppo, si è interagito con soggetti esterni al progetto, sia potenziali clienti di un'app del genere, che potenziali acquirenti. Nelle fasi iniziali, è stato mostrato loro il prototipo generato tramite Adobe XD, mentre nelle ultime fasi è stato possibile mostrare loro direttamente la demo disponibile [in questa pagina](#).

Si sono quindi raccolti feedback generati dall'interazione con questa. Questi sono stati molto utile per dare una direzione più precisa allo sviluppo dell'interfaccia e delle funzionalità.

Capitolo 7

Deployment

7.1 Installazione e messa in funzione

L'installazione e la messa in funzione dell'infrastruttura server si basa su tecnologia Docker. Per l'orchestrazione dei container viene invece utilizzato docker-compose. Questi applicativi sono quindi prerequisiti per l'installazione dell'applicativo nella sua versione di produzione.

Quindi, una volta scaricati i sorgenti dalla repo, è possibile generare e dispiegare i container tramite i comandi:

```
docker-compose build  
docker-compose up
```

Sarà quindi possibile accedere all'applicazione all'indirizzo:

```
http://localhost:3000
```

Dietro le quinte, il comando genera:

- **Il container MongoDB:** questo si basa sull'immagine ufficiale MongoDB fornita su DockerHub. In questa fase, il database viene anche pre-popolato, tramite lo script `mongo-init.js`, contenuto all'interno della directory `/storage-init`. Viene inoltre esposta alla rete interna di docker (e non all'esterno di questa) la porta 27017, per l'accesso al database da parte del server;
- **Il container Server:** il container si basa sull'immagine ufficiale di Node fornita su DockerHub, e viene costruito a partire da una build multi-stage. Nella prima fase, viene generato il codice compilato del client. Nella seconda fase, viene adibito il container del server, aggiungendo a quest'ultimo una sub-directory nel quale viene posto il compilato del client Angular, e una seconda sub-directory per gli asset pre-popolati.

La repository dispone inoltre di un **Makefile**, che consente di effettuare le varie build di produzione e di test ancora più facilmente, in maniera automatizzata. Tra i comandi più importanti:

- **make** (o **make deploy**): dispiega tutti container utilizzati per database e web server (per l'attivazione in produzione);
- **make server-dev**: genera dipendenze e build per il web server e avvia un'istanza locale del web server direttamente su Node, senza passare per la containerizzazione Docker (utile in fase di sviluppo server). Richiede un'istanza MongoDB in esecuzione in `localhost:27017`;
- **make client-dev**: genera dipendenze e build per il client Angular e avvia un'istanza locale del server Angular di sviluppo, senza passare per la containerizzazione Docker (utile in fase di sviluppo client). Non richiede istanze server e database, basando le risposte su un backend mock;
- **make integration**: genera dipendenze e build per il client Angular e per il web server, avviando un'istanza locale del web server comprensivo di client Angular (utile in fase di integrazione client-server). Richiede un'istanza MongoDB in esecuzione in `localhost:27017`;

7.2 Demo su GitHub Pages

Si ricorda inoltre che una demo per la visualizzazione del frontend dell'elaborato è disponibile a [questo indirizzo](#). Tale demo viene generata automaticamente tramite uno script GitHub Actions, che si attiva ad ogni push nel branch master della repository. Lo script crea in automatico una versione compilata del client, la pone sul branch `gh-pages` e la rende disponibile alla visualizzazione, tramite la tecnologia Pages fornita da GitHub. Si ricorda però che il sito è solo una demo, e come tale non si collega ad alcun backend: i dati visualizzato sono ricavati dal fake backend di test, interno al client Angular.

Capitolo 8

Conclusioni

Lavorare su questo progetto si è rivelato estremamente utile per studiare una moltitudine di tecnologie differenti, prima d'ora mai approfondate a questo livello di dettaglio. A partire da quelle dello stack MEAN, fino a tutte quelle a supporto dello sviluppo.

Il progetto è stato configurato in maniera tale da essere effettivamente utilizzabile, “vendibile” sul mercato. Cercheremo quindi di tener vivo questo progetto, e di migliorarlo ancora, in ottica di un suo auspicabile utilizzo all'interno di un vero stabilimento balneare.