

UNIVERSITÀ DEGLI STUDI DI TORINO

DIPARTIMENTO DI INFORMATICA

CORSO DI LAUREA IN INFORMATICA



TESI DI LAUREA IN INFORMATICA

**Sviluppo di un applicativo RESTful per il
controllo degli accessi ai tempi del Covid-19
utilizzando Spring e React**

Relatore

Chiar.mo Prof. Francesco BERGADANO

Tutor Aziendale

Dott. Simone FERRARO

Candidata

Deborah DORE

ANNO ACCADEMICO 2019-2020

Ai miei genitori

Dichiarazione di Originalità

Dichiaro di essere responsabile del contenuto dell'elaborato che presento al fine del conseguimento del titolo, di non avere plagiato in tutto o in parte il lavoro prodotto da altri e di aver citato le fonti originali in modo congruente alle normative vigenti in materia di plagio e di diritto d'autore. Sono inoltre consapevole che nel caso la mia dichiarazione risultasse mendace, potrei incorrere nelle sanzioni previste dalla legge e la mia ammissione alla prova finale potrebbe essere negata.

Sommario

La pandemia in corso non ha risparmiato nessuna categoria di lavoratori. In particolare, i ristoratori sono stati sommersi dalle regole imposte dal governo in conseguenza al covid-19. Il lavoro svolto in questa tesi rende più semplice seguire una di queste regole in particolare: la gestione delle presenze. Le attuali restrizioni impongono ai ristoratori la registrazione di ogni cliente che transita nel locale in modo da poterlo contattare velocemente in caso di necessità. Ad oggi, le presenze vengono registrate tramite l'utilizzo di carta e penna. L'applicativo realizzato si pone come principale obiettivo quello di automatizzare il controllo degli accessi. Da una parte, il cliente avrà a disposizione l' applicazione mobile con un QRCode unico e personalizzato. Dall'altra, il ristoratore, attraverso l'applicazione realizzata sia per mobile sia per web, potrà: scansionare il QRCode del cliente salvandolo fra gli ospiti che hanno transitato nel locale, ricevere notifiche su eventuali positività rilevate e avvalersi di ulteriori funzionalità utili al completamento della gestione.

Indice

Introduzione	2
1 Architettura	4
1.1 Il Pattern MVC	4
1.2 Sistema Client-Server	5
1.3 Protocollo HTTP	6
1.4 Web Service	9
1.5 REST & Restful API	11
2 Tecnologie	15
2.1 Front End	15
2.1.1 HTML	15
2.1.2 CSS	18
2.1.3 Javascript	24
2.1.4 React	27
2.2 Back End	40
2.2.1 Maven	40
2.2.2 Spring Framework	40
2.2.3 JSON Web Tokens	48
2.2.4 Google Firebase	50
2.3 Gestione dei dati	52
2.3.1 MySQL	52
2.3.2 MyBatis	52
2.4 Deployment	55
2.4.1 Docker	55
2.5 Version Control	57
2.5.1 GIT	57
3 Progetto	58
3.1 Introduzione	58
3.2 Front End	60

3.2.1	QRestaurant	60
3.2.2	QRestaurant Business	67
3.2.3	QRestaurant Business Web	77
3.3	Back End	83
3.3.1	Architettura a tre livelli	83
3.3.2	Struttura del Back-end	84
3.3.3	Creazione di Servlet ad hoc	85
3.3.4	Imbustamento della Risposta	87
3.3.5	Gestione delle Eccezioni	87
3.3.6	Livello di sicurezza e autenticazione	88
3.3.7	Ambienti di Sviluppo	91
3.3.8	Notifiche push con Firebase	92
3.3.9	Branching Strategy	92
3.4	Deployment	94
4	Sviluppi Futuri	95
Elenco delle figure		95
Elenco delle tabelle		98
Riferimenti bibliografici		99
Ringraziamenti		101
A Listato		102
A.1	Front-end	102
A.1.1	Rendere i componenti sicuri: PrivateRoute.jsx	102
A.2	Back-end	104
A.2.1	Configurazione delle servlet: ServletConfiguration.java	104
A.2.2	Imbustamento della risposta: Response.java	105
A.2.3	Body della risposta	106
A.2.4	Gestione delle eccezioni: FailureException.class	106
A.2.5	Configurazione di Spring Security: SecurityConfiguration.java	107
A.2.6	Scelta dell'ambiente di sviluppo: EnvironmentConfiguration.java	108
A.3	Deployment	110
A.3.1	DockerFile	110
A.3.2	Docker Compose	110

Introduzione

Questa tesi è il risultato finale di un lavoro di ricerca e sviluppo svolto presso l'azienda di consulenza informatica *Certimeter S.R.L.* All'interno dell'azienda ho incontrato persone competenti e disponibili che mi hanno seguita durante tutto il percorso. Ho inoltre avuto modo di comprendere e utilizzare le varie tecnologie impiegate tutt'oggi nel mondo del lavoro. Grazie all'inserimento in un team ho avuto la possibilità di confrontarmi con le varie dinamiche aziendali sotto la direzione di una figura competente.

L'obiettivo del progetto è stato quello di sviluppare un'applicazione che automatizzasse il processo di registrazione delle presenze in un ristorante ai tempi del COVID-19 al fine di renderlo più immediato. L'applicativo si pone come sostituto dell'attuale sistema che prevede la registrazione manuale dei clienti. L'attività che ho svolto è stata quella di sviluppare un applicativo web che esponesse dati ad applicazioni mobile e ad un front-end web tramite web services REST. Nel corso dello stage, ho inoltre sviluppato un front-end web che consumasse i web service esposti.

Nel primo e nel secondo capitolo viene presentata un'introduzione generale all'architettura e alle tecnologie impiegate. In particolare si è scelto di utilizzare il framework Spring Boot per il back-end e la libreria React per il front-end. Nel terzo capitolo viene introdotto il progetto nel dettaglio. Il quarto ed ultimo capitolo contiene le conclusioni e possibili sviluppi.

Capitolo 1

Architettura

1.1 Il Pattern MVC

Il pattern *Model-View-Controller* spesso abbreviato con **MVC** è un pattern architettonico utilizzato soprattutto nello sviluppo software. Lo scopo del pattern è organizzare l'architettura delle applicazioni complesse che hanno interfaccia grafica in componenti, ognuno con uno specifico compito, in modo da aumentarne la modularità. Tali componenti sono:

- Il **Modello (Model)** è il componente principale. Si occupa di gestire i dati e la logica dell'applicazione. Esso modella il problema da risolvere. Rappresenta lo stato dell'applicazione.
- La **Vista (View)** si occupa di visualizzare lo stato interno del modello e gestisce l'interazione con l'utente. È l'interfaccia dell'applicazione.
- Il **Controllore (Controller)** riceve i comandi dall'utente attraverso la vista e controlla il flusso di dati nel programma. Incapsula la logica dell'applicazione.

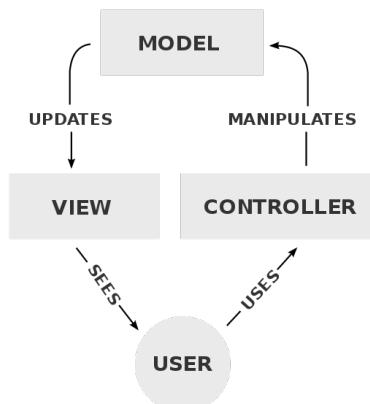


Figura 1.1
Pattern MVC schematizzato

I vantaggi nell'utilizzo di questo pattern sono molteplici. Le applicazioni che seguono questo pattern sono facilmente riutilizzabili, comprensibili, inoltre la modifica di un componente non implica che anche gli altri vengano modificati.

1.2 Sistema Client-Server

Il termine *client-server* indica un'architettura di rete genericamente formata da un *client* che si connette ad un *server* per poter usufruire dei servizi offerti da quest'ultimo, come per esempio l'utilizzo di determinate risorse.

Il software *client* è principalmente utilizzato come interfaccia per poter accedere ai servizi offerti dal server.

Il software *server* è più complesso e oltre alla gestione dei molteplici accessi contemporanei da parte di più client, deve implementare la condivisione, la sicurezza, l'allocazione e il rilascio di risorse e dati.

Il modello di comunicazione client-server è definito *asimmetrico*: il client designa esplicitamente il server a cui vuole connettersi mentre il server rimane sempre in attesa di ricevere richieste.

Grazie alla presenza di un solo server, più client possono condividere le stesse risorse lasciando che sia il server ad occuparsi di risolvere eventuali conflitti di utilizzazione. È però possibile che molti client vogliano poter accedere ad un servizio creando sovraffollamento; Questo può succedere se nel processo server non vengono prese misure adeguate per regolare le richieste o aumentare le risorse disponibili. Il sovraffollamento potrebbe anche essere artificiale e dovuto ad attacchi informatici del tipo DOS¹ o DDOS² in occasione dei quali un alto numero di client viene manipolato allo scopo di effettuare un numero di richieste nettamente superiore a quelle che il servizio sotto attacco può gestire con la conseguenza di renderlo inutilizzabile.

All'inizio della comunicazione i processi client e server possono scambiarsi pacchetti di controllo prima di spedire effettivamente i dati reali. Queste procedure sono dette di *handshaking* e vengono utilizzate per preparare le due componenti alla comunicazione. Questo tipo di interazione è chiamato *connection oriented* e indica un'interazione nella quale viene stabilito un canale di comunicazione virtuale prima di iniziare lo scambio di dati. Ad esso si contrappone l'interazione *connectionless* dove non vi è connessione ma semplice scambio di dati. Un esempio di livello di trasporto orientato alla connessione è il protocollo TCP³ mentre il protocollo UDP

¹Denial of Service

²Distributed Denial of Service

³Transmission Control Protocol

è uno dei principali protocolli di rete senza connessione.

Lo stato di un’interazione può essere diviso in due tipi: *stateful* o *stateless*. Con il termine *stateful* si intende un tipo di connessione che ad ogni richiesta tiene traccia dei dati del client. Una connessione *stateless* invece non conserva nessuna informazione sullo stato. La scelta fra i due tipi di interazione deve tenere traccia delle caratteristiche dell’applicazione: un’interazione *stateful* richiede che il server sia in grado di identificare il client mentre un’interazione *stateless* è possibile solo se il protocollo applicativo è progettato con operazioni idempotenti.

1.3 Protocollo HTTP

L’*Hypertext Transfer Protocol (HTTP)* è un protocollo a livello applicativo usato per la trasmissione di informazioni sul web. La prima versione del protocollo venne sviluppata negli anni 80 da Tim Berners-Lee al CERN di Ginevra e costituiva il nucleo base del World Wide Web. Venne inizialmente ideato con lo scopo di istituire un metodo comune per la condivisione delle informazioni tra la comunità di fisici. L’idea principale era quella di basare il protocollo su una struttura client-server e su un sistema di richiesta e risposta che utilizzasse lo standard ASCII⁴ eseguito sul protocollo TCP/IP.

Possiamo distinguere fra due tipi di messaggi HTTP: **messaggi di richiesta** inviati dal client al server e **messaggi di risposta** trasmessi dal server al client le cui parti più importanti sono:

- **Riga di richiesta** nel caso di messaggi di richiesta o **riga di stato** nel caso di messaggi di risposta.
- **Header** contenente informazioni aggiuntive. Nei messaggi di richiesta le informazioni riguardano *l’host* ossia il nome del server a cui si riferisce la richiesta, lo *User-Agent* che identifica il tipo di client e i *Cookie* che contengono informazioni a lungo termine lato client. Nei messaggi di risposta, lo header contiene informazioni riguardanti il tipo e la versione del *Server* e il *Content-Type* che identifica il tipo di contenuto restituito (text/html, text/plain, image/jpg sono alcuni esempi).
- **Body** ossia il corpo del messaggio.

La *riga di richiesta* è composta dal *metodo*, dall’*UR⁵* che indica l’oggetto della richiesta e dalla *versione del protocollo* utilizzato. Il metodo della richiesta può essere individuato fra i seguenti:

⁴Codice per la codifica di caratteri[3], American Standard Code for Information Interchange

⁵Uniform Resource Identifier

-
- GET per il recupero dei contenuti della risorsa identificata dall'URI.
 - HEAD il cui comportamento è simile al metodo GET ma restituisce solamente i campi dell'header.
 - POST per la creazione di una nuova risorsa.
 - PUT per l'aggiornamento o la modifica dello stato di una risorsa.
 - DELETE per l'eliminazione di una risorsa.
 - PATCH per l'aggiornamento o la modifica di parti dello stato di una risorsa.
 - TRACE attraverso il quale può essere ricostruito il percorso di una richiesta HTTP dal client al server e viceversa.
 - OPTIONS con il quale il client può richiedere al server quali metodi quest'ultimo supporta.
 - CONNECT che stabilisce una connessione diretta e protetta attraverso un proxy.

La *riga di stato* riporta invece un codice a tre cifre che identifica il successo o il fallimento della richiesta. Tali cifre possono essere catalogate nel seguente modo:

- 1xx: Informational, usato per messaggi informativi.
- 2xx: Successful, rappresenta una richiesta soddisfatta.
- 3xx: Redirection, la risorsa richiesta non è disponibile e nella risposta è indicato come ottenerla.
- 4xx: Client Error, la richiesta è incorretta.
- 5xx: Server Error, la richiesta non può essere soddisfatta per un errore del server.

```

POST /manager/web/login HTTP/1.1
Content-Type: application/json
Access-Control-Allow-Origin: *
Accept: application/json, text/plain, */*
Authorization: Bearer null
Accept-Language: it-it
Accept-Encoding: gzip, deflate
Host: localhost:8080
Origin: http://localhost:3000
Content-Length: 60
Connection: keep-alive
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15
_7) AppleWebKit/605.1.15 (KHTML, like Gecko) Version
/14.0.2 Safari/605.1.5
Referer: http://localhost:3000/

```

*Figura 1.2
Esempio di HTTP Request*

```

HTTP/1.1 200
Access-Control-Allow-Origin: *
Content-Type: application/json
Pragma: no-cache
Keep-Alive: timeout=60
X-XSS-Protection: 1; mode=block
Expires: 0
Transfer-Encoding: Identity
Cache-Control: no-cache, no-store, max-age=0, must-re
validate
Date: Sat, 02 Jan 2021 18:47:06 GMT
Connection: keep-alive
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
Vary: Origin, Access-Control-Request-Method, Access-C
ontrol-Request-Headers

```

*Figura 1.3
Esempio di HTTP Response*

La prima versione del protocollo chiamata anche *protocollo ad una riga* corrisponde alla **versione 0.9**, che permetteva solo il recupero di un file HTML dal server.

Nel 1996 è stata introdotta la versione **HTTP/1** dall'IETF⁶ che presenta le seguenti caratteristiche:

- *Connectionless*: il client stabilisce la connessione e invia la richiesta. Il server risponde e chiude la connessione. Ogni nuova richiesta necessita di una nuova connessione.
- *Stateless*: nessun dato riguardante il client viene conservato.
- *Indipendente dai media*: qualunque tipo di file può essere trasmesso a condizione che sia il client sia il server sappiano come gestirlo.

La popolarità del protocollo e il traffico HTTP che ne è derivato evidenziano le limitazioni principali dell'HTTP/1.0: la chiusura della connessione e la gestione di una singola richiesta alla volta.

Nel 1999 viene ufficializzata una nuova versione del protocollo tutt'oggi utilizzata: è il protocollo **HTTP/1.1** che ha apportato migliorie sostanziali rispetto alla versione precedente. Di seguito alcune fra le più importanti:

- Attraverso l'*Http Pipeling* il client può inviare più richieste allo stesso server senza aver ancora ricevuto risposta alle precedenti domande.
- Viene data al client la possibilità di mantenere la connessione attraverso un campo *keepalive* individuato nell'header.

⁶Internet Engineering Task Force

-
- Il client ha la possibilità di *richiedere solo una parte di un documento* riducendo così il carico sul server e il traffico di dati.
 - In HTTP/1.0 non era possibile specificare il nome dell'host. Di conseguenza il server web doveva essere associato ad un solo indirizzo IP e quindi ad un solo dominio. La versione 1.1 risolve il problema permettendo di specificare nomi host multipli attraverso il campo *Host* nell'header.

Nel maggio 2015 è stato annunciata dall'IETF una nuova versione del protocollo: **HTTP/2**. Basato sul protocollo SPDY⁷, ha rivoluzionato il trasferimento dei dati in rete grazie alla multiplazione delle trasmissioni.

Questa tecnica permette di caricare un sito web in un'unica connessione. Altre importanti novità riguardano l'introduzione dello scambio dei pacchetti utilizzando un meccanismo basato sulla priorità, un header compresso per evitare informazioni inutili; l'utilizzo del codice binario per lo scambio delle informazioni e la possibilità di inoltrare dati prevedibili dal server al client prima che questi vengano richiesti.

In futuro potremmo trovarci ad utilizzare il nuovo protocollo **HTTP/3**, basato su UDP⁸, ancora in fase di definizione.

1.4 Web Service

Dalla definizione data dal W3C⁹: "*un Web Service è un sistema software progettato per supportare l'interoperabilità tra diversi elaboratori su una medesima rete oppure in un contesto distribuito.*"[31]

Questa caratteristica viene raggiunta associando all'applicazione un interfaccia software di solito descritta attraverso il *WSDL*¹⁰ che espone all'esterno i servizi offerti con i quali altri sistemi possono interagire tramite il protocollo HTTP.

Grazie all'interfaccia software e all'utilizzo di un protocollo comune, applicazioni scritte in diversi linguaggi e implementati su diverse piattaforme possono collaborare. L'utilizzo dei Web service permette inoltre il "disaccoppiamento" tra il sistema utente e il Web service: le modifiche su una parte non si riflettono sull'altra. Ciò consente una forte riusabilità del codice.

Come già riportato in precedenza, i Web service utilizzando il protocollo HTTP

⁷Protocollo a livello applicativo ideato da Google che aveva come scopo quello di veicolare il protocollo HTTP al suo interno in modo da ridurre la latenza delle pagine web senza perderne la compatibilità.

⁸User Datagram Protocol

⁹World Wide Web Consortium: organizzazione non governativa internazionale che ha come scopo quello di sviluppare tutte le potenzialità del World Wide Web.

¹⁰Web Service Description Language: linguaggio formale in formato XML.

"over" TCP sulla porta 80, l'unica lasciata aperta dai firewall per la comunicazione verso sistemi esterni e sulla quale transita il traffico HTTP dei browser.

A partire dall'interoperabilità fra le diverse applicazioni, possiamo quindi individuare diversi vantaggi legati all'utilizzo dei Web service:

- L'utilizzo di *standard aperti* rende i servizi facilmente comprensibili, utilizzabili e mantenibili dagli sviluppatori.
- Le comunicazioni fra questo tipo di applicazioni sono basate sul protocollo HTTP e pertanto attraversano i firewall *senza modifiche alle regole di sicurezza* di quest'ultimi.
- Vari servizi possono essere *combinati* per formarne altri integrati e complessi.
- L'*interfaccia* offre all'utente il risultato finale dell'esecuzione del servizio nascondendone la complessità architetturale.

Nonostante ciò, chiunque voglia implementare un Web service deve tenere conto dei seguenti aspetti:

- *Non esistono standard consolidati* per applicazioni critiche (es. transazioni distribuite).
- *Performance migliori* possono essere riscontrate in *approcci alternativi*.
- L'uso dell'HTTP permette ai Web service di *evitare le misure di sicurezza* dei firewall.

Al fine di ottenere la comunicazione fra i diversi Web service, questi ultimi devono seguire una serie di regole e protocolli individuati dalla **Pila Protocollare dei Web Service**. Tale insieme di protocolli è composto principalmente da quattro aree:

- *Trasporto del servizio*: responsabile del trasporto dei messaggi in rete. Solitamente vengono utilizzati il protocollo HTTP, SMTP¹¹, FTP¹² o XMPP¹³.
- *Formato dei messaggi*: tutti i dati scambiati sono formattati tramite tag XML oppure utilizzando lo standard SOAP, JAX-RPC, XML-RPC o REST.
- *Descrizione del servizio*: utilizzando il linguaggio WSDL già discusso in precedenza.

¹¹Simple Mail Transfer Protocol

¹²File Transfer Protocol

¹³Extensible Messaging and Presence Protocol

-
- *Elencazione dei servizio:* inserendo il Web service creato all'interno di un "registro comune" che permette la ricerca e il reperimento in maniera veloce. Per fare ciò si utilizza il protocollo UDDI¹⁴.

Nella maggioranza dei casi un Web Service viene sviluppato basandosi su un **Web Server** ossia un applicazione software in esecuzione su un Server che si occupa di gestire le richieste HTTP. Il Web Service non si deve così preoccupare di gestire il protocollo HTTP e può essere velocizzato sopperendo alla necessità di gestirne le richieste.

1.5 REST & Restful API

Il **Representational State Transfer** ossia la rappresentazione del trasferimento di stato di una determinata risorsa, abbreviato con **REST**, è stato ideato nel 2000 da Roy Fielding autore già delle specifiche del protocollo HTTP. Il REST è un pattern architetturale che fornisce degli standard per sistemi distribuiti sul web rendendo più semplice la comunicazione fra i diversi componenti.

Una **API REST** detta anche **API RESTful** è un'interfaccia di programmazione che implementa i principi dell'architettura REST elencati di seguito:

1. **Client-Server:** tutte le architetture REST devono implementare il paradigma **Separations of Concerns** ossia la divisione dei compiti fra i vari componenti del sistema. Per fare ciò REST utilizza l'architettura Client-Server; un insieme di interfacce uniformi separano il client dal server e viceversa: il client non deve preoccuparsi di come vengono gestiti i dati e il server non si deve far carico dell'interfaccia grafica. Questo approccio permette lo sviluppo parallelo e indipendente dei componenti fintanto che le interfacce non vengono modificate.
2. **Stateless:** ogni richiesta è trattata come una transazione indipendente e il server non tiene traccia delle interazioni con i vari client. Ciò è possibile poiché ogni richiesta dai client contiene tutte le informazioni necessarie per l'esecuzione del servizio.
3. **Cacheable:** i client possono mettere in cache le risposte (solo se esplicitamente etichettate come *cachable*) dal server in modo da riutilizzarle per richieste successive.

¹⁴Universal Description Discovery and Integration: basato su XML, è un registro che permette la pubblicazione dei propri servizi da parte delle aziende.

-
4. **Layered System**: realizzazione di un sistema a strati dove ogni strato è indipendente, ha la sua funzione e tutti gli strati collaborano per fornire dei servizi. Questo sistema è nascosto al client.
 5. **Uniform Interface**: client e server comunicano utilizzando un’interfaccia uniforme che stabilisce quattro sotto vincoli aggiuntivi che devono essere seguiti per rispettare questo principio:
 - **Identificazione delle risorse**: ogni risorsa presente sul sistema deve essere identificata ed accessibile.
 - **Manipolazione delle risorse attraverso la loro rappresentazione**: si accede alle risorse attraverso la loro rappresentazione e mai direttamente.
 - **Messaggi autodescrittivi**: ogni messaggio scambiato fra le componenti contiene tutte le informazioni necessarie all’interazione.
 - **Hypermedia come motore per la gestione dello stato dell’applicazione**: ogni risorsa deve fornire l’accesso ad eventuali risorse collegate. Il client passa così da risorsa a risorsa attraverso i collegamenti fra di esse forniti dal server. L’unico indirizzo di cui un client REST è a conoscenza sin dall’inizio dell’interazione è l’entry point delle API; quest’ultimo principio elimina del tutto il *versionamento delle API*.
 6. **Code on Demand**: unico vincolo opzionale dell’architettura REST. I server possono trasferire del codice eseguibile come per esempio Applet Java o codice Javascript.

Uno dei principi fondamentali dell’architettura REST è la divisione dello stato dell’applicazione e delle funzionalità in **risorse web**.

Un API RESTful per essere definita come tale, permette di manipolare le risorse attraverso la loro rappresentazione solitamente utilizzando gli standard JSON, HTML, XLT, immagini o altri contenuti ma mai direttamente. Comunicando attraverso un’interfaccia standard come il protocollo HTTP, un’API RESTful consente l’accesso ad una risorsa attraverso il suo identificatore. Ogni risorsa è univocamente identificata da un URL.

Un metodo HTTP associato ad un URL specificare un’operazione da eseguire su una risorsa. Per convenzione, ad ogni operazione **CRUD**¹⁵ viene mappato un metodo HTTP:

¹⁵Acronimo che simboleggia le quattro operazioni fondamentali dei Database: Create, Read, Update e Delete.

- **Create:** per la creazione di una nuova risorsa si usa il metodo HTTP **POST**.
- **Read:** la lettura di una risorsa è associata al metodo HTTP **GET**.
- **Update:** per l'aggiornamento di una risorsa solitamente si utilizza il metodo HTTP **PUT**.
- **Delete:** l'eliminazione di una risorsa è mappata sull'omonimo metodo HTTP **DELETE**.

Nella Tabella di seguito viene esposto un esempio di quanto sopra descritto, utilizzando come "*Base URL*" da anteporre prima di ogni URL il seguente: <https://qrestaurant.certimeter.com/qrestaurant>.

Metodo HTTP	URL	Utilizzo
GET	/client/mobile/attendance	Restituisce l'intero elenco di risorse di tipo <i>attendance</i> e altre informazioni aggiuntive su di esse.
GET	/client/mobile/attendance/1	Restituisce la risorsa di tipo <i>attendance</i> identificata dal numero uno e altre informazioni aggiuntive su di essa.
POST	/manager/mobile/registration	Crea un nuovo <i>utente</i> la cui rappresentazione è contenuta nel corpo della richiesta e lo aggiunge all'elenco di <i>utenti</i> .
PUT	/manager/mobile/society	Rimpiazza la risorsa di tipo <i>society</i> con una nuova risorsa specificata nel corpo della richiesta.
DELETE	/manager/mobile/notification	Elimina l'intero elenco di risorse di tipo <i>notification</i> .
DELETE	/manager/mobile/notification/1	Elimina la risorsa di tipo <i>notification</i> identificata dal numero uno.

Tabella 1.1
Esempio di utilizzo dei metodi HTTP.

Come osservato nella Tabella 1.1, l'impiego del metodo GET non modifica in nessun modo la collezione ma si limita a recuperare le risorse o la risorsa richiesta e restituirla al chiamante. Questi tipi di metodi sono detti **nullipotenti** perché non producono nessun effetto collaterale: la collezione rimane la stessa prima e dopo la loro invocazione.

I metodi PUT e DELETE vengono invece detti **idempotenti**: lo stato del sistema non cambia nonostante il numero di volte che la loro invocazione viene ripetuta.

In conclusione, sebbene un API RESTful debba essere conforme a non pochi principi, il suo utilizzo è considerato più rapido e leggero. Un'architettura REST è quindi un'ottima candidata per l'IoT¹⁶ e per le applicazioni mobile rispetto a protocolli come SOAP¹⁷ che richiedono l'implementazione di requisiti specifici in termini di messaggistica, sicurezza e di transazioni e sono di conseguenza più adatti a sistemi aziendali complessi.

¹⁶Internet of Things

¹⁷Simple Object Access Protocol, è un framework per lo scambio di messaggi fra componenti software.

Capitolo 2

Tecnologie

In questo capitolo vengono delineate le maggiori tecnologie utilizzate nella gestione del progetto distinguendo fra tecnologie front-end e back-end, tecnologie utilizzate per la gestione dei dati, per il deployment e per il versionamento del codice.

2.1 Front End

Per quanto riguarda il front-end, nel progetto è stato fatto uso di React, una libreria JavaScript basata su HTML5 creata nel 2013 da Facebook e da una comunità di singoli sviluppatori e aziende. In questo sotto paragrafo verranno introdotte le tecnologie impiegate per realizzare l’interfaccia web del progetto utilizzando React.

2.1.1 HTML

L’**HyperText Markup Language** abbreviato con **HTML** è un *linguaggio di markup*. Venne sviluppato da Tim Berners-Lee al CERN di Ginevra nel 1989 ma fino al 1991 venne utilizzato solo esclusivamente all’interno dell’Organizzazione. Con *HyperText* si fa riferimento ai link che collegano le pagine web fra di loro, aspetto fondamentale del World Wide Web.

Le prime specifiche di HTML vennero rese ufficialmente pubbliche nel 1993 sostenute dall’IETF. Un anno dopo, frutto della collaborazione fra CERN e MIT, nacque il **World Wide Web Consortium (W3C)** e da allora lo sviluppo di HTML è diventato prerogativa del consorzio.

Negli anni il W3C si occupò di revisionare le specifiche di HTML e di rendere disponibili nuove versioni del linguaggio: nel 1995 venne definita la versione 3.0 a cui seguì la 3.2 nel 1997 fino ad arrivare ad HTML4 nel 1998. Al giorno d’oggi, viene utilizzato HTML5¹, reso ufficiale il 28 ottobre 2014. Alcuni dei motivi principali

¹ultimo aggiornamento 15 febbraio 2021.

che portano alla nascita di HTML5 furono la necessità di creare funzionalità direttamente fruibili e di garantire una compatibilità fra i diversi browser.

L'**HTML** è un *linguaggio di formattazione* ossia descrive il *layout* (impaginazione e visualizzazione grafica) di una pagina web che successivamente un browser si occuperà di renderizzare. Per creare il layout di una pagina, HTML utilizza dei *tag* detti *tag di formattazione*. Inoltre supporta script e oggetti esterni come immagini e filmati. Il suo unico scopo è quello di gestire i contenuti di una pagina specificandone una struttura grafica. Tutte le direttive sono contenute in un **documento** o **pagina HTML** ossia un file di testo che in genere ha estensione *.html*.

Le pagine HTML vengono immagazzinate su server costantemente connessi ad Internet in modo che, attraverso l'utilizzo di un Web Server che se ne occupa, possano essere prodotti ed inviati ai vari browser che ne fanno richiesta tramite il protocollo HTTP. Distinguiamo fra due tipi di pagine:

- **Pagine Web Dinamiche** generate lato server tramite l'utilizzo di altri applicativi presenti sul server che vengono poi inviate ai browser. Il contenuto di queste pagine cambia a seconda dell'input dell'utente che può interagire con esse;
- **Pagine Web Statiche** che non necessitano di computazioni lato server e il cui contenuto rimane invariato a meno che non venga modificato il codice sorgente.

HTML è composto da una serie di elementi racchiusi in un componente detto **tag** ossia una marcatura che stabilisce come la struttura di un elemento debba essere visualizzata e formattata. I tag sono racchiusi fra parentesi caporali; di seguito ne viene presentato un esempio.

```
1 <h1> Titolo </h1>
```

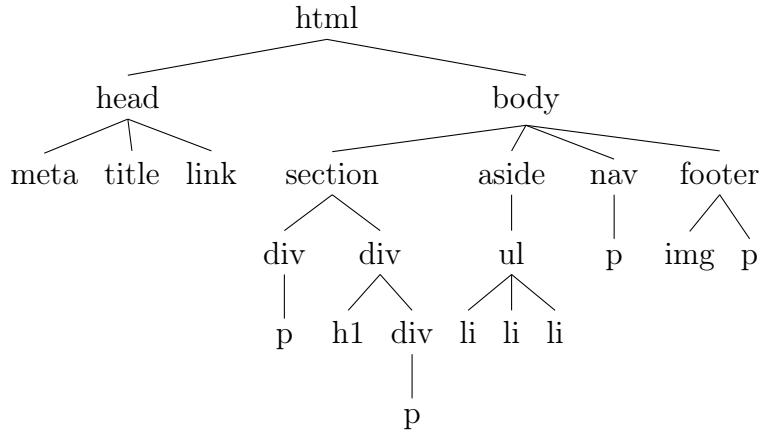
La maggior parte dei tag sono composti da *tag di apertura* e *tag di chiusura*. Il tag di apertura **<h1>** e il tag di chiusura **</h1>** sono identici, ma il tag di chiusura contiene la barra / subito dopo la parentesi angolare aperta. Alcuni tag non hanno il tag di chiusura: è il caso di **** che viene utilizzato per inserire un'immagine nella pagina. In questo caso si parla di *tag a chiusura implicita*.

Un documento HTML presenta una *struttura ad albero annidato* composta da sezioni delimitate da tag che a loro volta contengono sottosezioni delimitate da tag. All'inizio del documento vi è la dichiarazione dello standard utilizzato: per esempio, se il documento è scritto seguendo le specifiche della versione 5, utilizzeremo il tag

<!DOCTYPE html>.

La radice dell'albero è rappresentata dal tag <html> i cui tag di apertura e chiusura racchiudono la struttura esterna dell'albero. Ogni altro contenuto è situato all'interno di questi due tag. All'interno dei tag <html> è prevista dallo standard una ulteriore divisione in due sezioni ben distinte, ognuna con un ruolo specifico:

- **Sezione dello header:** delimitata da <head> e </head>. Contiene informazioni di controllo come:
 - Metadata che contengono informazioni utili per visualizzare la pagina utilizzando il tag <meta>;
 - Collegamenti verso file esterni (per esempio fogli stile esterni) con il tag <link>;
 - Informazioni di stile locali racchiuse all'interno dei tag;
 - Script mediante l'apposito tag <script>;
 - Il titolo della pagina web racchiuso fra i tag <title> e </title>.
- **Sezione del corpo:** delimitata da <body> e </body>, contiene il contenuto che verrà poi visualizzato nella pagina del browser ossia il testo, le immagini e i collegamenti. All'interno di questa sezione possono essere inseriti:
 - Elenchi e liste con i tag che definisce una lista non ordinata e il tag per le liste ordinate;
 - Intestazioni attraverso l'utilizzo dei tag <h1>, <h2>, <h3>, <h4>, <h5> e <h6>;
 - Collegamenti ipertestuali attraverso il tag ;
 - Moduli Elettronici utilizzando il tag <form>;
 - Immagini;
 - Contenuti multimediali e interattivi con i tag <video> e <audio>;
 - Strutture e aspetti del testo come <article>, <footer>, <nav> e <aside>;
 - Layout generico.



*Figura 2.1
Esempio di struttura ad albero di una pagina HTML.*

Ogni tag contiene inoltre degli **attributi** che delineano ulteriormente la funzione di un elemento. Gli attributi sono coppie "chiave=valore" e vengono scritti all'interno del tag stesso. Nell'esempio seguente viene mostrato l'utilizzo di un attributo relativo ad un tag HTML.

```
1      <h1 id="title"> Titolo </h1>
```

In questo caso, l'attributo *id* rappresenta l'identificatore di quel particolare tag. Ci si potrà riferire a quel tag, per esempio per modificarne la formattazione, utilizzando la keyword *title*. Altri attributi di base importanti e largamente utilizzati sono: *class* e *style* che si riferiscono rispettivamente alla classe di un tag e allo stile con cui deve essere visualizzato nella pagina. Esistono inoltre attributi che si occupano della gestione degli eventi come: *onClick*, *onLoad*, *onScroll* e *onDrag* che stabiliscono il comportamento della pagina in relazione all'evento causato dall'utente.

Lo standard non pone ulteriori regole precise a cui attenersi, tranne il seguire i giusti annidamenti: sezioni e sottosezioni non si devono sovrapporre.

2.1.2 CSS

HTML nacque come un linguaggio strutturale ossia come linguaggio che definisse solamente la struttura di una pagina. Lo strumento pensato e creato per arricchire l'aspetto e la presentazione di una pagina è **CSS**.

Il **Cascade Style Sheet** abbreviato con **CSS** è un linguaggio di stile per le pagine web. Nacque in contemporanea con HTML con lo scopo di istruire i browser su come un documento debba essere visualizzato in termini di font, colori, immagini, layout, posizionamento degli elementi, etc. Un foglio stile è un foglio di testo che presenta un'estensione *.css*.

La separazione fra contenuto e layout si è resa necessaria per permettere una programmazione più chiara e facile da utilizzare e allo stesso tempo un maggior riuso del codice.

Un foglio stile CSS è formato da una serie di regole composte da un *selettore* e un *blocco delle dichiarazioni*. Un selettore è un predicato che si riferisce ad un elemento del documento HTML mentre il blocco delle dichiarazioni è un insieme formato da proprietà assegnate al selettore e separate dal punto e virgola. Una proprietà è a sua volta composta da un'altra proprietà e da un valore divisi dai due punti.

```
1      selettore {  
2          proprietà1: valore1;  
3          proprietà2: valore2;  
4      }
```

I **selettori** si distinguono a seconda della tipologia:

- **Selettori di tipo** che si applicano a tutti i tag del tipo di elemento espresso dal selettore. Ad esempio la regola CSS di seguito verrà applicata a tutti i tag **<h1>**:

```
1      h1 {  
2          proprietà: valore;  
3      }
```

- **Selettori di classe** che si applicano a tutti i tag che contengono come attributo la classe individuata dal selettore.

Definiamo per esempio una regola CSS che si applica a tutti gli attributi che hanno come valore della classe *value*. Per realizzare questa regola il selettore dovrà essere preceduto dal punto.

```
1      .value {  
2          proprietà: valore;  
3      }
```

La proprietà verrà quindi attribuita a tutti i tag il cui valore della classe è *value*;

- **Selettori d'identificatore** che vengono applicati al tag il cui valore dell'identificatore è lo stesso di quello espresso dal selettore. In questo caso il selettore dovrà far precedere il cancelletto al nome dell'identificatore.

```
1      #value {
2          proprietà: valore;
3      }
```

Nell'ultimo esempio presentato, la proprietà viene applicata a tutti i tag con identificatore *value*;

- **Selettori di pseudo classe** sono selettori che identificano gli elementi in base alle proprietà. Si applicano in concomitanza con selettori di tipo, di classe e d'identificatore, e hanno forma:

```
1      selettore: selettore_pseudo_classe {
2          proprietà: valore;
3      }
```

Fra i selettori più usati di pseudo classe si annoverano: *link* e *visited* che si applicano rispettivamente ai link non visitati e a quelli visitati; *active*, *focus* e *hover* che si applicano agli elementi attivi, selezionati e a quelli sui quali è posizionato il cursore del mouse.

- **Selettori di pseudo-elementi** si riferiscono a selettori che identificano solo parte degli elementi. Consideriamo *first-line* che si applica agli elementi della prima linea di un paragrafo o *first-letter* per il primo carattere di un elemento.
- **Selettori di gerarchia** identificano gli elementi che si trovano in una particolare relazione all'interno dell'albero del documento HTML.

- **Selettore di discendenza** identifica gli elementi contenuti in altri elementi. Nell'esempio che segue, la proprietà viene applicata a tutti i *selettore2* che sono contenuti nel *selettore1*;

```
1      selettore1 selettore2 {
2          proprietà: valore;
3      }
```

- **Selettore figlio** identifica gli elementi che sono figli di altri elementi. Di seguito un esempio di selettore figlio dove la proprietà viene assegnata a tutti i *selettore2* figli di *selettore1*;

```
1      selettore1 > selettore2 {
2          proprietà: valore;
3      }
```

-
- **Selettore fratello** identifica il primo elemento immediatamente successivo ad un altro, entrambi contenuti nello stesso elemento padre, come nell'esempio:

```
1      selettore1 + selettore2 {  
2          proprietà: valore;  
3      }
```

Nell'esempio precedente, la proprietà viene assegnata al primo *selettore2* fratello di *selettore1*.

- **Selettore di attributo** permette di identificare elementi in base ai loro attributi. Sono della forma:

```
1      selettore[attr=val] {  
2          proprietà: valore;  
3      }
```

oppure semplicemente:

```
1      selettore[attr] {  
2          proprietà: valore2;  
3      }
```

Nel primo esempio, la proprietà viene assegnata a tutti gli elementi che presentano un attributo *attr* che ha valore *val*. Nel secondo esempio, la proprietà viene assegnata a tutti gli elementi che hanno attributo *attr*, a prescindere dal valore che quest'ultimo può assumere.

Le **proprietà** CSS che possono essere scritte all'interno del blocco delle dichiarazioni sono numerose. Di seguito si riportano le più utilizzate:

- **Width** ed **height** utilizzate per impostare rispettivamente la larghezza e l'altezza di un elemento;
- **Background** per impostare lo sfondo di un elemento;
- **Border** definisce il bordo di un elemento;
- **Color** definisce il colore del testo di un elemento;
- **Position** e **Float** che determinano rispettivamente la posizione di un elemento e la disposizione di altri elementi ai suoi lati;

-
- **Margin** e **Padding** specificano lo spazio circostante agli elementi. Margin definisce lo spazio esterno all'elemento mentre padding ne definisce lo spazio interno;
 - **Text-align** e **font-family** il primo stabilisce l'allineamento degli elementi e del testo, il secondo le proprietà del carattere.

Se per un elemento non viene specificata la proprietà, il browser le assegna un **valore** predefinito. Altrimenti, si può scegliere di assegnare alla proprietà uno fra i seguenti valori:

- **Inherit** che specifica che la proprietà deve essere ereditata dall'elemento padre;
- **Auto;**
- Un **numero** con unità di misura;
- Una **percentuale;**
- Un **colore** indicato con diversi sistemi;
- Un **URI** racchiuso tra le parentesi della forma url();
- Un **font;**
- Un **valore tipico della proprietà.**

CSS presenta numerosi vantaggi:

- Le istruzioni CSS possono essere raccolte in un file esterno detto *foglio di stile esterno* memorizzato nella cache dei browser. I server dovranno così trasmettere meno dati;
- Rendono il codice HTML semplice e leggibile delegando tutto ciò che riguarda la presentazione della pagina alle direttive CSS;
- Possibilità di creare fogli di stile separati per i dispositivi mobili.

L'inserimento del codice CSS all'interno di un documento HTML può essere effettuato attraverso tre modalità distinte:

1. Inserendo nella sezione dello header un collegamento ad un foglio stile esterno.
Il collegamento può essere a sua volta effettuato con due modi diversi:
 - (a) Tramite il tag <link> come nell'esempio sottostante:
 - (b) Utilizzando la direttiva import che riporta l'url del foglio.

Questa soluzione rende il codice CSS ampiamente riutilizzabile.

2. Inserendo le dichiarazioni di CSS all'interno di appositi tag <style> nella sezione dello header. Il codice scritto seguendo questa modalità può essere utilizzato solo nella pagina in cui è dichiarato.
3. All'interno dei tag HTML.

Le modalità 2 e 3 non consentono il riuso del codice .

Una funzione utilissima di CSS, già menzionata in precedenza, è quella di distinguere fra i diversi tipi di dispositivi che dovranno interpretare le regole di un foglio di stile. Un caso comune potrebbe essere quello in cui si sceglie di visualizzare la pagina in maniera differente a seconda dell'utilizzo, per esempio se la pagina deve essere visualizzata dal browser o se deve essere stampata.

E' possibile avvalersi di due modalità per realizzare questa divisione:

1. Attraverso la sintassi **media="mediatype"** nel tag <link> con il quale si importa il foglio di stile esterno.
2. Nel foglio di stile stesso usando la regola **@media** che presenta la sintassi:

```
1      @media not|only mediatype{  
2          proprietà: valore;  
3      }
```

Il valore **mediatype** rappresenta il tipo di browser a cui è destinato il foglio di stile da applicare alla pagina web. I valori più supportati sono:

- **Screen** per laptop e desktop;
- **Handheld** per i palmari e per gli smartphone;
- **Print** per le stampanti;
- **All** per tutti i dispositivi.

Uno dei principi per cui CSS impone direttive specifiche è l'ordine in cui applicare gli stili. Fogli di stile diversi associati allo stesso elemento di una pagina web possono infatti entrare in conflitto.

In questi casi è il browser, seguendo gerarchie prestabilite, che decide quale prendere in considerazione e quale no. L'insieme delle gerarchie è detto **Cascata** dal nome *Cascade Style Sheet* ed è basato su un sistema di importanza. Le regole con più alta importanza sono quelle marcate con la dicitura *!important* mentre quelle con

più bassa priorità sono quelle di default. La gerarchia è così organizzata (dal più importante al meno importante):

1. Regole marcate come importanti.
2. Regole Inline Style ossia scritte direttamente nel tag HTML.
3. Regole contenenti selettori d'identificazione.
4. Regole contenenti selettori di classe, di attributi e pseudo-classi.
5. Regole contenenti elementi e pseudo-elementi.
6. Regole ereditate.
7. Valori di default.

L'ordine delle regole va dalle più specifiche alle meno specifiche perciò è chiamato **CSS specificity**.

Ogni selettori ha inoltre un **peso** che, in caso di regole che utilizzano più selettori, ne determina la priorità. Il peso di ogni regola può essere misurato nel seguente modo: partendo da zero, si aggiunge 1000 per ogni attributo di style, 100 per ogni identificatore, 10 per ogni attributo, classe o pseudo-classe e 1 per ogni elemento o pseudo-elemento. La regola

```
1 body #identificatore .classe div:hover
```

ha peso 122, frutto della somma del peso di (in ordine) un elemento, un identificatore, una classe, un elemento e una pseudo-classe.

Attualmente, la quasi totalità dei browser supporta CSS. Tuttavia nessun browser ne supporta completamente tutte le funzionalità. Inoltre, browser diversi supportano prefissi di proprietà diversi che servono per testare nuove funzionalità non ancora finalizzate.

2.1.3 Javascript

Javascript è un linguaggio di scripting interpretato orientato debolmente agli oggetti e agli eventi, maggiormente utilizzato nella programmazione web lato client ma ugualmente spendibile lato server, in applicazioni desktop e mobile. Fu originariamente sviluppato da Brendan Eich presso la Netscape Communications nel 1995 con il nome di *Mochan* e successivamente ribattezzato come *LiveScript* ed infine come *JavaScript*. Nonostante il nome possa essere fuorviante, JavaScript, essendo un linguaggio lambda, ossia un linguaggio con espressioni anonime, ha molto di più

in comune con Lisp e Scheme che con Java.

È uno dei linguaggi di programmazione più usati al mondo, con numerose librerie e framework disponibili che semplificano la programmazione sul browser e lato server. Fra le librerie più utilizzate citiamo *JQuery*, *MooTools* e *Prototype*.

Javascript è un importante supporto ad HTML e la sua importanza è cresciuta con l'avvento di *HTML5*. Questa enorme diffusione è dovuta al fatto che Javascript permette alle pagine HTML di essere modificate dinamicamente in relazione alle azioni dell'utente, senza coinvolgere il server. Il codice viene eseguito lato client evitando il sovraccarico del Web Server.

Il duo HTML-JavaScript prese il nome di **Dynamic HTML** o **DHTML**.

L'esigenza di rendere il web sempre più interattivo portò alla nascita di numerose tecnologie fra cui Flash, sviluppata da Macromedia e acquistata in seguito da Adobe. Questo rinnovato interesse, e la concorrenza, portarono alla cosiddetta *Guerra dei browser* iniziata fra Microsoft e Netscape e conclusa con il ritorno di JavaScript all'apice grazie all'avvento di **AJAX**, tecnologia che permette la comunicazione asincrona con il server consentendo l'aggiornamento dei dati senza dover ricaricare la pagina.

Esistono essenzialmente tre modi per incorporare il codice JavaScript all'interno di una pagina HTML:

1. **Codice Inline:** consiste nell'incorporare le istruzioni JavaScript nel codice di un elemento HTML, assegnandolo ad attributi che si occupano della gestione degli eventi.
2. **Utilizzando il tag <script>:** si tratta di blocchi di codice JavaScript racchiusi all'interno del tag `<script>` in una pagina HTML.
3. **File esterno:** il codice JavaScript viene scritto in un file esterno, con estensione `.js`, che viene poi collegato alla pagina HTML utilizzando il tag `<script>` all'interno della sezione dello header.

JavaScript è un linguaggio debolmente tipizzato ossia effettua il controllo del tipo della variabile a runtime. Nella tipizzazione dinamica il tipo delle variabili non è specificato a priori e può cambiare dinamicamente nel corso del programma. Per questo i linguaggi debolmente tipizzati sono anche interpretati. L'interprete, infatti, è in grado di assecondare tutti i cambiamenti di tipo delle variabili. I browser moderni incorporano un interprete JavaScript che interpreta il codice JavaScript contenuto nelle pagine web. L'API che consente la manipolazione delle

pagine HTML da parte di JavaScript è il **Document Object Model (DOM)**. Il DOM definisce la struttura logica del documento e come questa struttura può essere modificata.

Javascript non prevede la dichiarazione obbligatoria delle variabili ma per maggiore chiarezza sintattica è possibile utilizzare la keyword *var* che permette di istanziare variabili globali a livello di programma o di funzione come nell'esempio seguente:

```
1     var nomeVariabile = "Variabile"
```

JavaScript permette inoltre di istanziare variabili la cui visibilità è limitata ad un blocco di codice utilizzando le keyword *let* e *const* con l'unica differenza che la prima può essere riassegnata mentre la seconda no.

In JavaScript i tipi primitivi sono i numeri, le stringhe, i booleani, *null* e *undefined*. Tutto il resto è un oggetto.

Un oggetto è un contenitore modificabile di proprietà, ognuna con un nome ed un valore. In questo linguaggio non esistono le classi e non vi sono regole sul nome delle proprietà e sui loro valori. Gli oggetti possono contenere qualsiasi cosa, anche altri oggetti. In questo modo si possono facilmente rappresentare strutture ad albero o grafi.

Ogni oggetto è collegato ad un **prototipo** ossia ad un altro oggetto di cui eredita le proprietà. Nella creazione di oggetti si può partire da una struttura comune creata tramite un costruttore e andare a modificarne le proprietà in seguito, secondo le esigenze, utilizzando il suo **prototype**.

```
1 // costruttore
2 function Obj(attr1, attr2){
3     this.attr1 = attr1
4     this.attr2 = attr2
5 }
6 //aggiunta di proprietà all'oggetto tramite il prototype
7 Obj.prototype.attr3 = "attr3"
```

Anche le funzioni sono oggetti in JavaScript. Ogni oggetto è legato ad *Object.prototype* ma le funzioni in particolare sono collegate a *Function.prototype* il cui valore è un oggetto con una proprietà costruttore il cui valore è la funzione stessa e che a sua volta è legato ad *Object.prototype*.

In conclusione, JavaScript è un linguaggio leggero ed espressivo. Inoltre il web è diventato una piattaforma importante per lo sviluppo di applicazioni e JavaScript è l'unico linguaggio che si trova in tutti i browser[6].

2.1.4 React

Introduzione

React è una libreria JavaScript open-source per il front-end creata da Jordan Walker nel 2013 e mantenuta da Facebook e Instagram in collaborazione con singole community di sviluppatori.

React è stato pensato per la creazione di interfacce utente. Come erroneamente si potrebbe pensare, React non è un framework ma una libreria. Un framework, per essere definito tale, dovrebbe includere tutto ciò che è necessario alla costruzione di un'applicazione. React invece è semplicemente una libreria che si occupa del livello della vista e che si avvale di altre librerie JavaScript per tutto ciò che non fornisce. La libreria non impone alcuna restrizione sull'architettura dell'applicazione. Perciò, lo sviluppatore è libero di affiancare il pattern MVC all'uso di React se preferisce. React si integra bene con molte altre librerie, framework e pattern architetturali utili alla creazione di applicazioni semplici e scalabili come: **Babel**, **React Bootstrap**, **React Router** e **Axios** utilizzate nel progetto oggetto di questa tesi.

React è una libreria giovane ed in continua evoluzione. JavaScript, su cui React si basa, sin dall'anno del suo rilascio, il 1995, ha subito notevoli cambiamenti e migliorie. Oggi JavaScript è utilizzato per costruire applicazioni full-stack.

Tutti i cambiamenti al linguaggio sono suggeriti e supportati da una community di sviluppatori. Chiunque può sottoporre una proposta di cambiamento all'ECMA che selezionerà le proposte più valide e le incorporerà nella nuova versione delle specifiche. Ad oggi, l'ultima versione rilasciata è la **EcmaScript 2019 o ES2019**².

Per quanto riguarda le variabili, già introdotte nel paragrafo precedente, ES2019 non ha apportato cambiamenti. A partire da ES2016, sono stati introdotti i **parametri di default**. Nell'implementare una funzione, lo sviluppatore può scegliere se assegnare o meno valori di default ai parametri in modo che, se alla funzione vengono passati meno argomenti di quelli che si aspetta, verranno usati quelli di default.

```
1 function f(par1 = "p1", par2 = "p2"){
2     console.log("valori:", par1, par2)
```

²ultimo aggiornamento 15 febbraio 2021.

```
3 }
```

Nell'esempio precedente, i parametri *par1* e *par2* hanno rispettivamente come valori di default *p1* e *p2*. Nel caso in cui la funzione *f* venisse invocata senza argomenti, verrà stampato "valori: *p1 p2*".

Un'altra importante novità introdotta a partire da ES2016 è l'uso delle **funzioni a freccia** che permettono di creare funzioni senza usare la keyword *function*. Nell'esempio seguente, trasformiamo la funzione *f* scritta in precedenza in una funzione a freccia.

```
1 const f = (par1, par2) => `valori: ${par1} ${par2}`
2 console.log(f("p1", "p2"))
```

Per quanto riguarda oggetti e array, da ES6³ in poi è possibile la *scomposizione* degli oggetti per poter utilizzare i valori dei campi separatamente come nel seguente esempio.

```
1 var person = {
2   name: "some name",
3   surname: "some surname",
4   age: 22
5 }
6 var {name, age} = person
```

L'eventuale stampa delle variabili *name* ed *age* produrrebbe come risultato rispettivamente "some name" e 22.

Lo **spread operator** ci consente di eseguire molte funzioni diverse. I tre punti infatti, ci danno la possibilità di combinare insieme il contenuto di due array per esempio.

```
1 var smith = ["Jon", "Smith"]
2 var doe = ["Susanne", "Doe"]
3 var combined = [...smith, ...doe]
4 console.log(combined) //Jon, Susanne, Smith, Doe
```

L'uso di questo operatore può tornare utile anche in altri casi. Pensiamo ad un esercizio in cui viene richiesto il salvataggio dell'ultimo elemento di un array in una variabile. La funzione *reverse* è utile in questo caso ma inverte l'array in modo

³ES2016

permanente. L'uso dell'operatore di spread invece permette di non modificare la forma originale dell'array ma di ottenere comunque il risultato desiderato.

```
1 var [last] = smith.reverse() //modifica l'array in modo  
    permanente  
2 var [last] = [...smith].reverse() //lascia l'array originale  
    intatto
```

Interfacce utente con React

Per poter lavorare con React nel browser, sono assolutamente necessarie due particolari librerie: **React** e **ReactDOM**. React si occupa di creare le viste mentre ReactDOM ha il compito di renderizzare l'interfaccia nel browser.

React ha inoltre bisogno di un elemento HTML da cui partire per renderizzare l'interfaccia. Solitamente viene definito un elemento all'interno di un documento HTML a cui viene assegnato come id una particolare keyword che React utilizzerà poi come punto di partenza per renderizzare la UI.

Il DOM Virtuale è un oggetto JavaScript che fornisce direttive a React su come costruire l'interfaccia nel browser. Mentre HTML non è altro che una serie di istruzioni gerarchiche che un browser segue quando costruisce un Document Object Model o DOM e che tutte le DOM API devono seguire, React non si interfaccia direttamente con le DOM API ma definisce un DOM Virtuale utilizzando i **React Elements** invece degli elementi HTML.

Un elemento React è una descrizione di come il DOM dovrebbe essere, estremamente scalabile in quanto un elemento può essere riutilizzato infinite volte cambiando i dati all'occorrenza. Sono istruzioni su come il DOM del browser dovrebbe essere creato.

Il modo più semplice per creare un elemento è utilizzare: *React.createElement*, una funzione che ha come parametri tre elementi:

1. Il tipo di elemento che si desidera creare.
2. Le proprietà dell'elemento.
3. I figli dell'elemento ossia qualunque nodo che si desideri inserire fra i tag di apertura e chiusura.

Un'espressione del tipo:

```
1 React.createElement(
```

```
2     "div",
3     {id: "react-element", class: "div"},
4     "Prova elemento")
```

verrà renderizzata come:

```
1 <div id="react-element" class="div"> Prova elemento </div>
```

La libreria **ReactDOM** contiene tutti gli strumenti necessari per renderizzare elementi React nel browser. Inoltre la renderizzazione non richiede un tempo significativo. ReactDOM infatti, per ogni cambiamento, non ricostruisce il DOM da zero ma applica solo le modifiche necessarie per trasformarlo. Possiamo renderizzare elementi React nel DOM utilizzando la funzione *render* di ReactDOM. Pensiamo alla funzione che ha come primo argomento l'elemento da renderizzare e come secondo argomento il nodo target del documento HTML in cui renderizzare l'elemento. Immaginiamo di avere un documento html e che il corpo del documento contenga un tag *<div>* con *id=react-container*: Il seguente frammento di codice:

```
1 var elem = React.createElement("p", null, "paragraph")
2 ReactDOM.render(elem, document.getElementById('react-container'))
```

verrà renderizzato come:

```
1 <body>
2     <div id="react-container">
3         <p> paragraph </p>
4     </div>
5 </body>
```

ReactDOM permette la renderizzazione di un solo elemento react all'interno del DOM. Tutti gli elementi react sono annidati all'interno del primo elemento react renderizzato da ReactDOM.

La funzione *createElement* di React permette infatti di specificare quanti più elementi figli si vogliono creando di fatto una struttura ad albero annidato dove la radice è rappresentata dall'elemento renderizzato dalla funzione *render* di ReactDOM. Immaginiamo di voler costruire una lista ordinata di elementi all'interno del corpo di un documento HTML:

```
1 <body>
2     <div id="container">
3         <ol>
```

```
4          <li> Primo elemento </li>
5          <li> Secondo elemento </li>
6          <li> Terzo elemento </li>
7      </ol>
8  </div>
9 </body>
```

Come prima cosa creiamo gli elementi React annidandoli uno dentro l'altro.

```
1 var listaOrdinata = React.createElement("ol", null,
2   React.createElement("li", null, "Primo elemento"),
3   React.createElement("li", null, "Secondo elemento"),
4   React.createElement("li", null, "Terzo elemento"))
```

Dopodichè, renderizziamo l'elemento radice all'interno di un elemento HTML.

```
1 ReactDOM.render(listaOrdinata,
  document.getElementById('container'))
```

Uno dei maggiori vantaggi nell'uso di React è la possibilità di separare i dati dagli elementi dell'interfaccia. Inoltre, siccome React è basato su JavaScript, possiamo incorporare codice JavaScript all'interno di elementi react. Possiamo per esempio renderizzare una lista di elementi di react nel seguente modo (invece che utilizzare la funzione *createElement* per ogni elemento):

```
1 var elementi = [
2   "Primo elemento",
3   "Secondo elemento",
4   "Terzo elemento"]
5 React.createElement(
6   "ol",
7   null,
8   elementi.map
9     (elem => React.createElement("li", null, elem)))
```

Quando si decide di creare una lista di elementi figli iterando su un array, React richiede che ognuno di essi abbia una chiave in modo da rendere efficienti le operazioni sugli elementi. Il modo corretto di annidare degli elementi react è quindi il seguente:

```
1 var elementi = [
2   "Primo elemento",
3   "Secondo elemento",
```

```
4     "Terzo elemento"]
5 React.createElement(
6     "ol",
7     null,
8     elementi.map(
9         (elem, i) => React.createElement("li", {key: i},
10            elem)
11        ))
```

Un altro modo per poter costruire elementi in React che produce lo stesso risultato dell'esempio appena visto è la funzione *createClass* di React.

```
1 const elem = [
2   "Primo elemento",
3   "Secondo elemento",
4   "Terzo elemento"]
5
6 const listaOrdinata = React.createClass({
7   displayName: "ListaOrdinata",
8   render(){
9     return React.createElement(
10       "ol",
11       null,
12       this.props.elem.map(
13         (elementi, i) =>
14           React.createElement("li", {key: i},
15             elementi)))}
16 })
17 ReactDOM.render(
18   React.createElement(listaOrdinata, {elem}, null),
19   document.getElementById("container"))
```

La funzione **render** si occupa di renderizzare un elemento react. Al suo interno è possibile utilizzare la keyword **this** per riferirsi all'istanza dell'elemento. È possibile inoltre accedere ad elementi esterni passati al momento della creazione di un elemento react con la keyword **this.props**, come mostrato nell'esempio precedente. Le props non sono altro che variabili passate dall'elemento genitore all'elemento figlio che non possono essere modificate da quest'ultimo; possono anche essere usate per permettere al componente figlio di accedere a metodi definiti nel componente genitore. Ogni componente può definire poi elementi dello **state**, modificabili e utilizzabili in tutto il componente.

È anche possibile creare elementi in React a partire da classi astratte di componenti: *React.Component*.

```
1 class ListaOrdinata extends React.Component{  
2     render(){  
3         return React.createElement("ol",  
4             null,  
5             this.props.elem.map(  
6                 (elementi, i) =>  
7                     React.createElement("li", {key: i},  
8                         elementi))  
9             )  
10        }  
11    }
```

In React è possibile utilizzare le funzioni stateless per creare elementi. Queste funzioni sono senza stato, non hanno quindi il riferimento a *this*. Prendono come parametri delle proprietà e ritornano un oggetto del DOM. Sono funzioni semplici che rendono il codice facilmente testabile.

```
1 const ListaOrdinata = ({elem}) =>  
2             React.createElement(  
3                 "ol",  
4                 null,  
5                 elem.map((elementi, i) =>  
6                     React.createElement("li", {key:i},  
7                         elementi)))
```

JSX JSX è un'estensione JavaScript che consente di definire elementi React che appaiono simili ad elementi HTML. Utilizzando quest'estensione è possibile definire elementi react con le stesse funzionalità che avremmo se avessimo utilizzato la funzione *createElement*. Un JSX ha la forma:

```
1 <ListaOrdinata elem={elem}>/>
```

Anche gli elementi JSX accettano proprietà che, nel caso non siano stringhe, devono essere passate fra parentesi graffe. Questo tipo di componenti vengono chiamate **espressioni JavaScript**. Le espressioni JavaScript possono includere qualsiasi oggetto, array o funzione e possono essere annidate. JSX, essendo un elemento JavaScript, può incorporare codice JavaScript al suo interno:

```
1 <ol>
```

```

2     {this.props.elem.map((element, i) =>
3         <li key={i}>{element}</li>
4     )}
5   </ol>

```

Il ciclo di vita dei componenti

Il ciclo di vita dei componenti consiste in funzioni che vengono invocate quando un componente è in fase di *mounting* ("montaggio") ossia in fase di inserimento all'interno dell'albero del DOM o in fase di *aggiornamento*. Queste funzioni sono invocate prima o dopo che i componenti sono renderizzati nell'interfaccia: anche la funzione *render* utilizzata per renderizzare i componenti fa parte del ciclo di vita degli stessi.

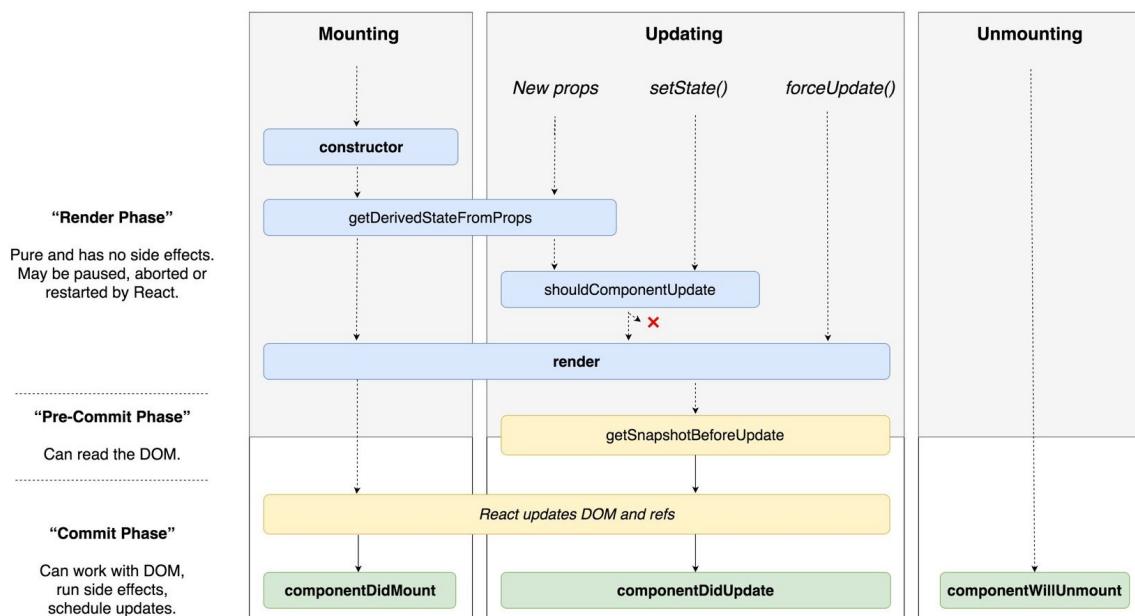


Figura 2.2
Ciclo di vita dei componenti in React

Le funzioni utilizzate nella fase di mounting del componente sono utili per definire chiamate ad API, incorporare codice JavaScript che assista nel processo, manipolare il DOM etc. Vengono chiamate in sequenza:

- **constructor**: indicato principalmente qualora si voglia inizializzare lo stato dei componenti o fare il bind dei metodi. È invocato prima della fase di montaggio del componente;

-
- **getDeliveredStateFromProps**: utilizzato subito prima di invocare la funzione render. Restituisce un oggetto che aggiorna lo stato o null se non ci sono aggiornamenti da effettuare;
 - **render**: unica funzione obbligatoria in ogni componente. Come detto in precedenza, viene utilizzato per renderizzare il componente. È buona pratica che questa funzione non interagisca con il browser e che rimanga *pura* in modo da rendere più semplice la gestione dei componenti;
 - **componentDidMount**: invocato dopo che il componente è stato montato.

Nella fase di unmounting o smontaggio, l'unica funzione chiamata è:

- **componentWillUnmount**: invocato subito prima che un componente venga smontato e distrutto. Questo metodo può essere utilizzato per effettuare operazioni di pulizia come per esempio la cancellazione di timer precedentemente avviati.

Nella fase di aggiornamento vengono chiamati in ordine:

- **getDeliveredStateFromProps**;
- **shouldComponentUpdate**: utilizzato per decidere se l'output di un componente è influenzato dalla modifica dello state o delle props. Questo metodo ha lo scopo di ottimizzare la performance;
- **render**;
- **getSnapshotBeforeUpdate**: invocato prima di ogni cambiamento, permette di catturare elementi del DOM prima che questi vengano modificati da un potenziale aggiornamento;
- **componentDidUpdate**: invocato immediatamente dopo che avviene un aggiornamento del componente. Questo metodo può essere utilizzato per effettuare operazioni sul DOM dopo che quest'ultimo è stato aggiornato.

Babel

Unico lato negativo dell'uso di JSX è fatto che questa estensione non è supportata da nessun browser. A tale scopo, nel 2014 Sebastian McKenzie creò Babel, un compilatore JavaScript che si occupa di convertire JSX in puro React. Questo processo è detto **transpiling**. Il transpiler è un tipo di compilatore che a partire dal codice sorgente di un programma scritto in un determinato linguaggio di programmazione produce un nuovo codice sorgente in un altro linguaggio, sullo stesso livello di astrazione[17]. Tutt'oggi Babel è usato in produzione da aziende come Facebook, Netflix e Paypal.

React Bootstrap

React Bootstrap è una libreria JavaScript che incorpora le funzionalità del noto framework per il front-end Bootstrap nel DOM Virtuale di React.

Si basa su un sistema a griglia che utilizza una serie di contenitori, righe e colonne per definire il layout degli elementi.

Prima riga, prima colonna	Prima riga, seconda colonna	Prima riga, terza colonna	
Seconda riga, prima colonna	Seconda riga, seconda colonna	Seconda riga, terza colonna	Seconda riga, quarta colonna

*Figura 2.3
Esempio di organizzazione a griglia Bootstrap*

I contenitori sono il primo elemento di un sito web creato con React Bootstrap completamente responsive e sono identificate dal tag `<Container>`. All'interno dei contenitori è inserito tutto il contenuto della pagina web per mezzo di righe e colonne. Ogni colonna, indicata con `<Col>` è inserita all'interno di una riga `<Row>`. All'interno di una riga possono coesistere al massimo 12 colonne ognuna con la stessa larghezza e altezza (a meno che non venga specificato diversamente dallo sviluppatore), fluide e dimensionate rispetto al proprio genitore.

Il sistema a griglia di React Bootstrap prevede un layout responsive garantito attraverso la presenza di cinque breakpoint: extra-small, small, medium, large ed extra large. Utilizzando i breakpoint è possibile ridimensionare la grandezza di righe e colonne.

xs=12 md=8	xs=6 md=2	xs=6 md=2
------------	-----------	-----------

*Figura 2.4
Breakpoint a confronto: medium*

xs=12 md=8	
xs=6 md=2	xs=6 md=2

*Figura 2.5
Breakpoint a confronto: small*

Le figure 2.4 e 2.5 sono le stesse: nella prima figura è stato simulato uno schermo con larghezza maggiore di 768px mentre nella seconda uno schermo con larghezza

minore di 768px. Il sistema a griglia responsive di React Bootstrap ci permette di definire una volta le colonne ed assegnargli una grandezza a seconda delle dimensioni dello schermo che le visualizza. Il codice usato è il seguente:

```
1 <Container>
2   <Row>
3     <Col xs={12} md={8}> xs=12 md=8 </Col>
4     <Col xs={6} md={2}> xs=6 md=2 </Col>
5     <Col xs={6} md={2}> xs=6 md=2 </Col>
6   </Row>
7 </Container>
```

Le keyword *xs* ed *md* nell'esempio precedente specificano il breakpoint. All'interno delle parentesi graffe si trova invece la larghezza che le colonne devono avere per quello specifico breakpoint.

React Bootstrap mette a disposizione inoltre una serie di componenti ereditati dal framework Bootstrap e riadattati per React come i form indicati con *<Form>* o le tabelle *<Table>*.

React Router

React viene principalmente utilizzato per creare Single Page Application in cui tutto il codice necessario è caricato in una singola pagina che JavaScript si occupa di modificare dinamicamente. Questo processo rende la pagina web più fluida.

Il processo di **Routing** ossia la definizione di endpoint per le richieste dell'utente, la sincronizzazione della cronologia e della location del browser diventa problematico. A seconda dell'endpoint selezionato, JavaScript si occupa di renderizzare l'interfaccia utente corrispondente. Per questo motivo gli ingegneri Michael Jackson e Ryan Florence hanno creato **React Router** che è presto diventato la soluzione più popolare per gestire il routing nelle app React.

L'elemento base di questa libreria è il **router** individuato dall'elemento React *<Router>* che si occupa di configurare una **route** (*<Route>*) per ogni sezione del sito web. Ogni route è un endpoint che può essere richiesto dall'utente e specificato dall'attributo *path*. Il componente **Switch** (*<Switch>*) ingloba ogni elemento route e semplifica ulteriormente la gestione del routing assicurando almeno la renderizzazione di un elemento ossia il primo che specifica l'endpoint richiesto. È buona pratica definire al fondo della lista dei route un elemento che non specifica l'attributo path: sarà l'elemento renderizzato nel caso in cui l'utente richieda

un endpoint che non è stato gestito. Lo Switch deve essere a sua volta inglobato all'interno dell'elemento router.

```
1 <Router>
2   <Switch>
3     <Route exact path="/home" component={Home}/>
4     <Route exact path="/info" component={Info}/>
5     <Route component={ErrorPage}/>
6   </Switch>
7 </Router>
```

Altri componenti utili della libreria sono: `<Link>` e `<Redirect>` che si occupano rispettivamente di creare link nell'applicazione e di reindirizzare l'utente verso un altro endpoint.

Axios

Axios è una libreria HTTP per i client utilizzata per inviare richieste HTTP e gestirne le risposte. Quando il web era ancora agli albori, la comunicazione fra client-server si basava su un modello sincrono senza possibilità di modifica dinamica delle pagine HTML. Solo in seguito venne introdotto **AJAX**. AJAX è una tecnica di programmazione che si basa su una comunicazione client-server asincrona in cui non vi è bisogno di ricaricare la pagina per aggiornare i dati. Axios fornisce un'API semplice e completa per poter lavorare con le richieste HTTP e supporta i metodi Http GET, DELETE, HEAD, OPTIONS, POST, PUT e PATCH. Un esempio di richiesta axios è presentata di seguito:

```
1 axios.post('/test-post', { data: "data"})
2   .then(function (response) {
3     console.log(response); //callback
4   })
5   .catch(function (error) {
6     console.log(error); //caso di errore
7   });
```

*Listing 2.1
Richiesta HTTP con Axios*

Nell'esempio 2.1 è stata presentata una richiesta HTTP utilizzando il metodo POST. La chiamata Axios accetta come primo parametro l'URL del server al quale inoltrare la richiesta e come secondo parametro i dati, che saranno serializzati automaticamente in un oggetto JSON, da inoltrare.

Se la richiesta è andata a buon fine, il controllo passerà al gestore della risposta. In caso contrario gli errori verranno processati nel gestore delle eccezioni nella *catch*.

Tutte le modifiche a React vengono analizzate e discusse in un repository chiamato Future of React⁴ dove ogni sviluppatore può creare una issue o fare una pull nell'ottica di continuare a migliorare la libreria.

⁴<https://github.com/reactjs/react-future>

2.2 Back End

2.2.1 Maven

Apache Maven è un framework open-source per la gestione di progetti software Java. È basato sull'utilizzo del **Project Object Model (POM)** ossia un file XML che aiuta lo sviluppatore nella gestione delle librerie e delle dipendenze fra di esse.

Una delle caratteristiche principali di Maven riguardanti la gestione delle dipendenze è il meccanismo di download delle stesse all'interno dell'archivio locale dell'utente. L'archivio può essere utilizzato da qualsiasi progetto Maven in modo che le dipendenze non debbano essere scaricate per più di una volta.

Un'altra funzionalità importante di Maven sono i **plug-ins**. Questi forniscono un set di obiettivi che possono essere eseguiti utilizzando i comandi di Maven che hanno la seguente forma:

```
1 mvn [nome-plugin] : [nome-objettivo]
```

Ad ogni comando di Maven è associata una fase: per esempio a *mvn compiler:compile* è associata la fase *compile*. Quindi quando viene eseguito il comando *mvn compile* vengono di conseguenza eseguiti anche tutti gli obiettivi associati.

2.2.2 Spring Framework

Introduzione

Spring nacque nel 2003 come risposta alla complessità delle specifiche J2EE⁵. La prima versione del framework venne scritta da Rod Johnson sotto la licenza Apache 2.0. Da allora è stato un susseguirsi di miglioramenti e rilasci fino alla versione 5.3.3 rilasciata il 12 gennaio 2021⁶. È considerato il framework open-source Java più popolare al mondo⁷.

I principi che guidano lo Spring Framework sono:

1. Garantire la possibilità di scelta allo sviluppatore a qualsiasi livello e prendere decisioni sul design il più tardi possibile.
2. Flessibilità. Spring supporta una vasta tipologia di applicazioni con diversi bisogni e diverse prospettive.

⁵Java 2 Enterprise Edition, standard per lo sviluppo di applicazioni enterprise a più livelli.

⁶ultimo aggiornamento 15 febbraio 2021.

⁷<https://snyk.io/blog/jvm-ecosystem-report-2018-platform-application/>

-
3. Retrocompatibilità. Tutta l'evoluzione di Spring è stata studiata per garantire pochi cambiamenti sostanziali tra le versioni. Spring inoltre supporta un piccolo range di JDK e librerie di terze parti per facilitare il mantenimento delle applicazioni e delle librerie da cui dipende.
 4. Design delle API ben pensato. Ogni APIs è intuitiva e resiste per molti anni anche attraverso numerose versioni.
 5. Alta qualità del codice. Spring è uno dei pochi progetti che può vantare una struttura del codice pulita e nessuna dipendenza circolare tra i package.

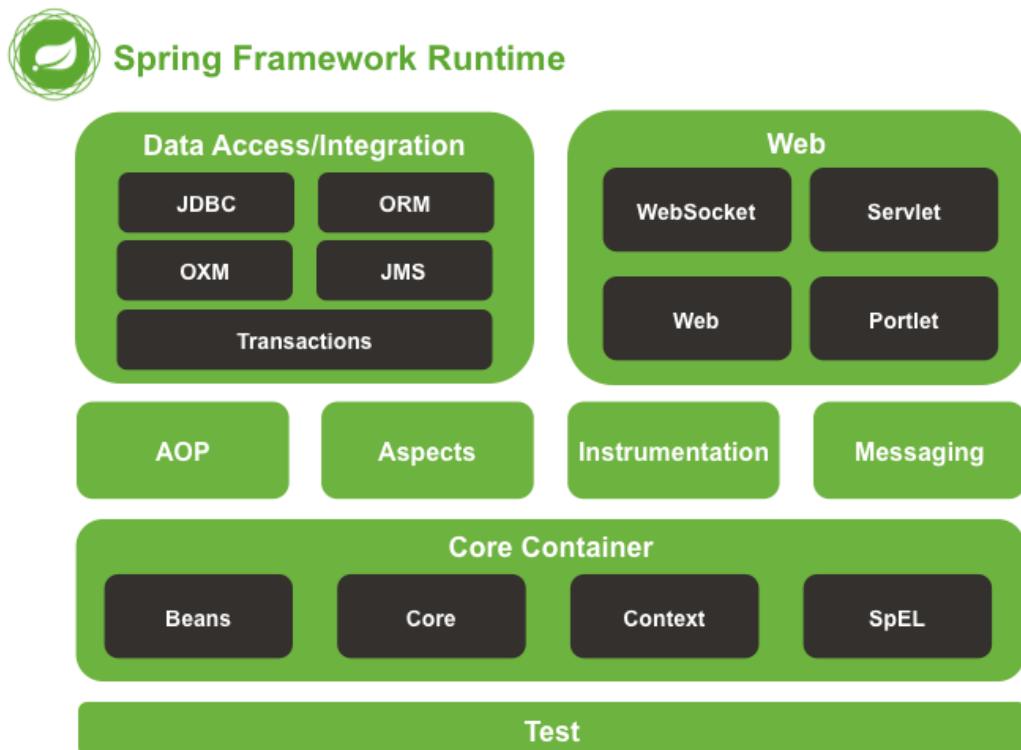
Attorno a Spring ruotano numerosi progetti come **Spring Boot**, **Spring Security**, **Spring Cloud**, **Spring Batch** e **Spring Data**.

Spring Framework include una vasta gamma di servizi:

- **Spring Core Container**: modulo base di Spring che fornisce gli *Spring Container*.
- **Aspect Oriented Programming**: permette l'implementazione di *cross-cutting concerns*⁸;
- **Autenticazione e autorizzazione**: Spring security fornisce la possibilità di configurare dei processi di sicurezza che supportano vari standard e protocolli;
- **Convention over Configuration**: Spring fornisce una soluzione rapida per l'implementazione di applicazioni enterprise;
- **Accesso ai dati**: Spring fornisce gli strumenti necessari per lavorare con database relazionali e non;
- **Inversion of Control (IoC)**: configurazione dei componenti dell'applicazione e del ciclo di vita degli oggetti realizzato attraverso la *dependency injection*;
- **Messaggistica**: Java Message Service permette la configurazione di *Message Listener* per la consumazione di code di messaggi;
- **Model-View-Controller**: Spring MVC come framework per le applicazioni web RESTful basato sul protocollo http e sulle servlet;
- **Remote Access Framework**: consente la configurazione di chiamate di procedura remota attraverso l'utilizzo di *Java remote method invocation (RMI)*, *Common Object Request Broker Architecture (COBRA)* e protocolli basati su HTTP come *Simple Object Access Protocol (SOAP)*;

⁸Requisiti che devono essere soddisfatti al fine di realizzare gli obiettivi dell'intero sistema e la cui implementazione è trasversale a molti moduli dell'applicativo.[1]

- **Gestore delle transazioni:** unificazione delle API per la gestione delle transazioni;
- **Remote Management:** Java Management Extensions (JMX) permette la gestione di oggetti per la configurazione locale o remota;
- **Test:** Spring supporta classi per la scrittura di unit test.



*Figura 2.6
Container di Spring*

Inversion of Control

Lo **IoC** è considerato il cuore di Spring e fornisce uno strumento semplice ed efficace per la gestione delle dipendenze tra componenti chiamati **bean**. Lo IoC Container, utilizzando la *Reflection* di Java, si occupa di gestire tutto il ciclo di vita dei componenti: dalla creazione alla configurazione che può essere realizzata tramite file XML o attraverso le annotazioni.

Il Container è composto da due interfacce:

1. **BeanFactory** che definisce le funzionalità per la registrazione dei bean: creazione, inizializzazione, gestione del ciclo di vita.

-
2. **ApplicationContext** che si occupa di gestire funzionalità più avanzate come la gestione degli eventi.

Il pattern più utilizzato per implementare l'IoC è il **Dependency Injection**. Un oggetto definisce le sue dipendenze senza istanziarle direttamente, in questo modo le componenti vengono *iniettate* dal framework esterno.

```
1 @Service
2 class UserService{
3     ...
4 }
5
6 @RestController
7 class UserController{
8     @Autowired
9     UserService userService;
10    ...
11 }
```

Listing 2.2

Utilizzo della Dependency Injection mediante le annotazioni di Spring

Le annotazioni `@Service` e `@RestController` vengono utilizzate per definire dei tipi di bean. L'annotazione `@Autowired` realizza la dependency injection, iniettando il bean `UserService` all'interno del bean `UserController`.

Spring Boot

Uno dei servizi di spring già presentati in precedenza è la *convention over configuration*. Questo servizio è rappresentato da **Spring Boot**: una soluzione open-source per creare applicazioni spring stand-alone. È preconfigurato con le configurazioni base date dal team di Spring, usa piattaforme di Spring, numerose librerie pre-configurate e può essere modellato a piacimento dallo sviluppatore in modo semplice utilizzando il **Project Object Model (POM)** di Maven. Nel progetto è stato fatto uso di questo framework per la creazione dell'applicativo.

Spring Boot configura vari componenti automaticamente, registrandone i bean necessari. Se per esempio viene aggiunta nel classpath la dipendenza `spring-boot-starter-actuator`, Spring crea automaticamente gli endpoint necessari per il recupero dei dati che contengono informazioni sulla salute dell'applicazione.

Un altro vantaggio del framework è la possibilità di incorporare nell'applicativo web

server e web container come Apache Tomcat o Jetty, dunque non è più necessario l'utilizzo di file WAR.

Creazione di un progetto Spring Boot La creazione di un progetto Spring Boot può essere fatta utilizzando lo *Spring Initializr* all'url <https://start.spring.io>.

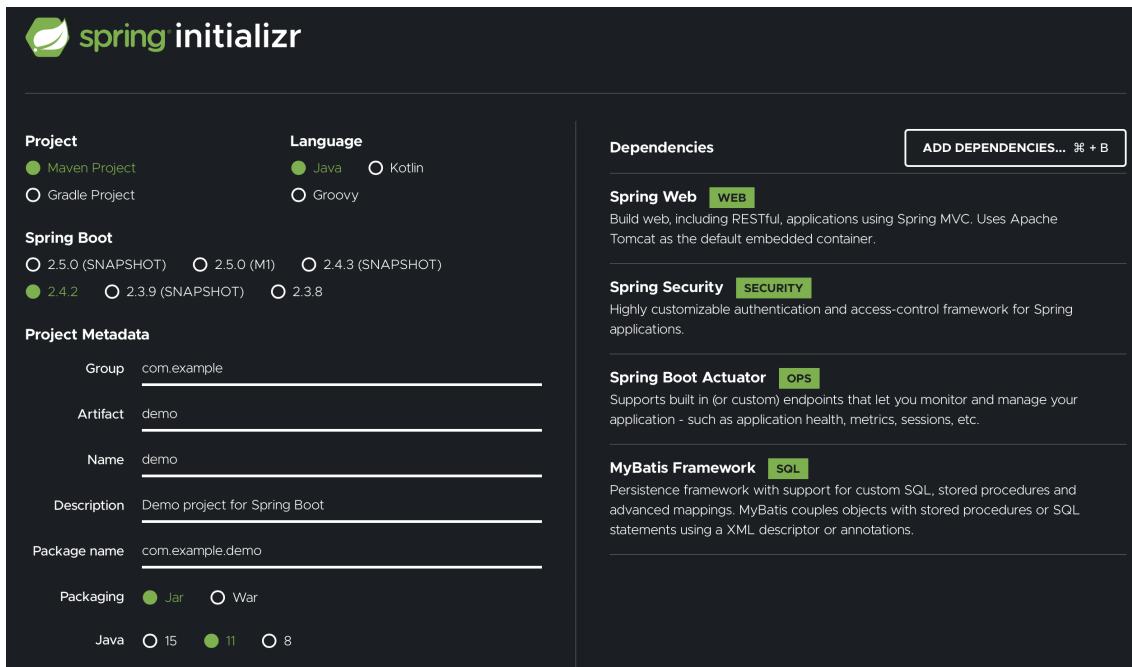


Figura 2.7
Spring Initializr

Questo tool permette di impostare in modo semplice e veloce il tipo di progetto, il linguaggio che si vuole utilizzare, la versione di Spring Boot che si vuole utilizzare, i dati del progetto come il nome e la descrizione, il tipo di packaging, la versione di Java e le eventuali dipendenze. Tutte le configurazioni verranno salvate nel POM rendendone più facile la modifica.

Spring MVC

Spring MVC è un framework basato sul pattern Model-View-Controller che permette di creare applicazioni web che incorporano web server come Tomcat. Attraverso l'utilizzo di questo framework è possibile utilizzare la *Dispatcher Servlet* che costituisce il primo controller che riceve le richieste dai client e le delega agli altri handler o controller che si occuperanno di gestirle e restituire la risposta.

Per designare i controller che dovranno processare le richieste, Spring MVC fornisce un sistema di annotazioni: l'annotazione `@Controller` indica il componente che si occupa di gestire le richieste. A partire da Spring 4.3 sono state introdotte le annotazioni `@GetMapping`, `@PostMapping`, `@PutMapping`, `@DeleteMapping` etc per

specificare quale determinato endpoint il metodo di un controller mappa. Inoltre l'annotazione `@ResponseBody` lega il valore di ritorno del metodo al corpo della risposta.

```
1 @Controller
2 public class ManagerController{
3
4     @ResponseBody
5     @GetMapping("\manager")
6     public ResponseEntity<Response> getManager(){ ... }
7
8     @ResponseBody
9     @DeleteMapping("\manager")
10    public ResponseEntity<Response> deleteManager(){ ... }
11 }
```

Listing 2.3

Utilizzo delle annotazioni spring per definire controller delle richieste

Spring MVC fornisce supporto nell'implementazione di servizi web RESTful come la definizione di un'annotazione ad hoc per i controller Rest: `@RestController`. Questa annotazione di permette di evitare di scrivere l'annotazione `@ResponseBody` in quanto già sottintesa. Altre annotazioni importanti sono:

- `@PathVariable("nomeVar")` viene usato per gestire le variabili del modello nella mappatura dell'uri della richiesta;
- `@RequestHeader("nomeHeader")` per legare il valore dell' header della richiesta ai parametri del metodo;
- `@RequestBody` per collegare il corpo della richiesta ai parametri del metodo;
- `@CrossOrigin`: Cross-Origin Resource Sharing che consente di abilitare i CORS a livello controller. Il browser infatti non consente di eseguire richieste AJAX verso risorse che stanno al di fuori del dominio corrente.

```
1 @RestController
2 @CrossOrigin
3 public class SocietyController{
4
5     @DeleteMapping("\society\{id}")
6     public ResponseEntity<Response> deleteSociety(
7         @RequestHeader("Authorization") String token,
8         @PathVariable("id") Integer idSociety){ ... }
```

```

9
10    @PutMapping("\society")
11    public ResponseEntity<Response> insertSociety(
12        @RequestHeader("Authorization") String token,
13        @RequestBody Society society){ ... }
14 }
```

Listing 2.4

Utilizzo delle principali annotazioni Spring MVC per Rest Controller

Il Controller è l’artefatto principale di un sistema composto da servizi RESTful. Si occupa di ricevere le richieste e delegarle al livello successivo ossia il *service layer*. I **bean del service layer**, annotati con `@Service` si occupano del livello della business logic e possono collaborare con i bean del livello sottostante, il livello di persistenza, per recuperare i dati da un eventuale database. I **bean del livello di persistenza** sono annotati con la dicitura `@Repository` e si occupano di effettuare operazioni sui dati e interagire con il database attraverso vari framework come MyBatis.

Un altro componente importante della suite Spring è il **controller delle eccezioni** la cui annotazione dedicata è `@ControllerAdvice` o `@RestControllerAdvice`. Questo particolare tipo di controller può essere configurato per gestire le eccezioni che vengono lanciate nel programma e quindi restituire sempre una risposta al client anche in caso di errore.

```

1  @RestControllerAdvice
2  public CustomExceptionController{
3
4      @ExceptionHandler(Exception.class)
5      public ResponseEntity<String> handleEx(Exception e){
6          return ResponseEntity
7              .status(HttpStatus.INTERNAL_SERVER_ERROR)
8              .body("ERROR");
9      }
10 }
```

Listing 2.5

Controller delle eccezioni in Spring

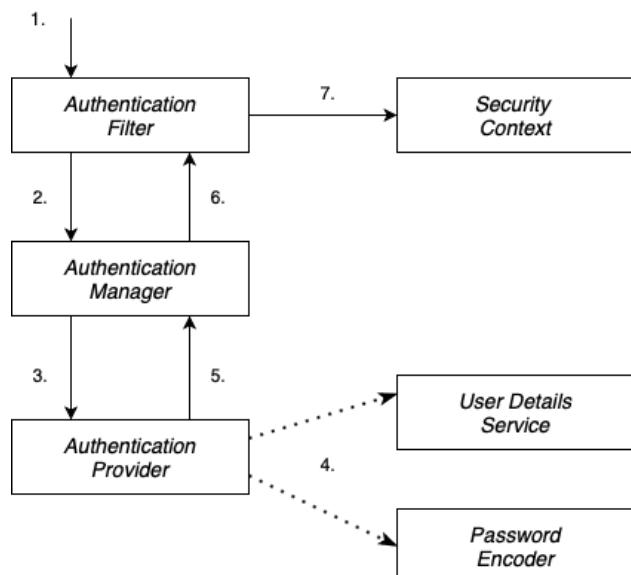
Se viene lanciata un’eccezione, di qualsiasi tipo, durante l’esecuzione dell’applicazione, il controller si occuperà di catturarla e restituire al client il messaggio predisposto.

L'annotazione `@ExceptionHandler`(TIPO ECCEZIONE) si occupa di stabilire di quale tipo di eccezione quel metodo si occuperà. Anche il meccanismo della gestione delle eccezioni è ereditario. Ad esempio, se viene lanciata una `NullPointerException` e non vi è nessun handler che gestisce quel tipo di eccezione, il controllo passa al metodo che gestisce un'eccezione antenata di quella lanciata. Per questo è buona pratica definire un metodo che mappi come tipo di eccezione `Exception` in modo da essere sicuri che tutte le eccezioni vengano controllate.

Spring Security

I software oggi gestiscono una grossa mole di dati, dati che sono considerati sensibili dai requisiti imposti dal GDPR. Ogni informazione sensibile scambiata fra l'utente e l'applicazione deve essere resa sicura e accessibile solo dal proprietario dei dati. Spring Security è un framework potente e personalizzabile per gestire autenticazione, autorizzazione e protezione contro gli attacchi più comuni. L'approccio di Spring Security è, in sostanza, quello di creare dei **filtri** fra l'applicazione e l'esterno. Ogni filtro richiede un diverso approccio e ha un diverso livello di sicurezza. Maggiore è il livello di sicurezza di ogni filtro, minore è la probabilità che un individuo non autorizzato riesca ad accedere all'applicativo e quindi ai dati.

Spring Security mette in campo una catena di filtri così composta:



*Figura 2.8
Autenticazione in Spring Security*

1. Ogni richiesta viene intercettata dall'**Authetication Filter**.
2. L'Authentication Filter delega l'**Authentication Manager**.
3. L'Authentication Manager si serve dell' **Authentication Provider** per implementare la logica dell'autenticazione.

-
4. L'Authentication Provider si serve di due bean: lo **User Details Service** e il **Password Encoder** che si occupano rispettivamente di gestire i dettagli e la password dell'utente. Vengono usati per individuare l'utente e verificarne la password.
 5. Quando l'utente è stato autenticato, il controllo passa dall'Authentication Provider all'Authentication Manager.
 6. Il controllo risale dall'Authentication Manager all'Authentication Filter.
 7. L'Authentication Filter si occupa di salvare nel **Security Context** i dati riguardanti l'autenticazione appena conclusa.

Tutta la catena è altamente configurabile e può essere personalizzata a piacimento sovrascrivendo le configurazioni di default.

2.2.3 JSON Web Tokens

Il **JSON Web Tokens (JWT)** è uno standard per il trasporto sicuro di informazioni fra parti sotto forma di JSON Object.

I JWT sono utili per molteplici scopi:

- **Autorizzazione:** uno scambio di messaggi fra client e server che include sempre il JWT che il server ha fornito al client in fase di login permette l'autenticazione e l'autorizzazione del chiamante ad ogni richiesta;
- **Scambio di informazioni:** il JWT è anche un ottimo metodo per lo scambio sicuro di informazioni fra più parti.

Perchè i JWT Token sono sicuri?

I JSON Web Tokens sono composti da tre parti separate dal punto: **Header**, **Payload** e **Signature**.

Lo **Header** è a sua volta composto da due parti che riguardano: il tipo di token e il tipo di algoritmo utilizzato per la signature. Di questa parte viene fatta la codifica in Base64Url.

Il **Payload** contiene i *claim* ossia le dati riguardanti l'utente ed eventuali informazioni aggiuntive. I claim si dividono in: *registered*, *public* e *private*. I *registered claim* sono i dati che è raccomandabile inserire quando si crea un JWT Token: il mittente, il tempo di scadenza del token, il soggetto e altri. I *public claim* fanno riferimento a parametri definiti nel IANA JSON Web Token Registry. I *private claim*

sono claim definiti dalle parti che si scambiano il token.

Anche di questa parte viene fatta la codifica in Base64Url.

La **Signature** è utilizzata per verificare che il messaggio non sia stato modificato o manomesso durante la comunicazione e che il mittente sia chi dice di essere. Per creare la signature occorre firmare, utilizzando un determinato algoritmo (per esempio il HMAC SHA256) e una chiave privata, lo header e il payload codificati.

Il sito ufficiale dei JSON Web Token nella figura 2.9 mostra la forma finale di un token, i campi che lo compongono e permette di verificare se un token è autentico o meno.

Encoded PASTE A TOKEN HERE

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{  
  "alg": "HS256",  
  "typ": "JWT"  
}
```

PAYOUT: DATA

```
{  
  "nome": "nome_esempio"  
}
```

VERIFY SIGNATURE

```
HMACSHA256(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload),  
  secret  
)  secret base64 encoded
```

SHARE JWT

Figura 2.9
Sito ufficiale JSON Web Tokens: jwt.io.

Durante la fase di autenticazione, quando l'utente effettua il login con successo, il server risponde inviando il JWT Token. Ad ogni richiesta, l'utente dovrà includere nella richiesta il token. Questo tipo di token è di tipo Bearer e deve essere predisposto nell'*Authorization Header*. Ogni volta che il client effettuerà una richiesta al server , il server validerà il token attraverso librerie già predisposte e restituirà al chiamante le risorse cercate (solo nel caso in cui il token sia valido).

2.2.4 Google Firebase

Firebase è una piattaforma utilizzata per lo sviluppo di applicazioni web e mobile. Venne sviluppata da James Tamplin e Andrew Lee nel 2011 come società indipendente e successivamente venne acquisita da Google nel 2014. Firebase offre innumerevoli prodotti fra cui *Firebase Realtime Database*, *Firebase Machine Learning*, *Cloud Functions* e *Firebase Cloud Messaging* utilizzato nel progetto oggetto di questa tesi.

Firebase Cloud Messaging

FCM è una soluzione di messaggistica multi-piattaforma che consente di inviare messaggi. Il servizio è gratuito e mette a disposizione diversi tipi di messaggi verso uno o più dispositivi.

Il corretto funzionamento di FCM è subordinato alla registrazione di ogni dispositivo alla piattaforma. Una volta registrato il dispositivo, quest'ultimo riceverà un token univoco che sarà utilizzato come riconoscimento nell'invio delle notifiche.

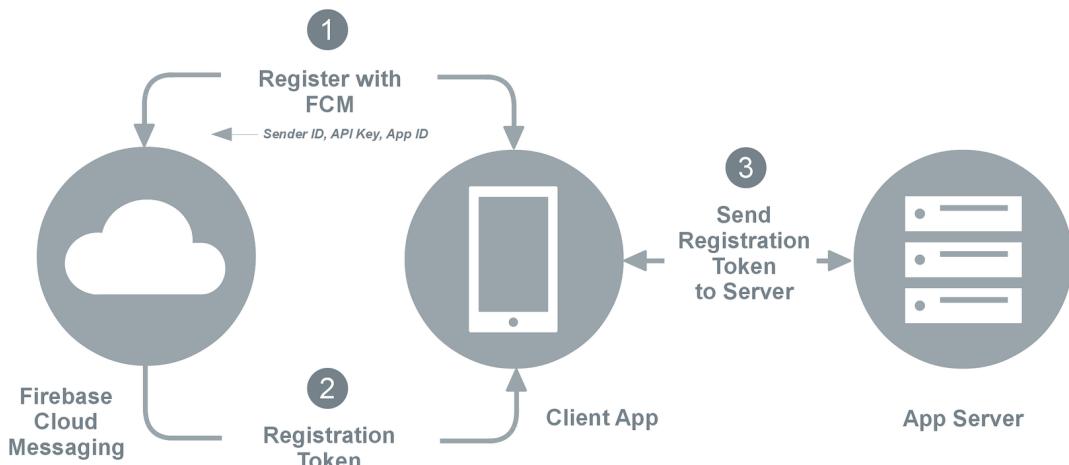
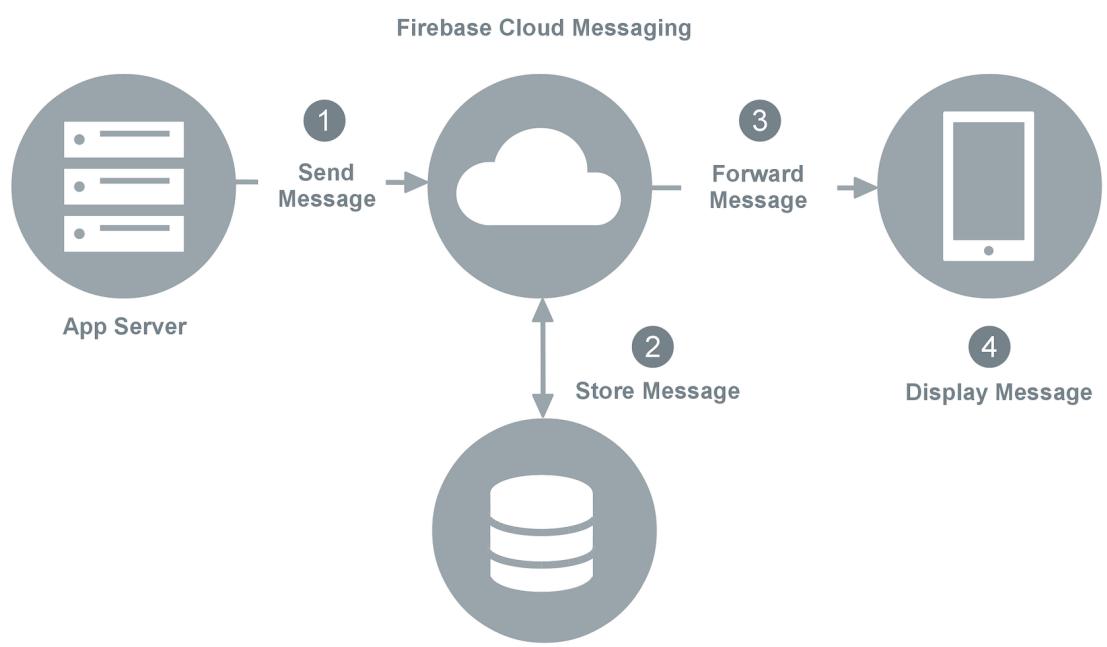


Figura 2.10
Registrazione di un device a FCM

Sarà il server, utilizzando i servizi web esposti da FCM e il token di ogni dispositivo, a generare il messaggio da inviare. I messaggi si dividono in messaggi di notifica e messaggi di dati: i messaggi di notifica sono utilizzati quando non si vuole gestire la visualizzazione di una notifica attraverso l'app client, mentre i messaggi di dati sono utilizzati quando se ne vuole elaborare il contenuto.



*Figura 2.11
Invio di una notifica con FCM*

2.3 Gestione dei dati

2.3.1 MySQL

MySQL è un **RDBMS**⁹. È un database relazione fra i più diffusi, open-source e basato sul linguaggio SQL. Venne creato da David Axmark e Michael Widenius. Nel 1996 divenne open-source e nel giro di pochi mesi riscosse un grande successo. L’idea vincente alla base di MySQL fu il pacchetto di installazione che consentiva di installare e provare il database in 15 minuti. Molti furono i download dell’app che fidelizzò i clienti per i successivi anni.

Un altro punto di forza di MySQL è rappresentato dalla velocità di connessione più bassa rispetto ai database tradizionali. Questa caratteristica è subito stata apprezzata dagli sviluppatori web. Un’applicazione web si connette al database molto più spesso di una tradizionale e MySQL, ottimizzato da questo punto di vista, riscosse subito un grande successo.

MySQL offre un insieme strutturato di tabelle all’interno del quale conservare i dati. Altre caratteristiche fondamentali sono: un’architettura client-server, supporto alle viste, stored procedures, trigger, unicode, transazioni, clustering, funzioni, ricerche full-text, un’architettura a plugin; inoltre è indipendente dalla piattaforma, supporta un gran numero di sistemi, supporta API per numerosi linguaggi di programmazione fra cui Java, C, C++, Perl, PHP e lo standard ODBC.

Esistono diversi strumenti per l’amministrazione di MySQL: nel progetto è stato fatto uso di MySQLWorkBench sviluppato dalla Oracle. Questo software permette la gestione visuale di database locali e remoti tramite protocollo SSH.

2.3.2 MyBatis

MyBatis è un framework open-source evolutiva di IBatis che fornisce un set di API per l’interazione con i database. I punti di forza del framework sono la semplicità e la facilità di utilizzo. MyBatis astrae da tutto ciò che riguarda il codice a basso livello di JDBC come la creazione della connessione verso il database per permettere allo sviluppatore di concentrarsi sulla creazione delle queries.

MyBatis può essere facilmente integrato in un’applicazione Spring Boot aggiungendo la dipendenza al POM. Caratteristiche come l’url del database, l’username

⁹Relational DataBase Management System

e la password per accedervi, vengono definite nelle properties del progetto Spring Boot.

```
1 spring.datasource.url=jdbc:mysql://localhost:3306/qrestaurant  
2 spring.datasource.username=root  
3 spring.datasource.password=root
```

Listing 2.6

Configurazioni di base per l'accesso al database nel file application.properties

L'elemento più importante del framework sono i **Mapper** ossia interfacce che contengono statement SQL mappati su dei metodi. Ogni interfaccia di questo tipo è annotata con `@Mapper`. Attraverso l'annotazione `@MapperScan(PATH)`, Spring Boot potrà poi riconoscere e inizializzare i mapper.

Anche i metodi sono annotati a seconda della query che si vuol eseguire: MyBatis mette a disposizione un'annotazione diversa per ogni statement. Uno *Statement Select* può essere eseguito attraverso l'annotazione `@Select` come nell'esempio che segue.

```
1 @Mapper  
2 public interface UserMapper{  
3     @Select("SELECT * FROM User WHERE idUser=#{idUser}")  
4     User getUserById(@Param("idUser") Integer idUser);  
5 }
```

Listing 2.7

Utilizzo di uno SELECT statement attraverso MyBatis

Nell'esempio 2.7 è stato definito un metodo per recuperare un utente attraverso la chiave primaria della tabella, il suo id. L'annotazione predisposta da MyBatis è la `@Select("query")` dove *query* rappresenta il codice SQL da eseguire.

Un'altra qualità di questo framework è la mappatura dei parametri. Ogni parametro da inserire all'interno della query può essere passato come parametro del metodo mappato. All'interno della query il parametro viene racchiuso tra parentesi graffe e preceduto dal cancelletto. Nel metodo, il parametro viene annotato con `@Param("nome")`.

Il metodo restituisce un utente: MyBatis è in grado di mappare ogni colonna della tabella User in ogni attributo della classe User e creare così un oggetto da restituire, a patto che la query abbia avuto successo e che i nomi delle colonne della tabella e

degli attributi della classe siano gli stessi. Nel caso in cui i nomi non corrispondano, è possibile usare l'annotazione `@Results` per definire il risultato dell'operazione. Le annotazioni `@Insert`, `@Update` e `@Delete` restituiscono rispettivamente il numero di righe inserite, aggiornate ed eliminate.

In conclusione, MyBatis si differenzia dagli altri framework come Hibernate per la visione SQL-centrica che spinge lo sviluppatore verso la scrittura di query invece che evitarle.

2.4 Deployment

2.4.1 Docker

Docker è una piattaforma open-source per l'automazione del deployment delle applicazioni. Docker consente di separare l'applicazione dall'infrastruttura per un rilascio del software veloce. Il deployment delle applicazioni con Docker consente di isolare le applicazioni in ambienti chiamati *container*. I container sono leggeri e contengono tutto ciò che serve all'applicazione. In questo modo, sullo stesso host possono girare contemporaneamente più container che non dipendono dalla macchina su cui sono eseguiti.

Il fulcro di Docker sta nella **CI/CD**: continuos integration and continuous delivery ossia un'ottica di sviluppo, test, integrazione, deployment e monitoraggio continuo degli applicativi.

I container, essendo poi isolati dall'infrastruttura sui cui vengono eseguiti, sono *portabili* ossia possono essere eseguiti su numerosi ambienti senza necessità di modifiche.

Docker utilizza un'architettura client-server. Il *Docker Client* comunica con il *Docker Deamon* attraverso REST API.

- Il Docker Daemon chiamato *dockerd* rimane in attesa di ricevere richieste e si occupa di costruire, eseguire e distribuire i container. Inoltre gestisce oggetti docker come le immagini, i network e i volumi.
- Il Docker Client chiamato *docker* è il modo più semplice per eseguire comandi chiamando dockerd.

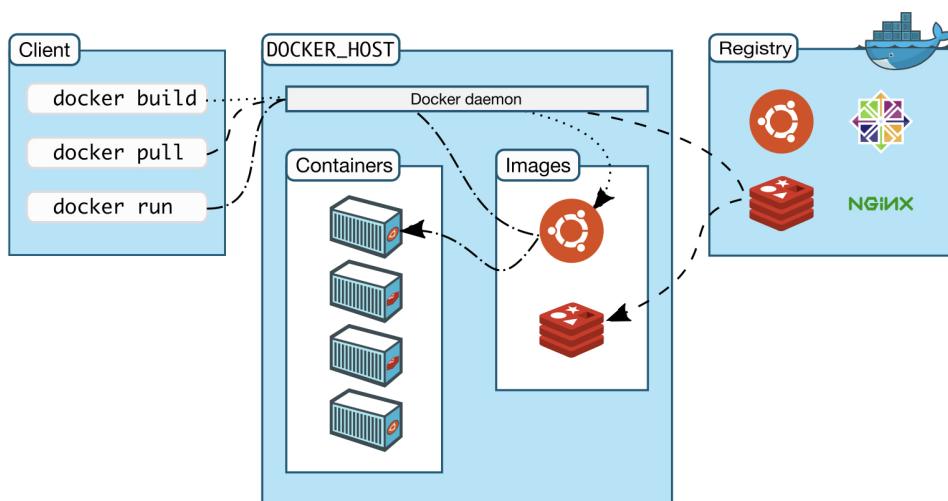


Figura 2.12
Architettura Client-Server di Docker

Oggetti Docker

L'oggetto alla base di Docker sono le **immagini**.

- Un'**immagine** è un set di istruzioni che hanno lo scopo di creare un container. Le istruzioni sono definite in un file chiamato *Dockerfile*. Ogni istruzione crea un layer nell'immagine. Quando il file viene modificato, solo il layer in questione viene cambiato. Ciò rende le immagini piccole, leggere e veloci. Immagini già pronte possono essere trovate su **Docker Hub** una libreria per immagini.
- Il **container** è un'istanza dell'immagine eseguibile. Ogni container può essere creato, inizializzato, fermato, rimosso o spostato. Più container sono isolati gli uni dagli altri. Le configurazioni di un container sono date dall'immagine e da altri elementi definiti quando lo si inizializza o crea.
- Quando un container viene eseguito, Docker crea automaticamente un **network** a cui collegarsi. Più container possono essere connessi ad uno stesso network. Docker fornisce diversi driver al riguardo: bridge (usati per eseguire applicazioni in stand-alone container), host (per usare il network della macchina host), overlay, macvlan. È possibile anche non definire alcun network per un container, in questo caso si parla di none.
- Quando un container viene rimosso, anche tutte le informazioni che esso conteneva vengono rimosse. Per garantire la persistenza dei dati è necessario collegare un **volume** al container. Il volume esiste al di fuori del ciclo di vita del container, non occupa spazio nel container a cui è collegato e può essere riutilizzato da più container contemporaneamente.

Docker Compose

Il **Docker Compose** è un Docker Client utile per la definizione e l'esecuzione di applicazioni Docker multi-container. Con il Docker Compose, le configurazioni di più service (container che vengono eseguiti utilizzando le stesse configurazioni) possono essere racchiuse all'interno di un unico file YAML. Il file di configurazione docker-compose.yml può essere eseguito con un solo comando.

2.5 Version Control

2.5.1 GIT

GIT è stato sviluppato da Linus Torvalds nel 2005. È stato pensato, in un primo momento, per semplificare lo sviluppo del kernel linux e attualmente viene utilizzato come strumento di versionamento del codice. È un sistema distribuito, poichè ogni client ha una copia locale del repository. Supporto non-lineare allo sviluppo, utilizzo di protocolli standard per l'accesso ai repository e adattabilità sia a piccoli sia a grandi progetti sono alcuni vantaggi dell'utilizzo di Git.

Una caratteristica di Git è la presenza di un repository centrale e di un repository locale, copia di quello centrale, che ogni utente può scaricare, per ogni progetto. Quando un utente modifica un file localmente e ne vuole rendere persistente la modifica, effettua un *commit*. Git crea quindi un'immagine dei file al momento del salvataggio. Quando l'utente effettuerà una *push*, tale modifica verrà riportata nel repository remoto. Attraverso Git, funzioni separate l'una dall'altra possono essere sviluppate in rami di lavoro diversi e unite all'occorrenza.

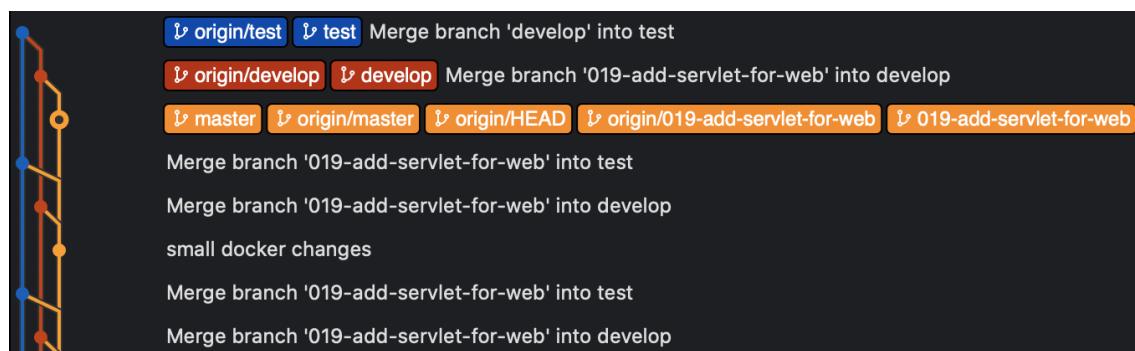


Figura 2.13
Esempio di Git History

Capitolo 3

Progetto

3.1 Introduzione

Il progetto sviluppato nei mesi di tirocinio è nato con lo scopo di fornire un sistema veloce e facilmente utilizzabile dai ristoratori che, per via delle norme volte al contenimento del contagio da COVID-19, hanno l'obbligo di registrare ogni cliente che transita nel locale. L'attuale sistema prevede che i ristoratori registrino i loro clienti manualmente su carta.

Del progetto fanno parte tre applicazioni: *QRestaurant* in versione mobile per iOS e Android rivolta ai clienti e *QRestaurant Business* in versione mobile per iOS e Android e in versione web, *QRestaurant Business Web*, per i ristoratori. *QRestaurant Business* permette di scansionare il codice del cliente, sotto forma di QRCode, generabile da quest'ultimo utilizzando *QRestaurant*, per inserirlo nella lista di clienti che sono transitati nel locale.

L'applicazione *QRestaurant Business*, oltre a consentire al ristoratore di registrare i clienti, permette di aggiungere ristoranti, ricevere notifiche dai clienti su casi di positività rilevati, visionare il proprio profilo e le presenze nel proprio locale ed esportare la lista di clienti che sono stati nel locale nello stesso arco di tempo della persona risultata positiva.

L'applicazione *QRestaurant Business Web* è il corrispettivo della versione mobile sviluppata per il ristoratore ma ideata per il web, realizzata utilizzando la libreria React. A differenza della versione mobile, la versione web non permette di scansionare i QRCode dei clienti ma offre la possibilità di visualizzare statistiche riguardanti il locale come: la possibilità di visualizzare, mese per mese, le statistiche delle presenze registrate nel locale e le segnalazioni di positività ricevute.

L'applicazione *QRestaurant* è l'applicazione mobile sviluppata per il cliente. Permette di generare un QRCode univoco da mostrare ai ristoratori all'ingresso del locale, di visionare la lista di ristoranti frequentati e di inviare una notifica di positività a tutti i proprietari dei locali frequentati dal cliente nelle precedenti 48 ore.

Il back-end comune a tutte le applicazioni è stato scritto utilizzando il framework Spring. In particolare sono stati utilizzati i progetti Spring Boot e Spring Security. Il progetto si basa sull'utilizzo di API RESTful e la sicurezza della comunicazione tra client e server è stata garantita attraverso l'utilizzo dei JSON Web Tokens. La connessione al database MySql è semplificata dall'utilizzo del framework MyBatis mentre Google Firebase rende possibile l'invio di notifiche push verso i client.

Il progetto è stato realizzato da un team all'interno nel quale mi sono occupata di sviluppare il back-end e *QRestaurant Business Web*. Nei seguenti paragrafi vengono presentate le funzionalità di ogni applicazione.

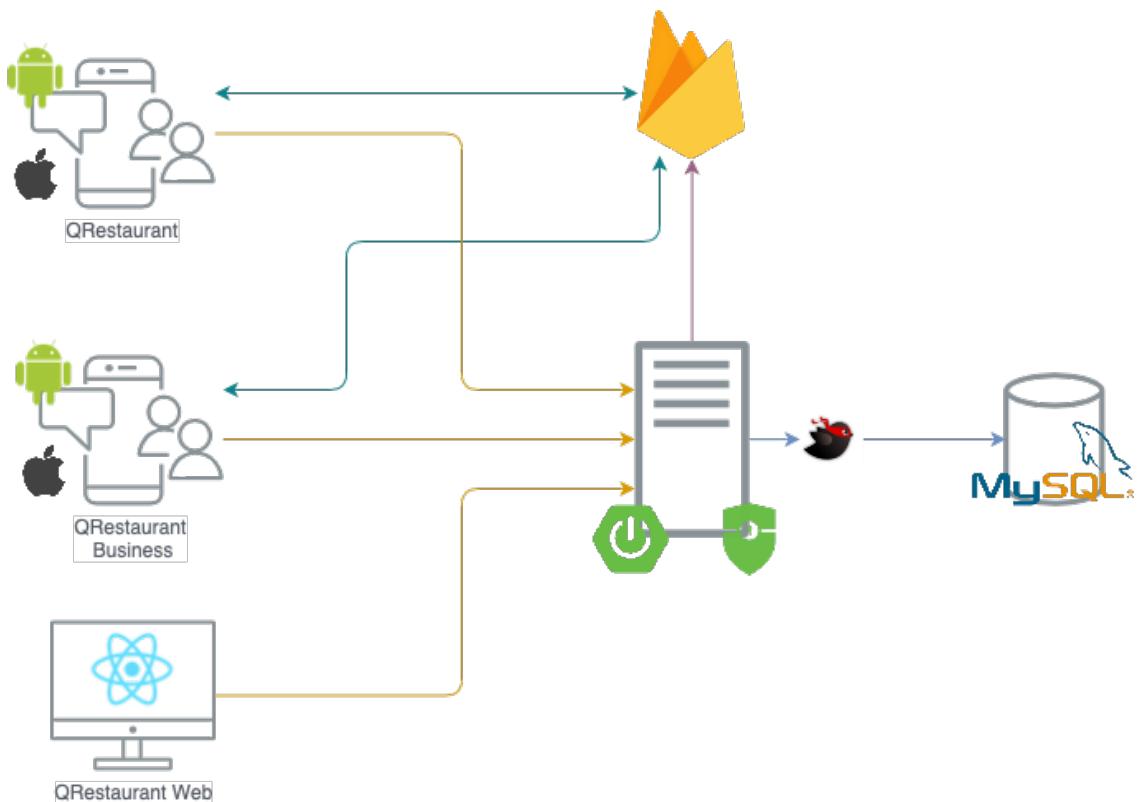


Figura 3.1
Panoramica architettura software

3.2 Front End

3.2.1 QRestaurant

Tutorial

All'apertura dell'applicativo *QRestaurant* vengono presentate le schermate del Tutorial che spiegano all'utente come utilizzare l'applicazione ed alcune importanti informazioni per il corretto funzionamento della stessa.

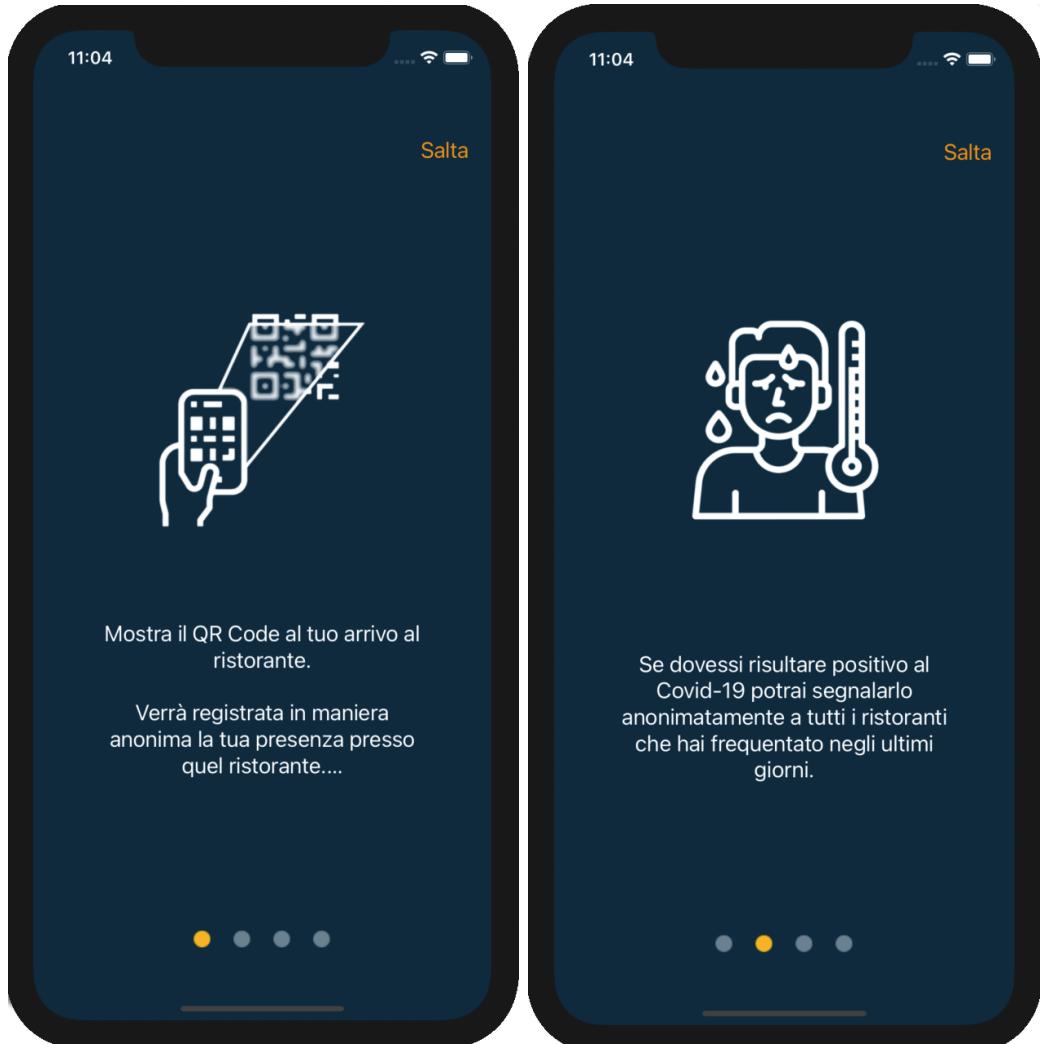
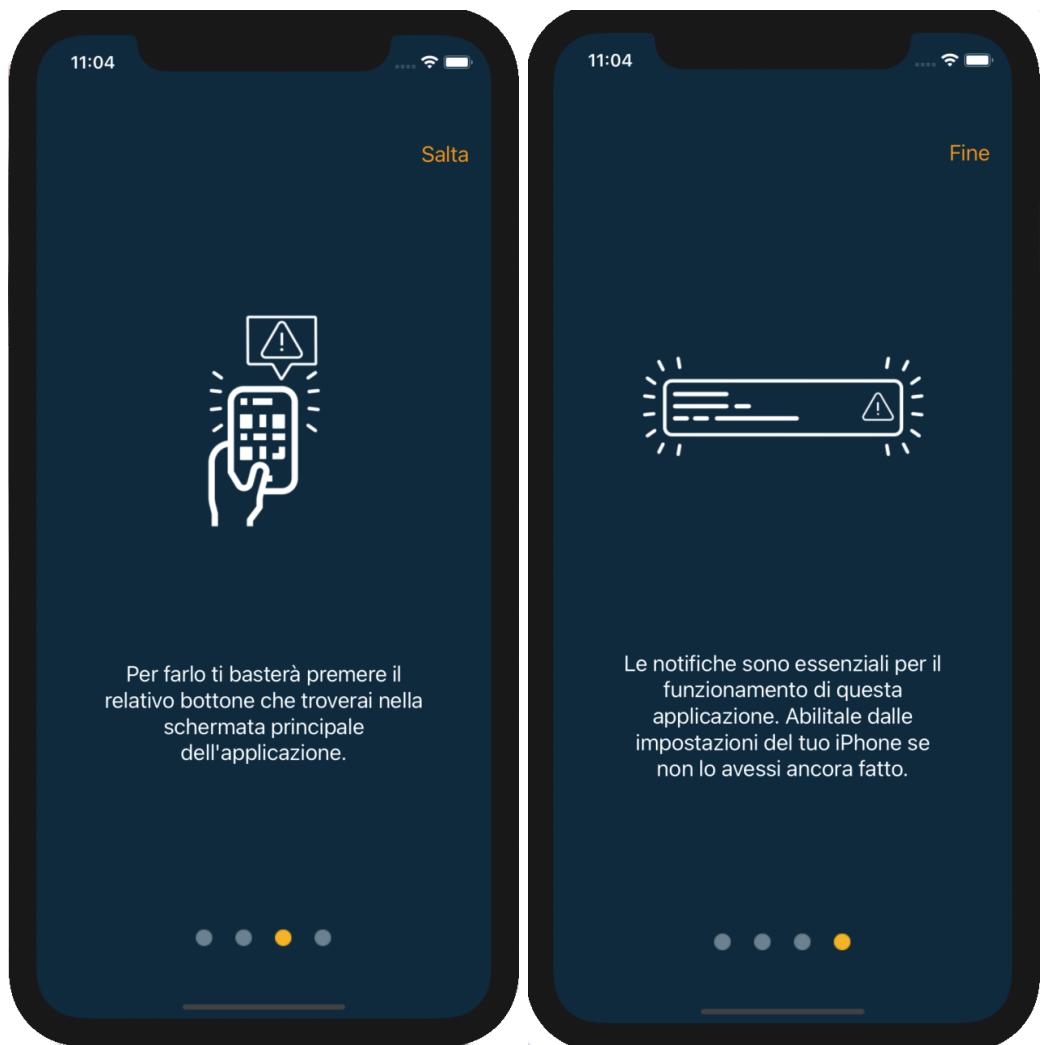


Figura 3.2
QRestaurant: schermate del Tutorial



*Figura 3.3
QRestaurant: schermate del Tutorial*

Registrazione

L'applicazione, pensata per il cliente, prevede una fase di registrazione in cui viene richiesto l'inserimento di dati personali quali *nome*, *cognome* e *numero di telefono*.



Figura 3.4
QR Restaurant: schermata di registrazione

Home page

Superata con successo la fase di registrazione, l'utente accede alla pagina principale dove può visionare il proprio QRCode, accedere alla sezione contenente la lista di ristoranti frequentati e segnalare la positività al Covid-19 ai proprietari dei locali visitati nelle precedenti 48 ore. Il cliente può anche decidere di effettuare il logout. In tal caso, dovrà ripetere il processo di registrazione.



*Figura 3.5
QRestaurant: schermata di Home Page*

Segnalazione di positività

Cliccando il pulsante *Segnala Positività* presente nella home page dell'applicazione, l'utente può segnalare la propria positività al Covid-19 a tutti i proprietari dei ristoranti frequentati nelle precedenti 48 ore.

Il cliente viene reindirizzato ad una schermata per la conferma dell'operazione che contiene ulteriori informazioni circa la segnalazione.

Effettuata la segnalazione, il pulsante *Segnala Positività* sarà disattivato. Al suo posto comparirà il pulsante *Segnala Negatività*, anch'esso disattivato. Quest'ultimo si attiverà automaticamente trascorse 2 settimane dalla segnalazione dando all'utente la possibilità di riprendere ad utilizzare l'applicazione con tutte le sue funzionalità. Il tempo medio per guarire dal COVID-19 è infatti di due settimane.

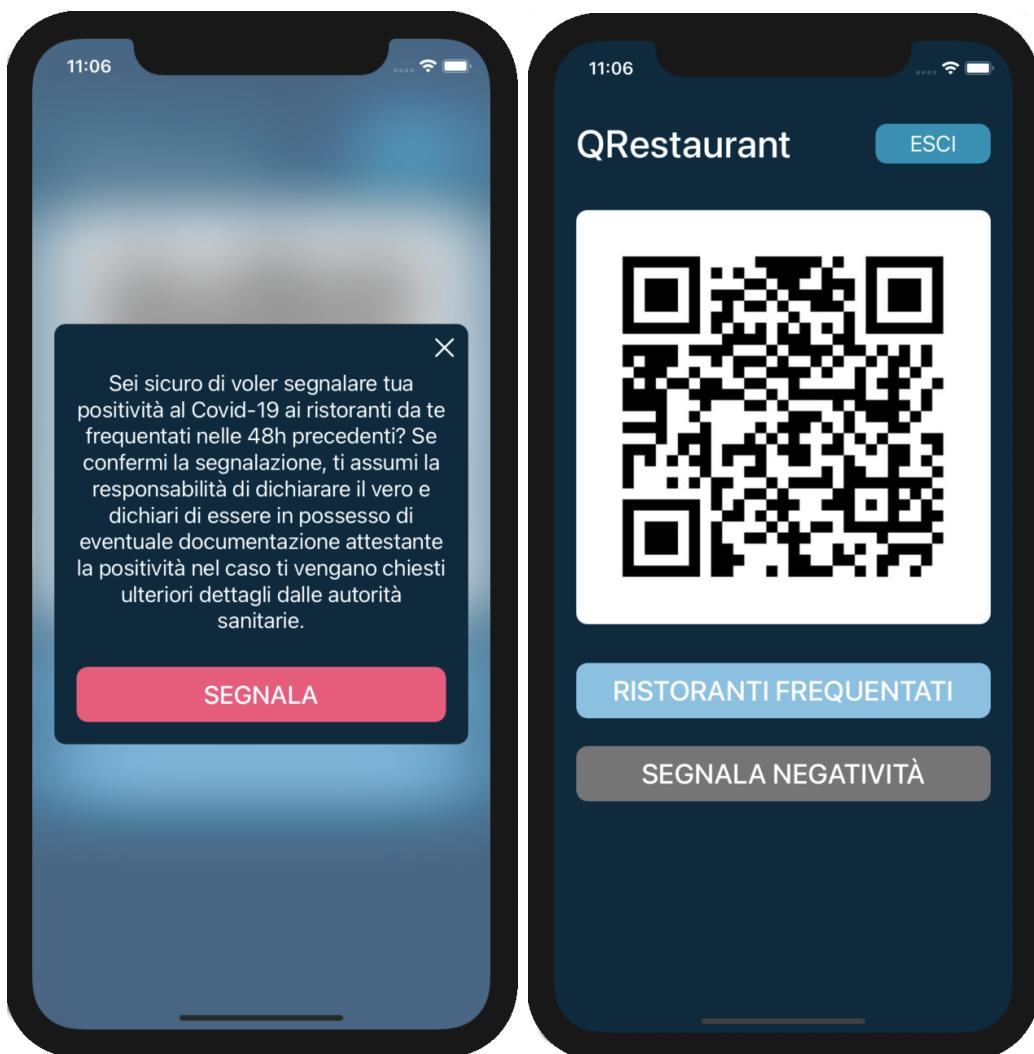


Figura 3.6
QRestaurant: schermata di segnalazione della positività

Notifiche

Ad ogni scansione del suo QRCode il cliente riceve una notifica contenente un messaggio di benvenuto del ristorante appena visitato.

La notifica permette all'utente di essere al corrente della buona riuscita della scansione. Inoltre, grazie al nome del locale visualizzato nel messaggio, l'utente può verificare in prima persona che il suo dispositivo non venga utilizzato in modo malevolo. In aggiunta, ogni codice è utilizzabile una sola volta. Dopo una scansione infatti, il QRCode cambia, il precedente diventa inutilizzabile e non verrà più accettato dal sistema. Per questo è importante disporre di una connessione ad internet quando si utilizza l'app. La notifica push inviata nel momento della scansione funge da allerta per l'applicazione che provvederà a richiedere un nuovo QRCode.



*Figura 3.7
QRestaurant: notifiche push*

Ristoranti Frequentati

Cliccando sul pulsante *Ristoranti Frequentati* nella home page, l'utente può visionare la lista di ristoranti frequentati ossia quelli in cui la sua presenza è stata registrata mediante la scansione del QRCode. Ogni slot mostra data e ora della visita oltre al nome del locale frequentato.



Figura 3.8
QRestaurant: schermata di visualizzazione dei ristoranti frequentati

3.2.2 QRestaurant Business

Tutorial

L'applicazione dedicata ai ristoratori, *QRestaurant Business*, presenta, alla prima apertura, le schermate di tutorial riguardanti il funzionamento per il ristoratore e le principali funzionalità offerte.

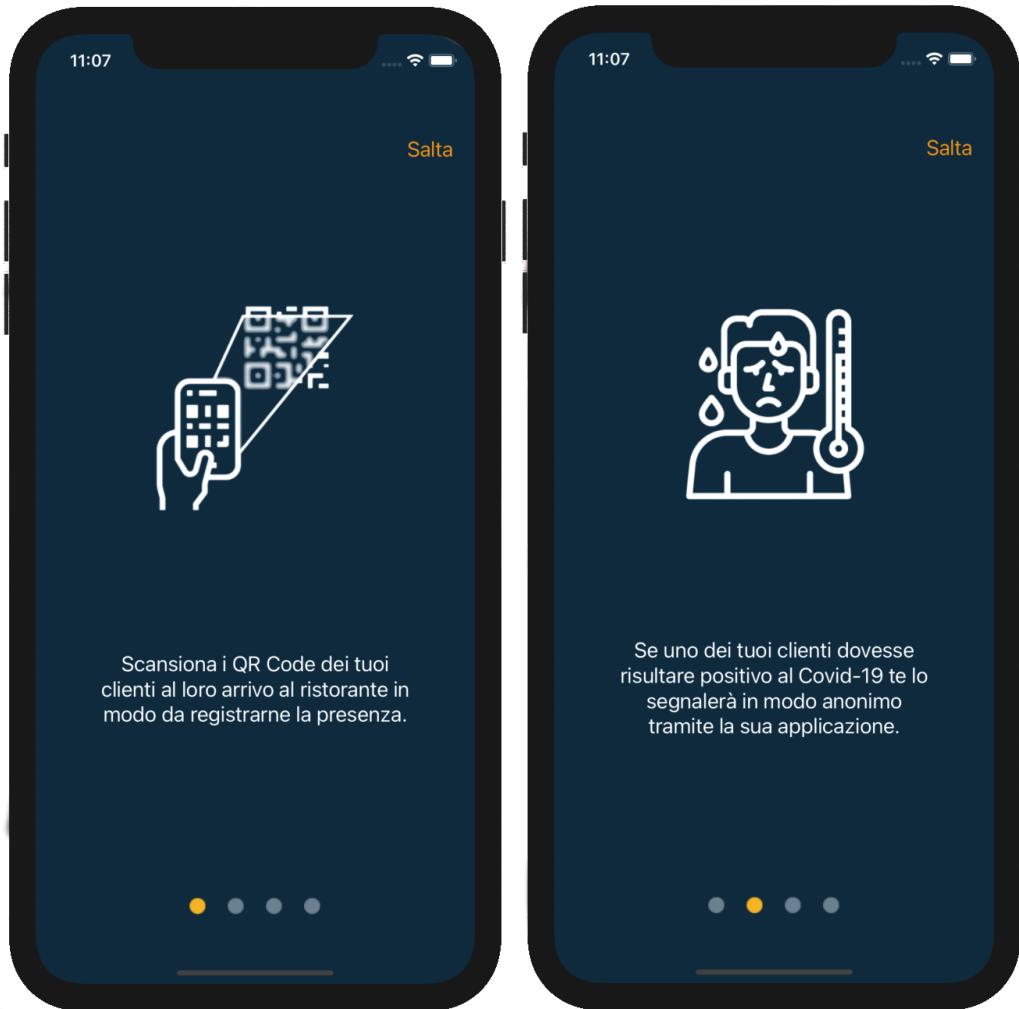
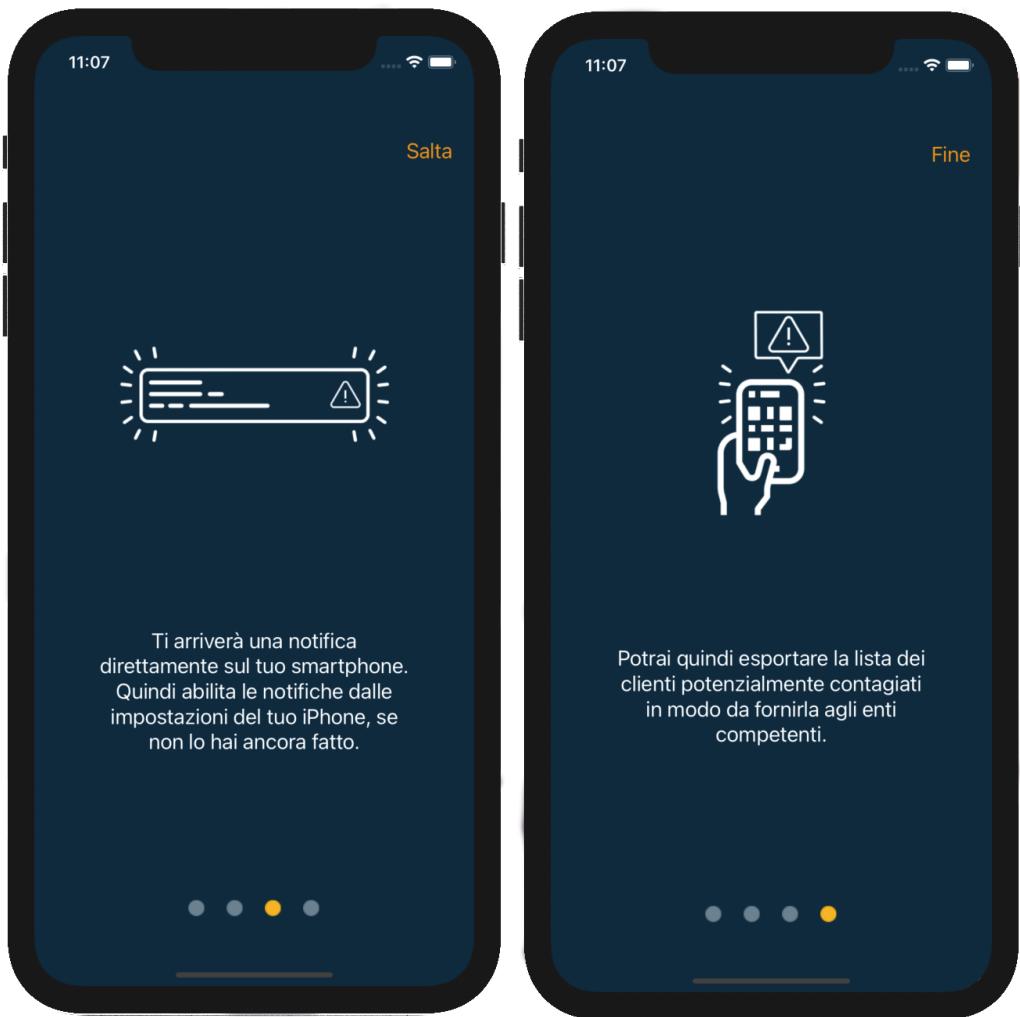


Figura 3.9
QRestaurant Business: schermate di Tutorial



*Figura 3.10
QR Restaurant Business: schermate di Tutorial*

Registrazione

La schermata di registrazione di *QRestaurant Business* richiede al ristoratore di inserire dati personali tra i quali: *nome*, *cognome* e *numero di telefono*. Viene richiesta inoltre una *password* per creare il profilo del ristoratore.

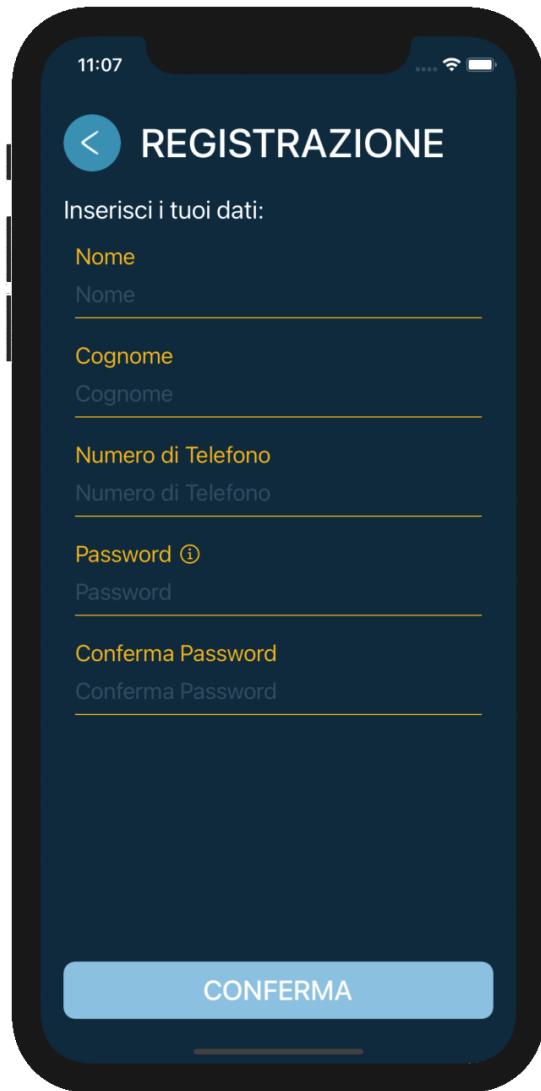
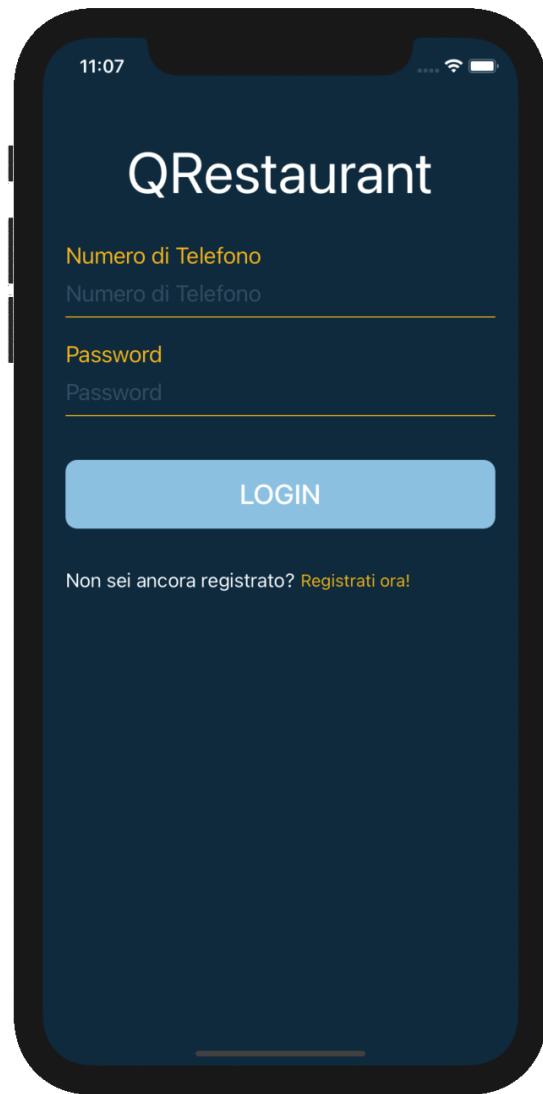


Figura 3.11
QRestaurant Business: schermata di registrazione

Login

Un ristoratore che effettua il logout, non è tenuto a ripetere il processo di registrazione (come invece avviene per il cliente). Dovrà però effettuare il login con le credenziali con cui ha effettuato la registrazione.



*Figura 3.12
QRestaurant Business: schermata di Login*

Home

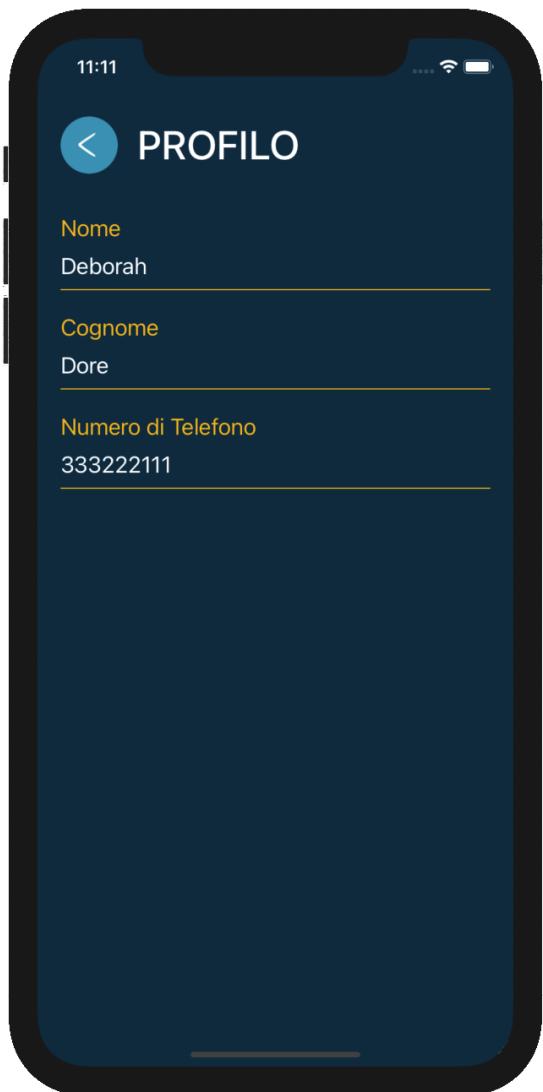
La schermata principale dell'applicazione mostra al ristoratore tutte le funzionalità di *QRestaurant Business*. Da qui è possibile aggiungere un nuovo ristorante, scansionare un QRCode, visualizzare la lista di segnalazioni ricevute, il proprio profilo, le presenze nel locale ed effettuare il logout. Se il processo di registrazione va a buon fine l'utente accede alla Home Page, nella quale verrà richiesto di aggiungere un ristorante. In seguito alla registrazione di un ristorante, l'applicazione mostrerà le informazioni principali dello stesso sulla Home Page. È importante la possibilità di aggiungere più ristoranti, per esempio per i ristoratori proprietari di catene.



Figura 3.13
QRestaurant Business: schermata di Home Page

Profilo

La schermata riguardante il profilo riepiloga alcuni dei campi inseriti in fase di registrazione.



*Figura 3.14
QRestaurant Business: schermata del profilo*

Ristoranti

È possibile visualizzare la lista dei ristoranti registrati nella pagina dedicata, accessibile dalla Home Page cliccando il pulsante *Ristorante Selezionato*. La schermata mostra ogni ristorante registrato e le relative informazioni principali. Si possono inserire ulteriori ristoranti cliccando il pulsante *+* collocato in basso a destra. Per ogni nuovo ristorante che si desidera aggiungere è necessario specificare *nome*, *città*, *indirizzo*, *stato* e *partita IVA*. È possibile modificare il ristorante *in uso* in quel momento cliccando sull'elemento della lista che si desidera utilizzare.

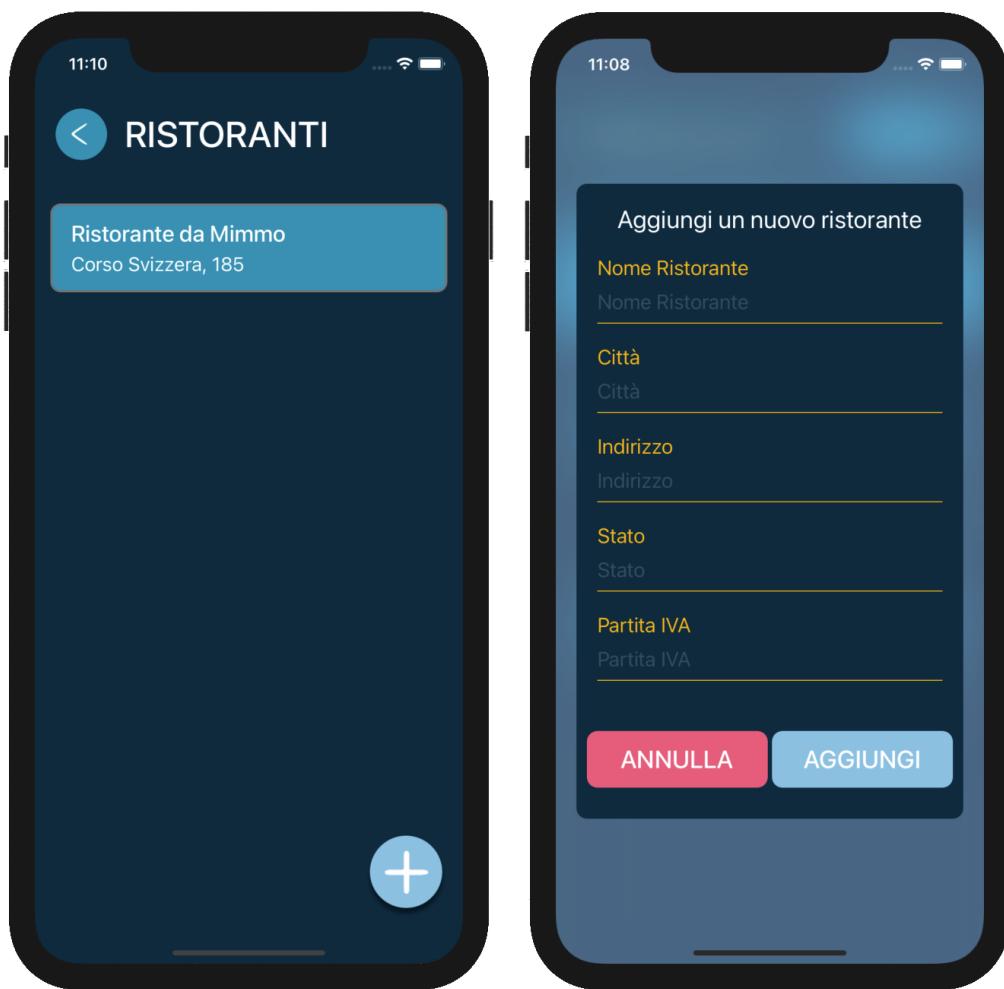


Figura 3.15
QRestaurant Business: schermata di visualizzazione e aggiunta dei ristoranti

Scan QRCode

La schermata di scansione del QRCode permette di inquadrare e scansionare il QR-Code attraverso la fotocamera del device. Il riconoscimento del codice è automatico e il corretto funzionamento della registrazione viene confermato da un *check* che compare al centro della schermata se tutto è andato a buon fine. Dopo la scansione, la schermata non si chiude automaticamente per consentire al ristoratore di scansionare immediatamente altri codici.

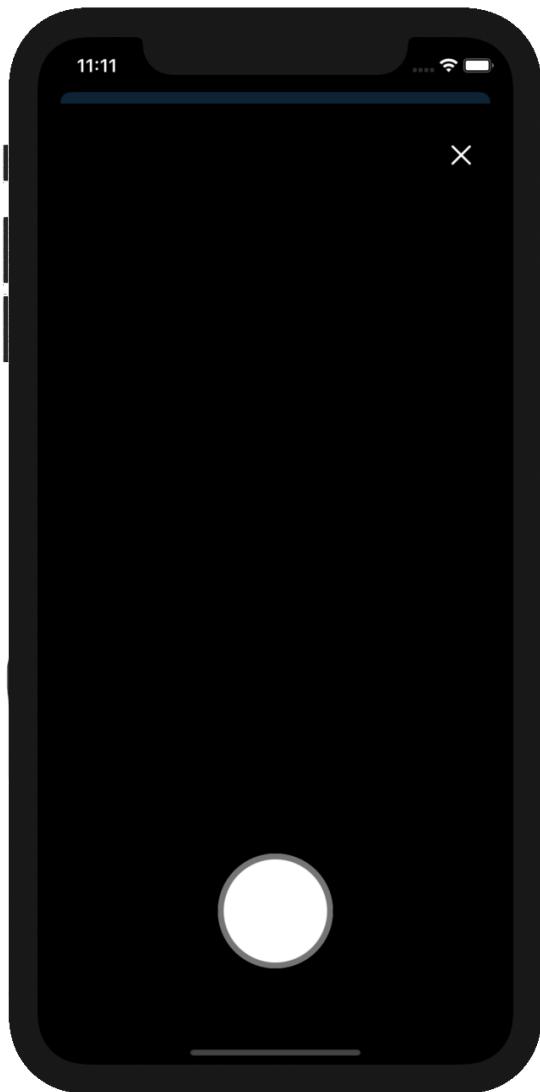


Figura 3.16
QRestaurant Business: schermata di scansione di un QRCode

Presenze

Accedendo alla schermata dedicata alle presenze, il ristoratore può visualizzare quante persone sono transitate nel suo locale in due fasce orarie distinte: pranzo e cena. La scelta del giorno, del mese e dell'anno di cui visualizzare le presenze è possibile grazie all'utilizzo di un calendario.

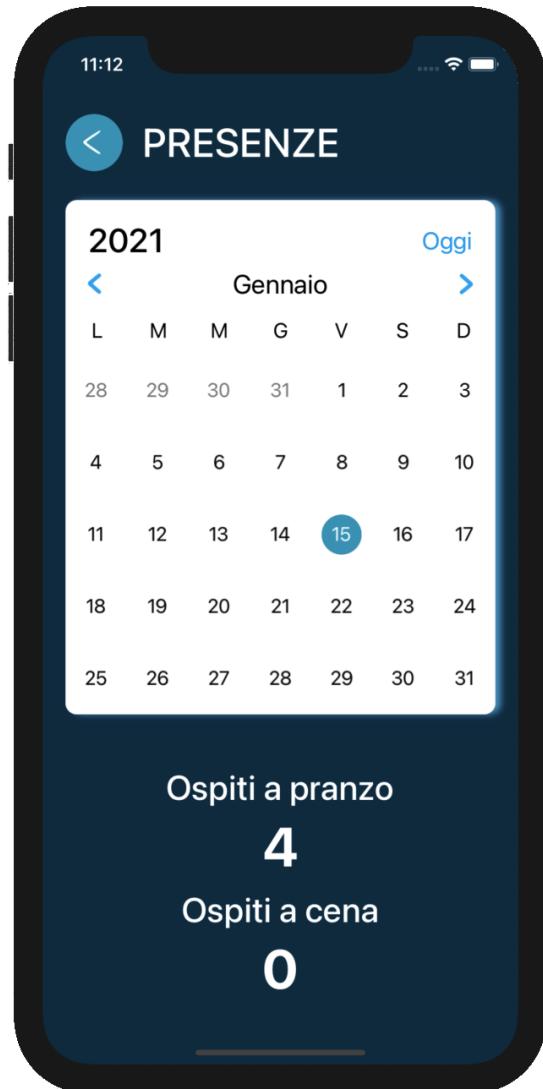


Figura 3.17
QRestaurant Business: calendario delle presenze

Notifiche

Quando un utente segnala la propria positività al COVID-19 utilizzando *QRestaurant*, il proprietario del locale in cui l'utente è stato nelle precedenti 48 ore riceve una notifica. La notifica, in forma anonima, avvisa il ristoratore dell'avvenuta segnalazione. Accedendo alla schermata dedicata in *QRestaurant Business* il ristoratore può ricavare ulteriori informazioni.



Figura 3.18
QRestaurant Business: notifica push di segnalazione della positività

Segnalazioni

Accedendo alla schermata di visualizzazione delle segnalazioni, è possibile visualizzare informazioni utili come *giorno* e *fascia oraria* per i quali il cliente ha effettuato la segnalazione. Inoltre, cliccando sulla segnalazione è possibile esportare la lista di clienti, in formato CSV, che sono stati nel locale nella stesso momento della persona che ha segnalato la propria positività.

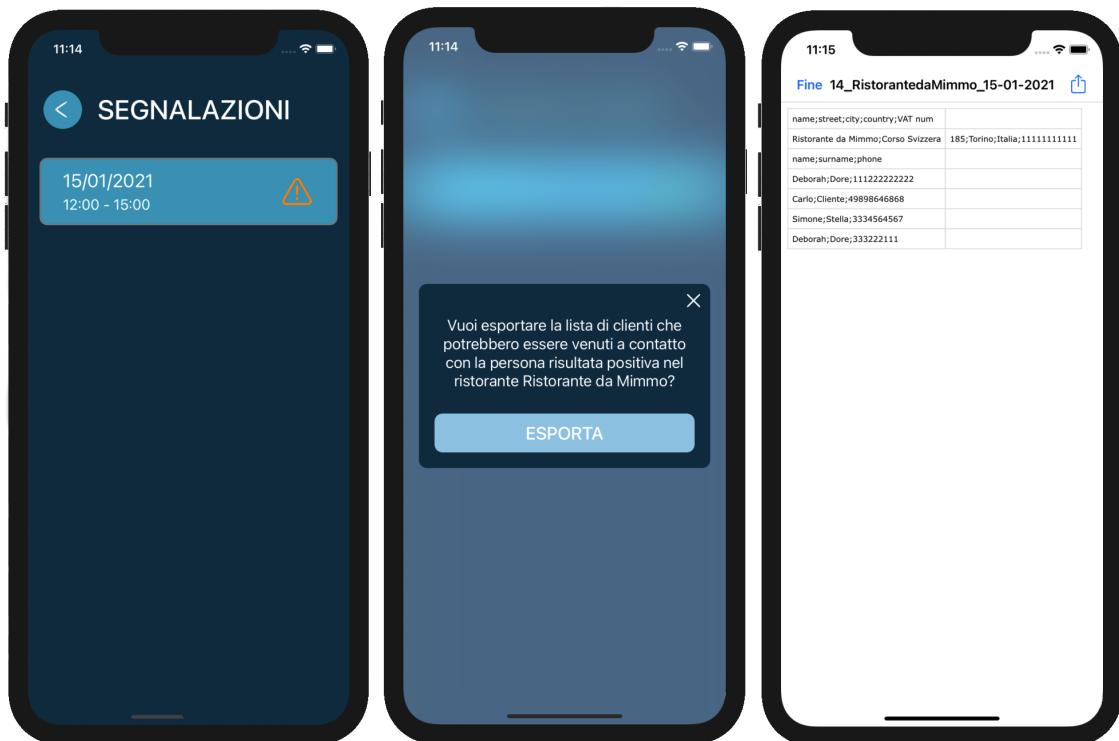


Figura 3.19
QRestaurant Business: schermata segnalazioni

3.2.3 QRestaurant Business Web

L'applicazione web è stata sviluppata interamente utilizzando la libreria JavaScript React. All'uso di React sono state affiancate ulteriori librerie di supporto fra cui le più degne di nota sono:

- *React Bootstrap* per la modellizzazione dell'interfaccia responsive;
- *Axios* per la comunicazione con il server;
- *React Router* per la navigazione all'interno della Single Page Application.

L'applicazione è organizzata in componenti React annidati in cui ogni componente è reso il più indipendente possibile in modo da poterlo riutilizzare in altri contesti.

Le componenti principali dell'applicazione sono due: la *landing page* e la *dashboard*. All'interno della landing page sono presenti due componenti che renderizzano la registrazione e il login (contenenti a loro volta ulteriori componenti annidati). La Dashboard contiene i componenti che renderizzano: la navbar, la sidebar, il profilo, i dati del ristorante, le segnalazioni, le visite e le statistiche.

Con il supporto di React Router tutte le componenti, eccetto per i componenti che renderizzano la pagina di registrazione e la pagina di login, sono stati resi *privati*. Non è possibile infatti accedere alla Dashboard senza aver eseguito il login o la registrazione. La sicurezza è garantita da un componente scritto ad hoc, visionabile nel listato A.1.1. Per accedere alla Dashboard è necessario essere in possesso di un JSON Web Token valido e rilasciato dal back-end con cui l'applicazione web comunica.

Il token, detto *access token*, viene utilizzato principalmente per l'autenticazione del client ad ogni richiesta HTTP con il back-end. Quest'ultimo espone una serie di servizi utili al front-end. Ogni servizio, eccetto quelli di registrazione e login, richiede che il client inserisca nell'header della richiesta, nel campo *Authorization*, l'access token.

Al momento del login e della registrazione il client riceve, insieme all'access token, il *refresh token*. Questo tipo di JSON Web Token viene utilizzato per richiedere un nuovo access token quando quest'ultimo scade e diventa inutilizzabile.

Registrazione

La schermata di registrazione di *QRestaurant Business Web* è la controparte per il web della schermata di registrazione *QRestaurant Business*. Richiede di inserire i seguenti campi del ristoratore: *nome*, *cognome*, *numero di telefono* e *password* come avviene nell'applicazione mobile.

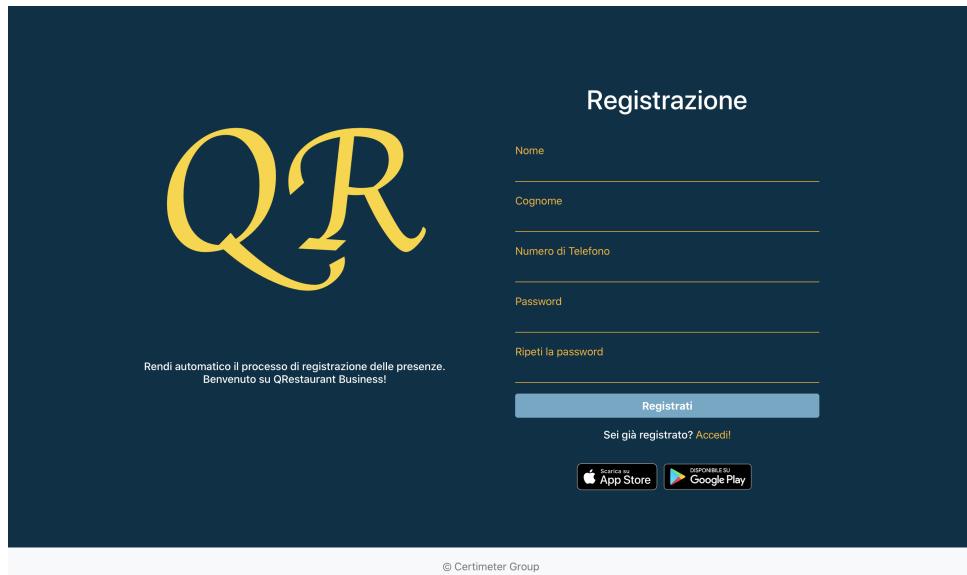


Figura 3.20
QRestaurant Business Web: schermata di registrazione

Login

La schermata di Login richiede di inserire i campi con cui ci si è registrati: *numero di telefono* e *password*.

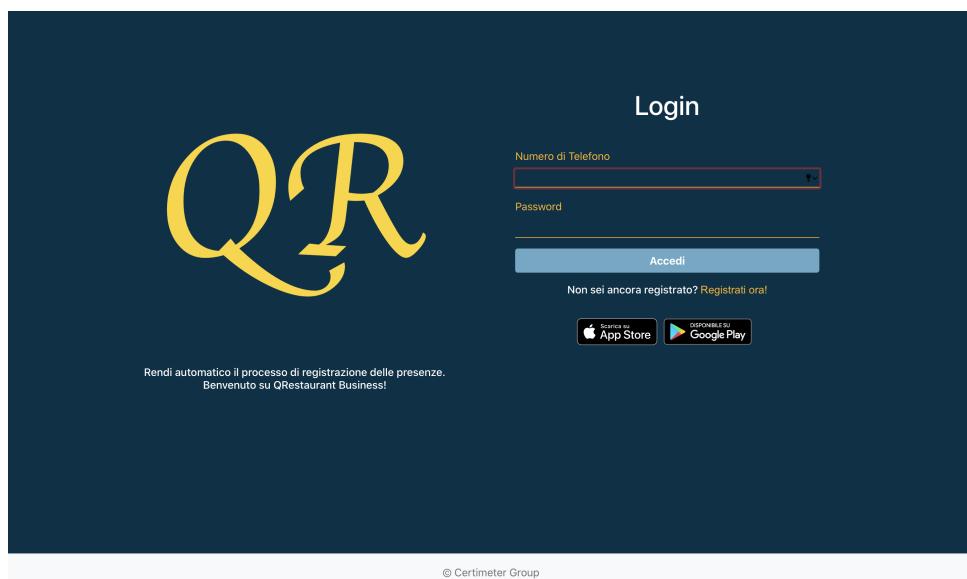


Figura 3.21
QRestaurant Business Web: schermata di Login

Dashboard

Superate la fase di registrazione o di login con successo, il ristoratore accede alla Dashboard. In questa pagina vengono riepilogate tutte le funzionalità disponibili nella versione web: visualizzazione del profilo, visualizzazione dei dati del ristorante attualmente in uso, visualizzazione delle segnalazioni ricevute con conseguente esportazione delle persone venute in contatto con il cliente positivo, calendario delle visite e la visualizzazione delle statistiche.

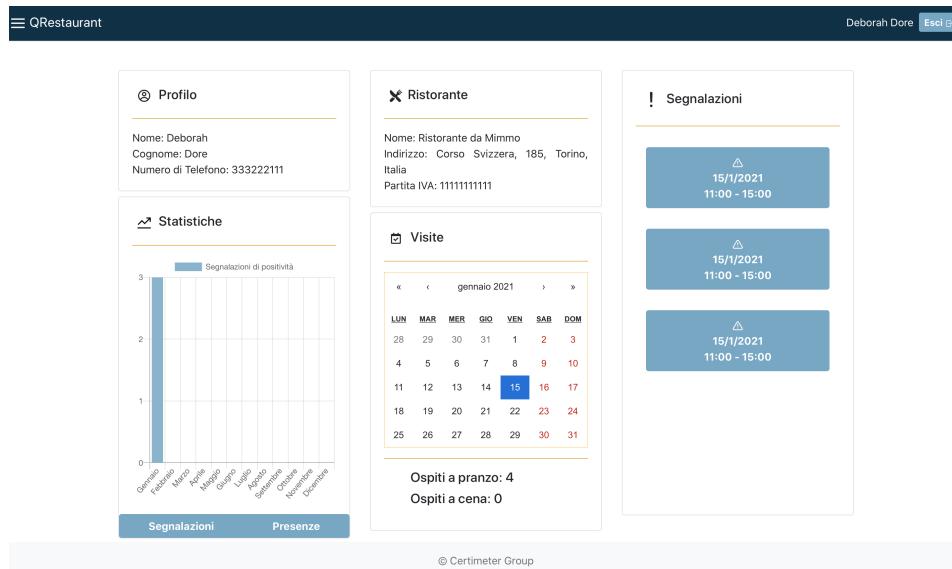


Figura 3.22
QRestaurant Business Web Dashboard

È possibile aggiungere un ristorante o modificare quello in uso aprendo la *sidebar* posta sulla sinistra della pagina.

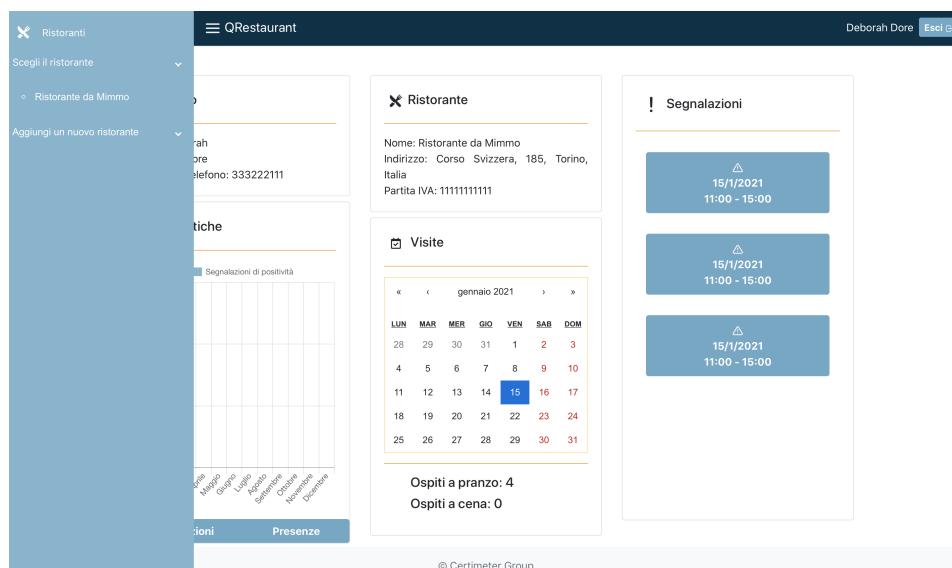


Figura 3.23
QRestaurant Business Web: sidebar

È presente anche nella versione web la possibilità di visualizzare le presenze per fascia oraria del locale selezionato facendo uso del calendario apposito.

The screenshot shows a calendar titled "Visite" for January 2021. The days of the week are labeled LUN, MAR, MER, GIO, VEN, SAB, DOM. The dates are numbered from 28 to 31 for the first week, followed by 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, and 31. The 14th is highlighted in blue, and the 15th is highlighted in yellow. Below the calendar, two statistics are displayed: "Ospiti a pranzo: 0" and "Ospiti a cena: 1".

*Figura 3.24
QRestaurant Business Web: calendario visite*

Le statistiche proposte sono di due tipi: presenze per mese e numero di segnalazioni per mese nel locale. L'utente può scegliere quale visualizzare cliccando l'apposito pulsante.



*Figura 3.25
QRestaurant Business Web: dettagli statistiche*

Attraverso la colonna che elenca le segnalazioni, è possibile esportare la lista di clienti venuti a contatto con la persona che ha effettuato la segnalazione.



Figura 3.26
QR Restaurant Business Web: segnalazioni

The screenshot shows the main dashboard of the QR Restaurant Business Web. On the left, there are sections for Profilo (Profile), Statistiche (Statistics), and Visite (Visits). In the center, there is a modal dialog asking if you want to export a list of clients who may have come into contact with the person who made the report. The dialog includes buttons for Annulla (Cancel) and OK. On the right, there is a column for Segnalazioni (Reports) showing three entries from January 15, 2021, between 11:00 and 15:00. At the bottom, there is a footer with the text "© Certimeter Group".

Figura 3.27
QR Restaurant Business Web: esportare la lista dei clienti

L'aggiunta di un nuovo ristorante prevede che il ristoratore sia al corrente delle seguenti informazioni: *nome*, *indirizzo*, *città* e *partita IVA*. Il nuovo ristorante, una volta completata con successo l'aggiunta, sarà visualizzato nella sidebar.

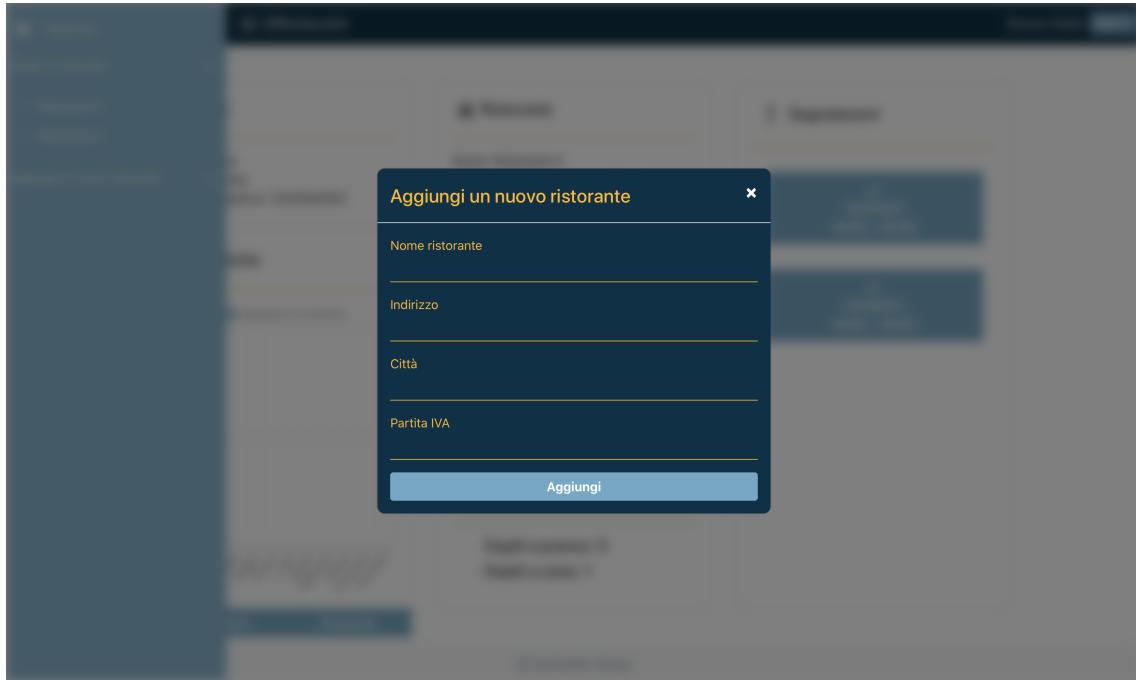


Figura 3.28
QRestaurant Business Web: aggiunta di un ristorante

3.3 Back End

3.3.1 Architettura a tre livelli

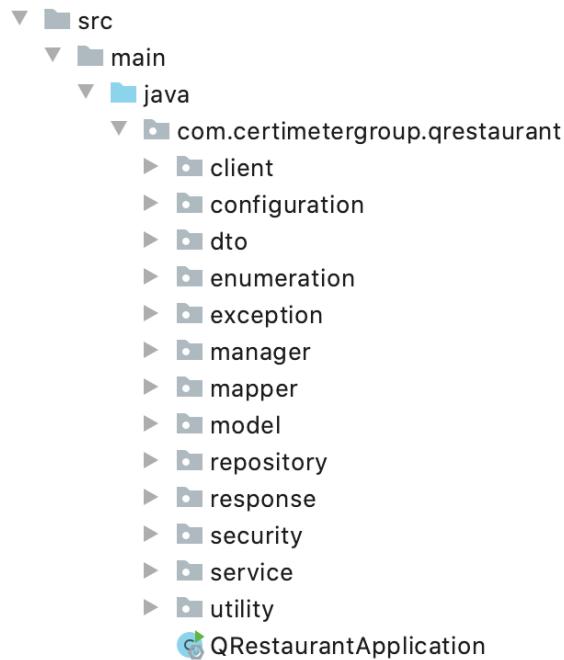
Il back-end è un'architettura a tre livelli. Lo scopo della divisione in livelli è quello di assegnare ruoli e funzioni specifiche a ciascuno di essi in modo da disaccoppiarli il più possibile e focalizzare le responsabilità di ogni componente. I livelli o *layer* sono:

- **Controller Layer:** si occupa di ricevere le richieste provenienti dai vari client, eseguire controlli sui parametri della richiesta e redirigere il flusso dell'applicativo verso il componente del layer successivo più appropriato;
- **Service Layer:** si occupa di gestire la logica di business. È il layer chiamato successivamente a quello del controller;
- **Repository Layer e Mapper Layer** si occupano di recuperare i dati dal database ed effettuare eventuali operazioni di conversione su di stessi; In particolare, il Mapper Layer è stato gestito attraverso il framework MyBatis.

Un generico caso d'uso relativo ad una richiesta HTTP è il seguente:

1. La richiesta HTTP viene intercettata dal controller predisposto che ne valida i parametri e invoca il service più adatto per gestire il servizio richiesto.
2. Nel caso in cui i dati debbano essere recuperati dal database, il service si avvale del repository layer per il recupero degli stessi.
3. Successivamente, il repository individua il mapper necessario al recupero dei dati.
4. Il mapper effettua la chiamata al database e restituisce il risultato al repository che effettua le operazioni necessarie sul dato.
5. Il repository restituisce il risultato al service.
6. Infine, il service si occupa di eseguire le logiche sul dato e restituisce il controllo al controller che invia una risposta al client.

3.3.2 Struttura del Back-end



*Figura 3.29
Struttura del Back-End*

Il progetto è suddiviso in package, ognuno dei quali contiene le classi necessarie per svolgere determinate funzioni:

- **package configuration:** contiene le configurazioni relative all'applicativo: configurazione delle servlet, configurazione degli ambienti di sviluppo, configurazione di Firebase e configurazione di Swagger¹;
- **package client:** contiene gli elementi necessari per gestire le richieste HTTP che hanno prefisso */qrestaurant/client/mobile* ossia tutte le richieste che provengono dall'applicazione mobile *QRestaurant*;
- **package model:** i modelli sono classi Java che rappresentano in modo esatto le entità presenti nel database. Vengono utilizzati esclusivamente all'interno dell'applicativo e non vengono mai restituiti come risposta ai client;
- **package dto:** i Data Transfer Object sono classi java che rappresentano i modelli in modo più o meno esatto e vengono utilizzati per scambiare i dati con i client;
- **package enumeration:** questo package contiene le costanti relative a codici di errore e di successo dell'applicazione;

¹Interfaccia per la descrizione di API RESTful utilizzando JSON

-
- **package exception:** contiene eccezioni strutturate ad hoc per la gestione degli errori;
 - **package manager:** contiene gli elementi necessari per gestire le richieste HTTP che hanno prefisso `/qrestaurant/manager/mobile` e `/qrestaurant/manager/web` ossia tutte le richieste che provengono rispettivamente dall'applicazione mobile *QRestaurant Business* e dall'applicazione web *QRestaurant Business Web*;
 - **package mapper:** contiene i mapper dell'applicazione ossia coloro che effettuano le interrogazioni al database;
 - **package repository:** contiene i repository dell'applicazione che si occupano di effettuare operazioni di conversione sui dati;
 - **package response:** contiene le classi che gestiscono i body delle risposte HTTP. In particolare questo package contiene la classe **Response** che tutte le risposte estendono;
 - **package security:** contiene le configurazioni relative al livello di sicurezza gestito con Spring Security;
 - **package service:** contiene i service base dell'applicazione che gestiscono la logica di business sui dati;
 - **package utility:** classi java che svolgono funzioni statiche utili a tutto l'applicativo.

Per ogni modello è stato creato un mapper, un repository, un service ed un controller apposito.

3.3.3 Creazione di Servlet ad hoc

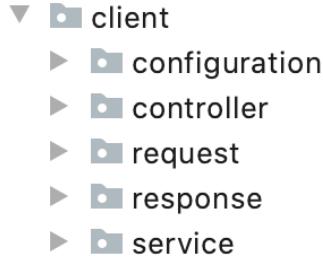
Data la natura del back-end che deve gestire richieste provenienti da tre tipi di front-end diversi, si è scelto di creare tre diverse Servlet.

Ogni servlet gestisce richieste provenienti da uno dei tre front-end:

- Una servlet gestisce le richieste con prefisso `/qrestaurant/client/mobile` ossia quelle provenienti da *QRestaurant*;
- La seconda Servlet gestisce le richieste con prefisso `/qrestaurant/manager/mobile` ossia le richieste che provengono da *QRestaurant Business*;
- Infine, una Servlet gestisce le richieste provenienti dal front-end web, *QRestaurant Business Web*, ossia tutte quelle con prefisso `/qrestaurant/manager/web`.

Il codice di configurazione delle servlet è visionabile nel listato A.2.1. Nella struttura esterna del back-end non è presente un package contenente i controller. Tali package sono annidati dentro i package client e manager e sono relativi a quelle servlet.

Per quanto riguarda il package client, è stata predisposta una sola Servlet. Di seguito la struttura del package:



*Figura 3.30
Struttura del package client*

- **package configuration:** contiene le configurazioni della Servlet;
- **package controller:** contiene tutti i controller relativi alla Servlet che gestisce le richieste di *QRestaurant*;
- **package request:** contiene le classi java che mappano i body delle richieste HTTP;
- **package response:** contiene le classi java che mappano i body delle risposte HTTP;
- **package service:** contiene i service necessari a gestire le logiche di business sui dati.

All'interno del package manager sono annidate due servlet per la gestione di *QRestaurant Business* versione web e mobile.



*Figura 3.31
Struttura del package manager*

Anche in questa struttura sono presenti i **package configuration**, **controller**, **request**, **response** e **service**. In particolare, i service, le request e le response sono utilizzate sia dalla Servlet che gestisce la parte mobile sia da quella che gestisce il lato web in quanto trattano lo stesso tipo di dato: il manager. Le configurazioni e i controller sono gestiti separatamente.

3.3.4 Imbustamento della Risposta

Ogni controller restituisce al chiamante un oggetto di tipo *Response* il cui codice è visionabile nel listato A.2.2. Ogni altra risposta estende questa classe che contiene tre diversi attributi:

1. **Code** di tipo *Integer* che rappresenta il codice della risposta indicante il successo (codice 0) o il fallimento (codici da 1 a 700) della richiesta.
2. **Message** di tipo *String* che indica il messaggio della risposta (associato al codice).
3. **Timestamp** di tipo *Timestamp* indica la data e l'orario di elaborazione della risposta.
4. **Path** di tipo *String* che rappresenta l'identificatore della risorsa richiesta.

Il costruttore di *Response* prende come parametro un oggetto di tipo *ResponseType* che contiene codice e messaggio della risposta. Un esempio di body del messaggio di risposta che comprende i campi base è disponibile nel listato A.2.3.

3.3.5 Gestione delle Eccezioni

Ogni package controller contiene un gestore delle eccezioni annotato con la dicitura `@RestControllerAdvice` di Spring Boot. Questo controller gestisce due tipi di eccezioni: un'eccezione creata *ad hoc* chiamata *Failure Exception* (A.2.4) e la classe *Error* di Java. L'eccezione *Failure Exception* presenta due attributi: *ResponseType* di cui si è già parlato in precedenza e l'*Http Status* che indica il codice di stato HTTP ossia il codice di stato della risposta.

Quando in uno dei qualsiasi livelli dell'applicazione viene lanciata un'eccezione, il controller delle eccezioni configurato per quella Servlet la cattura e, nel caso l'eccezione sia di tipo *Failure Exception* restituisce al client una risposta basata sulla *ResponseType* e sullo *HTTP Status* che quella eccezione contiene.

Nel caso l'eccezione lanciata non sia quella definita, questa verrà gestita restituendo al chiamante un body di errore generico e l'HTTP Status *Internal Server Error*.

3.3.6 Livello di sicurezza e autenticazione

Come già detto in precedenza, la sicurezza dell'applicazione è stata garantita attraverso l'uso dei JWT Token e del framework Spring Security.

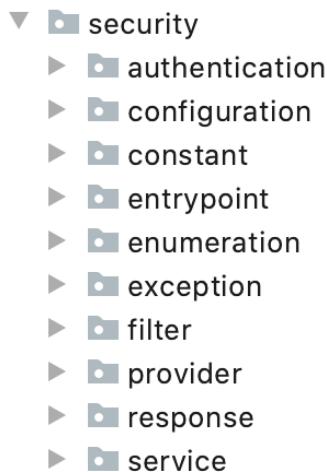
Nel caso di servizi esposti alle applicazioni dedicate al ristoratore, sono stati utilizzati due diversi JSON Web Token: un token chiamato *access token* e un token detto *refresh token* creati rispettivamente a partire da una chiave pubblica e da una chiave privata. Il primo è utilizzato per autenticare l'utente ad ogni richiesta. Contiene nel payload tre elementi utili all'autenticazione: *id*, *numero di telefono* e *uuid del device* dell'utente che sta utilizzando l'applicazione. Questo token scade due minuti dopo la sua creazione. Alla scadenza è necessario richiederne uno nuovo. Per far ciò è necessario il *refresh token*; anch'esso contiene informazioni sull'utente ma, a differenza dell'*access token*, non ha scadenza ed è salvato nel database.

Ad ogni richiesta dei client, l'*access token* deve essere inserito nel campo *Authorization* nell'header della richiesta. Il back-end provvederà a verificare l'integrità del token con appositi metodi messi a disposizione e a verificare l'identità del client autorizzandolo o meno ad accedere al servizio richiesto.

Allo scadere del token, sarà compito del client richiedere un nuovo access token tramite il servizio apposito esposto dal back-end. Il servizio predisposto richiede sia l'*access token* sia il *refresh token*. Di entrambi i token verrà testata l'integrità e la corrispondenza del *refresh token* con quello presente nel database. In caso di successo verranno rilasciati due nuovi token.

Nel caso di richieste provenienti dall'applicazione pensata per i clienti, il token utilizzato non ha scadenza. Viene registrato nel database e ad ogni nuova richiesta ne viene verificata la corrispondenza con quello salvato. Questa scelta è dovuta al fatto che il client non effettua un vero e proprio login ma solo una fase di registrazione in cui non viene richiesta una password. In fase di analisi dei requisiti, infatti, si è scelto di rendere *QRestaurant* il più semplice possibile per gli eventuali utilizzatori.

La verifica dei token e l'autenticazione del client sono descritte all'interno del *package security*:



*Figura 3.32
Struttura del package security*

- **package authentication:** contiene due classi rappresentati i due tipi di token utilizzati: JSON Web Token e Token base;
- **package configuration:** contiene le configurazioni di Spring Security;
- **package constant:** contiene una serie di costanti utili nella fase di autenticazione come l'elenco degli endpoint in cui non è necessario che il client inserisca il token nella richiesta;
- **package entrypoint:** in caso di fallimento nelle fasi di autenticazione, in questo package è contenuta la classe che si occuperà di imbustare la risposta e restituirla al client;
- **package enumeration:** contiene i codici di errore che possono essere restituiti al chiamante in caso di fallimento nelle fasi di autenticazione;
- **package exception:** contiene un'eccezione definita ad hoc che può essere lanciata in fase di autenticazione;
- **package filter:** al suo interno sono presenti i filtri utilizzati per rendere sicura l'applicazione;
- **package provider:** contiene i provider ossia le classi che si occupano di verificare l'integrità dei token e l'identità del richiedente del servizio;
- **package response:** contiene l'oggetto che andrà a rappresentare il corpo della risposta in caso di errore;
- **package service:** contiene i service del livello di sicurezza.

Per verificare l'autenticità di due tipi di token diversi sono stati implementati due *Abstract Authentication Token* differenti: uno che gestisce i token JWT e uno per i token semplici.

Sono stati poi implementati due *Authentication Provider*: un provider supporta solo i JWT token e l'altro solamente i token base.

È stato descritto un filtro che si occupa di determinare se la richiesta proviene da *QRestaurant* o da *QRestaurant Business* nella versione web e mobile e di creare l'*Abstract Authentication Token* apposito per la richiesta. Il token creato verrà poi passato come argomento all'*Authentication Manager* che si occuperà di scegliere il Provider appropriato per verificare il token.

Questi passaggi sono possibili grazie al file di configurazione A.2.5. Quest'ultimo è stato usato per: registrare i provider, aggiungere filtri a quelli già presenti di Spring Security, definire entry point, scegliere il tipo di sessione, specificare quali endpoint sono protetti e quali sono accessibili senza autorizzazione.

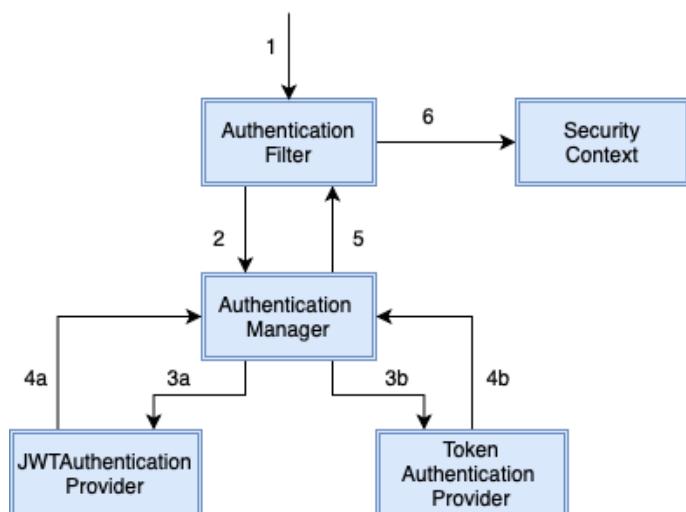


Figura 3.33
Schema di autenticazione di un client utilizzando Spring Security

Nella figura 3.33 è stato schematizzato l'utilizzo dei vari elementi messi a disposizione da Spring Security per realizzare le fasi di autenticazione e autorizzazione:

1. Una richiesta HTTP viene processata dal filtro creato ad hoc ed inserito nella catena di filtri di Spring Security. Il filtro si occupa di determinare il token appropriato da utilizzare: il JWT Authentication Token il Base Authentication Token. Infine viene invocato il metodo *authenticate* dell'Authentication Manager.

-
2. L'Authentication Manager si occupa di scegliere il provider responsabile del token. Ogni Provider implementa un metodo *supports* dove vengono individuati i tipi di Token supportati.
 3. Il controllo passa all'Authentication Provider adatto che implementa le logiche di autenticazione.
 4. Una volta autenticato il client, l'Authentication Provider restituisce il token all'Authentication Manager.
 5. L'Authentication Manager restituisce il token all'Authentication Filter.
 6. L'Authentication Filter imposta il token nel Security Context e chiama il filtro successivo nella catena.

Viene realizzata così la catena di filtri che mette in atto le fasi di autenticazione e autorizzazione.

3.3.7 Ambiente di Sviluppo

Per testare il progetto sono stati creati tre diversi ambienti di sviluppo: **Dev**, **Test** e **Local**. Le configurazioni dei diversi ambienti sono definite all'interno di appositi package nelle risorse dell'applicazione. Per scegliere di usare un ambiente è sufficiente assegnare alla variabile *env* nel file *application.properties* il nome dell'ambiente.

Questa soluzione è resa possibile da un file di configurazione, presente nel listato A.2.6, che dato l'ambiente di sviluppo, all'avvio dell'applicazione Spring Boot si occupa di ricercare il package corretto e di conseguenza le *properties* corrette.

I tre ambienti sono stati usati per fasi diverse:

- L'ambiente **local** è utilizzato quando si vuole testare l'applicativo sul localhost;
- L'ambiente **test** è utilizzato nel momento in cui si vuole distribuire il WAR² dell'applicativo sul server della Certimeter;
- L'ambiente **Dev** è stato utilizzato per creare le immagini dell'applicazione da utilizzare come base di partenza per creare i container di Docker.

²Web application ARchive: formato di file che raggruppa diversi tipi di files.

3.3.8 Notifiche push con Firebase

I servizi offerti da Google Firebase, in particolare da **Google Firebase Cloud Messaging**, sono stati utilizzati per la realizzazione di notifiche personalizzate verso i ristoratori e i clienti che utilizzano *QRestaurant* e *QRestaurant Business*.

I due casi d'uso principali sono:

- La notifica di segnalazione di positività che il ristoratore riceve in seguito alla segnalazione di un cliente;
- La notifica di benvenuto nel locale che il cliente riceve in seguito alla scansione del suo QRcode.

Per poter usufruire del servizio, è necessario registrare l'applicazione nell'apposita console di FCM. Successivamente, per quanto riguarda le applicazioni mobile, è obbligatorio richiedere un token chiamato *firebase registration token* che identifica univocamente un device. Dal progetto creato nella console FCM verrà poi generata una chiave privata, sotto forma di JSON, che dovrà essere utilizzata dal back-end. Inoltre, ogni device dovrà inviare all'applicativo il proprio firebase registration token. Il server, attraverso l'uso dell'SDK messo a disposizione da FCM e della chiave privata, potrà inviare notifiche push ad un device utilizzando il suo firebase registration token.

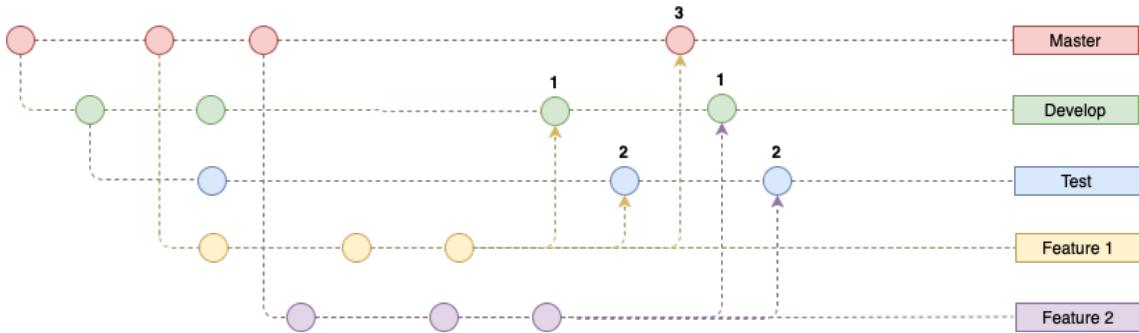
Per quanto riguarda il progetto di stage nella console di FCM sono stati creati due progetti: uno per l'applicazione *QRestaurant* e uno per l'applicazione *QRestaurant Business*. Nel back-end, sono stati quindi generati due bean di tipo *FirebaseMessaging* nel file di configurazione di Firebase all'interno del package configuration, ognuno con la propria chiave privata, utilizzati per inviare notifiche al ristoratore e ai clienti. Ogni qualvolta si rendeva necessario inviare una notifica veniva scelto quale bean utilizzare in base al destinatario. All'interno della notifica è stato inserito un messaggio ad hoc che le applicazioni mobile si limitano a mostrare all'utente.

3.3.9 Branching Strategy

Nel corso del progetto è stato usato *GIT* come metodo per il versionamento del codice. Per tener traccia dei cambiamenti, visionare i cambiamenti nel codice e avere una gestione *pulita* dello stesso, si è scelto di affiancare all'uso del software l'utilizzo di una strategia di branching particolare.

Vi sono tre diversi branch principali: **master**, **develop** e **test** che vengono creati nelle fasi iniziali di realizzazione del progetto. Si riferiscono a tre versi ambienti di sviluppo: develop indica l'ambiente locale, test indica quello di test e master

quello di produzione (non utilizzato). Quando viene creata una nuova *feature*, viene staccato un nuovo branch direttamente dal branch master. I nuovi branch, una nuova completezza di funzionalità per cui erano stati creati e in base alle esigenze del cliente, vengono poi uniti ai branch develop, test e master. Viene utilizzata questa branching strategy, chiamata *issue-driven* ossia *guidata dagli eventi*, quando vi è la necessità di inserire solo determinate feature su determinati ambienti ma nuove funzionalità devono essere già pronte per successivi rilasci. Di seguito uno schema esplicativo.



*Figura 3.34
Branching Strategy: issue-driven*

Come è evidenziato nella figura 3.34, una nuova feature, *feature 1*, viene creata a partire dal branch master. Dopo vari commit, la feature viene unita prima al branch develop, poi al branch test ed infine al branch master. Una seconda feature, *feature 2*, viene creata a partire da master ma dopo gli sviluppi viene unita solamente ai branch develop e test. Verrà successivamente inserita nell'ambiente di PROD quando il cliente lo richiederà.

3.4 Deployment

Le fasi di test del progetto hanno coinvolto anche il software Docker. Sono stati creati due container a partire dalle immagini del database MySQL e del progetto del back-end.

Il listato A.3.2 presenta il file di configurazione docker utilizzato nel progetto back-end. All'inizio del file è presente la versione del docker compose che si intende utilizzare, i volumi e i network dichiarati. Alla voce *services* sono presenti i servizi che si vogliono eseguire: *MySQL* e *QRestaurant*. Il primo rappresenta il servizio che si occuperà di gestire il database, il secondo è il progetto back-end.

L'immagine mysql è stata scaricata direttamente dal Docker Hub ed è indicata alla voce *image*. Le voci *container_name*, *environment*, *ports*, *volumes* e *networks* servono rispettivamente per: impostare il nome del container, configurare le variabili d'ambiente del container, mappare le porte su cui il container rimarrà in ascolto, collegare il volume per la persistenza dei dati e impostare un network dedicato. La voce *healthcheck* definisce delle istruzioni che servono al docker per determinare se un container è in buono stato o meno.

L'immagine qrestaurant è invece stata creata a partire dal Dockerfile dedicato, riportato nel listato A.3.1. Quest'ultimo, attraverso l'utilizzo di maven, definisce una serie di istruzioni volte alla creazione dell'immagine del progetto. Le istruzioni per la creazione del servizio, all'interno del file docker-compose.yml, non si differenziano troppo da quelle utilizzate per creare il container mysql. Uniche voci aggiuntive sono: *build* che indica a docker dove poter recuperare l'immagine a partire dal quale creare il container e *depends_on* che indica a docker che questo particolare servizio dipende da un altro e che quindi deve essere eseguito dopo che è stato avviato il primo servizio.

Una volta eseguito il comando *docker-compose up*, Docker si è occupato di creare le immagini, i container, i network e i volumi descritti nel file *docker-compose*.

Capitolo 4

Sviluppi Futuri

L'applicazione sviluppata durante il percorso di tirocinio presso la Certimeter mi ha permesso di imparare nuove tecnologie, largamente utilizzate nel mondo del lavoro.

Oggi, da dipendente dell'azienda, ho l'opportunità di continuare a lavorare sul progetto che ha subito notevoli evoluzioni per essere adattato ad una visione più generica e rivolta non solo ai ristoratori e ai loro clienti ma ad un pubblico di utillizzatori più ampio. Per rendere la scansione del QRCode più rapida, sono stati invertiti i ruoli: in *QAccess*, così è stata chiamata la nuova applicazione, è il cliente che scansiona il QRCode di una società per registrarvisi. La grafica è stata modificata e sono state aggiunte funzionalità come la *One Time Password* per la verifica del numero di telefono. Ritengo di aver apportato un notevole contributo alle fasi di modifica e rivisitazione dell'applicazione, rendendola meno polarizzata al fine di adattarla ad un nuovo scopo.

In conclusione, posso affermare che il percorso è stato senza dubbio positivo e mi ha introdotta nel mondo del lavoro in un contesto consecutivo al percorso universitario.

Elenco delle figure

1.1	Pattern MVC schematizzato	4
1.2	Esempio di HTTP Request	8
1.3	Esempio di HTTP Response	8
2.1	Esempio di struttura ad albero di una pagina HTML.	18
2.2	Ciclo di vita dei componenti in React	34
2.3	Esempio di organizzazione a griglia Bootstrap	36
2.4	Breakpoint a confronto: medium	36
2.5	Breakpoint a confronto: small	36
2.6	Container di Spring	42
2.7	Spring Initializr	44
2.8	Autenticazione in Spring Security	47
2.9	Sito ufficiale JSON Web Tokens: jwt.io.	49
2.10	Registrazione di un device a FCM	50
2.11	Invio di una notifica con FCM	51
2.12	Architettura Client-Server di Docker	55
2.13	Esempio di Git History	57
3.1	Panoramica architettura software	59
3.2	QRestaurant: schermate del Tutorial	60
3.3	QRestaurant: schermate del Tutorial	61
3.4	QRestaurant: schermata di registrazione	62
3.5	QRestaurant: schermata di Home Page	63
3.6	QRestaurant: schermata di segnalazione della positività	64
3.7	QRestaurant: notifiche push	65
3.8	QRestaurant: schermata di visualizzazione dei ristoranti frequentati .	66
3.9	QRestaurant Business: schermate di Tutorial	67
3.10	QRestaurant Business: schermate di Tutorial	68
3.11	QRestaurant Business: schermata di registrazione	69
3.12	QRestaurant Business: schermata di Login	70
3.13	QRestaurant Business: schermata di Home Page	71

3.14	QRestaurant Business: schermata del profilo	72
3.15	QRestaurant Business: schermata di visualizzazione e aggiunta dei ristoranti	73
3.16	QRestaurant Business: schermata di scansione di un QRCode	74
3.17	QRestaurant Business: calendario delle presenze	75
3.18	QRestaurant Business: notifica push di segnalazione della positività .	76
3.19	QRestaurant Business: schermata segnalazioni	76
3.20	QRestaurant Business Web: schermata di registrazione	78
3.21	QRestaurant Business Web: schermata di Login	78
3.22	QRestaurant Business Web Dashboard	79
3.23	QRestaurant Business Web: sidebar	79
3.24	QRestaurant Business Web: calendario visite	80
3.25	QRestaurant Business Web: dettagli statistiche	80
3.26	QRestaurant Business Web: segnalazioni	81
3.27	QRestaurant Business Web: esportare la lista dei clienti	81
3.28	QRestaurant Business Web: aggiunta di un ristorante	82
3.29	Struttura del Back-End	84
3.30	Struttura del package client	86
3.31	Struttura del package manager	86
3.32	Struttura del package security	89
3.33	Schema di autenticazione di un client utilizzando Spring Security . .	90
3.34	Branching Strategy: issue-driven	93

Elenco delle tabelle

1.1 Esempio di utilizzo dei metodo HTTP.	13
--	----

Riferimenti bibliografici

- [1] *AOP, la programmazione orientata agli aspetti.* Disponibile on line. URL: <https://www.appuntisoftware.it/aop-la-programmazione-orientata-agli-aspetti/>.
- [2] *Apache Maven.* Disponibile on line. URL: https://it.wikipedia.org/wiki/Apache_Maven.
- [3] *ASCII.* Disponibile on line. URL: <https://it.wikipedia.org/wiki/ASCII>.
- [4] Alex Banks e Eve Porcello. *Learning React.* O'Reilly Media, Inc, 2017.
- [5] *Bootstrap.* Sito web ufficiale. URL: <https://getbootstrap.com/docs/5.0/getting-started/introduction/>.
- [6] Douglas Crockford. *JavaScript: The Good Parts.* O'Reilly Media, Inc, 2008.
- [7] *CSS.* Disponibile on line. URL: <https://it.wikipedia.org/wiki/CSS>.
- [8] *CSS: Cascading Style Sheets.* Disponibile on line. URL: <https://www.informarsi.net/css-cascading-style-sheets/>.
- [9] *Dall'hosting al cloud computing: storia del protocollo HTTP.* Disponibile on line. URL: <https://www.hostingtalk.it/hosting-cloud-computing-storia-http/>.
- [10] *Docker Docs.* Sito web ufficiale. URL: <https://docs.docker.com>.
- [11] *Ecma International.* Sito web ufficiale. URL: <https://www.ecma-international.org>.
- [12] *Firebase Cloud Messaging.* Sito Web Ufficiale. URL: <https://firebase.google.com>.
- [13] *GIT.* Disponibile on line. URL: [https://it.wikipedia.org/wiki/Git_\(software\)](https://it.wikipedia.org/wiki/Git_(software)).
- [14] *Guida JavaScript.* Disponibile on line. URL: <https://www.html.it/guide/guida-javascript-di-base/>.
- [15] *HTML.* Disponibile on line. URL: <https://it.wikipedia.org/wiki/HTML>.
- [16] *Hypertext Transfer Protocol.* Disponibile on line. URL: https://it.wikipedia.org/wiki/Hypertext_Transfer_Protocol.
- [17] *I transpiler, cosa sono e come vengono usati?* Disponibile on line. URL: <https://www.html.it/05/09/2017/i-transpiler-cosa-sono-e-come-vengono-usati/>.

-
- [18] *Introduzione alle API REST*. Disponibile on line. URL: <http://databasemaster.it/introduzione-alle-api-rest/>.
 - [19] *IoC Container*. Disponibile on line. URL: <https://www.html.it/pag/18719/ioc-container/>.
 - [20] *JavaScript*. Disponibile on line. URL: <https://it.wikipedia.org/wiki/JavaScript>.
 - [21] *JSON Web Tokens*. Sito Web Ufficiale. URL: <https://jwt.io>.
 - [22] *Model-View-Controller*. Disponibile on line. URL: <https://it.wikipedia.org/wiki/Model-view-controller>.
 - [23] *React – Una libreria JavaScript per creare interfacce utente*. Sito web ufficiale. URL: <https://it.reactjs.org>.
 - [24] *React Bootstrap*. Sito web ufficiale. URL: <https://react-bootstrap.github.io>.
 - [25] *React Router: Declarative Routing for React.js*. Sito web ufficiale. URL: <https://reactrouter.com>.
 - [26] *Respresentational State Transfer*. Disponibile on line. URL: https://it.wikipedia.org/wiki/Representational_State_Transfer.
 - [27] *Sistema Client Server*. Disponibile on line. URL: https://it.wikipedia.org/wiki/Sistema_client/server.
 - [28] Laurentiu Spilca. *Spring Security in Action*. Manning Pubns Co, 2020.
 - [29] *Spring*. Sito web ufficiale. URL: <https://spring.io>.
 - [30] *Spring*. Disponibile on line. URL: https://it.wikipedia.org/wiki/Spring_Framework.
 - [31] *Web Services Architecture*. Disponibile on line. URL: <https://www.w3.org/TR/ws-arch/>.
 - [32] *What is Axios and how to use it with React*. Disponibile on line. URL: <https://medium.com/javascript-in-plain-english/what-is-axios-and-how-to-use-it-with-react-1470d19e1b83>.

Ringraziamenti

Ai miei genitori, che mi hanno sostenuta (e finanziata) sin da subito nella mia scelta. È grazie a voi se ho realizzato quello che volevo. Grazie perché non mi avete mai ostacolata anche se sapevate che ci saremmo visti poche volte all'anno. Vi porto sempre con me.

A Gabriele, che mi è stato accanto in ogni momento. Hai sempre creduto in me. Mi sei stato vicino quando ho iniziato il tirocinio e quando ho firmato il mio primo contratto. Grazie per aver migliorato tutte le mie giornate.

A Loris Cernich, il mio tutor aziendale che mi ha seguita durante tutto il percorso di formazione. Farò tesoro di ogni insegnamento.

A Giorgia, Anna e Linda. Se non avessi condiviso con voi l'esperienza a Dublino non sarei qui oggi.

Ai miei amici dell'Erasmus e a quelli dell'Università. Ai miei compagni di tirocino, Matteo, Simone e Giovanni con cui ho condiviso le migliori risate. A Lou, che è diventata una sorella. Ad Arianna, Francesca, Sofia e Melissa che, seppur distanti, hanno sempre trovato un momento per me.

Ai miei amici del terzo piano, Alessandra, Valeria, Babdi, Andrea, Marcello e Gigi che ci sono stati all'inizio di questo percorso e ai miei amici del primo piano, Serena, Elisa, Camilla e Pierpaolo che sono con me alla fine.

A me stessa, che non si è lasciata spaventare (non troppo comunque) dal non aver fatto *il giusto liceo* e ha scelto di studiare quello che le è sempre piaciuto

Appendice A

Listato

A.1 Front-end

A.1.1 Rendere i componenti sicuri: PrivateRoute.jsx

```
1 import React from "react";
2 import {Redirect, Route} from "react-router-dom";
3 import {isValid} from "../common/API/APIUtils";
4
5 const PrivateRoute = ({component: Component, path, onLogout,
6   ...rest}) => {
7   return (
8     <Route
9       exact
10      path={path}
11      {...rest}
12      render={(props) => {
13        return isValid() ? (
14          <Component {...props} onLogout={onLogout}/>
15        ) : (
16          <Redirect
17            to={{
18              pathname: "/",
19              state: {
20                prevLocation: path,
21                error: "You need to login
22                  first!",
23              },
24            }}>
25          />
26        );
27      }}
```

```
25          }}
26      />
27  );
28 };
29 export default PrivateRoute;
```

A.2 Back-end

A.2.1 Configurazione delle servlet: ServletConfiguration.java

```
1 package com.certimetergroup.qrestaurant.configuration;
2 @Configuration
3 @EnableMBeanExport(registration =
4     RegistrationPolicy.IGNORE_EXISTING)
5 public class ServletConfiguration {
6     @Bean
7     public ServletRegistrationBean<DispatcherServlet>
8         apiMobileCLIENT() {
9         DispatcherServlet dispatcherServlet =
10            new DispatcherServlet();
11         AnnotationConfigWebApplicationContext
12             applicationContext =
13                new AnnotationConfigWebApplicationContext();
14         applicationContext
15             .register(ApiMobileClientServletConfiguration.class);
16         dispatcherServlet
17             .setApplicationContext(applicationContext);
18         ServletRegistrationBean<DispatcherServlet>
19             servletRegistrationBean =
20                 new ServletRegistrationBean<DispatcherServlet>
21                     (dispatcherServlet, "/client/mobile/*");
22         servletRegistrationBean.setName("apiMobileCLIENT");
23         servletRegistrationBean.setLoadOnStartup(1);
24         servletRegistrationBean
25             .setMultipartConfig(
26                 new MultipartConfigElement("", 1024 *
27                     1024 * 10, 1024 * 1024 * 30, 0));
28         return servletRegistrationBean;
29     }
30     @Bean
31     public ServletRegistrationBean<DispatcherServlet>
32         apiMobileMANAGER() {
33         DispatcherServlet dispatcherServlet =
34            new DispatcherServlet();
35         AnnotationConfigWebApplicationContext
36             applicationContext =
37                new AnnotationConfigWebApplicationContext();
38         applicationContext
39             .register(ApiMobileManagerServletConfiguration.class);
40         dispatcherServlet
```

```

34             .setApplicationContext(applicationContext);
35         ServletRegistrationBean<DispatcherServlet>
36             servletRegistrationBean =
37                 new ServletRegistrationBean<DispatcherServlet>
38                     (dispatcherServlet,
39                         "/manager/mobile/*");
40         servletRegistrationBean.setName("apiMobileMANAGER");
41         servletRegistrationBean.setLoadOnStartup(1);
42         servletRegistrationBean
43             .setMultipartConfig(
44                 new MultipartConfigElement("", 1024 *
45                     1024 * 10, 1024 * 1024 * 30, 0));
46     return servletRegistrationBean;
47 }
48 @Bean
49 public ServletRegistrationBean<DispatcherServlet>
50     apiWebMANAGER() {
51     DispatcherServlet dispatcherServlet =
52         new DispatcherServlet();
53     AnnotationConfigWebApplicationContext
54         applicationContext =
55             new AnnotationConfigWebApplicationContext();
56     applicationContext
57         .register(ApiWebManagerServletConfiguration.class);
58     dispatcherServlet
59         .setApplicationContext(applicationContext);
60     ServletRegistrationBean<DispatcherServlet>
61         servletRegistrationBean =
62             new ServletRegistrationBean<DispatcherServlet>
63                 (dispatcherServlet, "/manager/web/*");
64     servletRegistrationBean.setName("apiWebMANAGER");
65     servletRegistrationBean.setLoadOnStartup(1);
66     servletRegistrationBean
67         .setMultipartConfig(
68             new MultipartConfigElement("", 1024 *
69                 1024 * 10, 1024 * 1024 * 30, 0));
70     return servletRegistrationBean;
71 }

```

A.2.2 Imbustamento della risposta: Response.java

1 package com.certimetergroup.qrestaurant.response;

```
2 public class Response {
3     private final int code;
4     private final String message;
5     private final Timestamp timestamp;
6     private final String path;
7
8     public Response(ResponseType response) {
9         this.code = response.getResponseCode();
10    this.message = response.getResponseMessage();
11    this.timestamp = new
12        Timestamp(System.currentTimeMillis());
13    this.path = PathUtility.getCurrentPath();
14 }
14 // getter e setter omessi
15 }
```

A.2.3 Body della risposta

```
1 {
2     "code": 0,
3     "message": "SUCCESS",
4     "timestamp": "2020-12-14T09:45:06.908+00:00",
5     "path": "/client/mobile/registration",
6     "registrationToken": "Rgy4NvcSt2Q3_1607939106891"
7 }
```

A.2.4 Gestione delle eccezioni: FailureException.class

```
1 package com.certimetergroup.qrestaurant.exception;
2 public class FailureException extends RuntimeException {
3     private final ResponseType responseType;
4     private final HttpStatus status;
5
6     public FailureException(ResponseType responseType,
7         HttpStatus status) {
8         super(responseType.getResponseMessage());
9         this.responseType = responseType;
10        this.status = status;
11    }
11 //getter e setter omessi
12 }
```

A.2.5 Configurazione di Spring Security: SecurityConfiguration.java

```
1 package com.certimetergroup.qrestaurant.security.configuration;
2 @Configuration
3 @EnableWebSecurity
4 public class SecurityConfiguration extends
5     WebSecurityConfigurerAdapter {
6     @Autowired
7     private AuthenticationEntryPoint authenticationEntryPoint;
8     @Autowired
9     private JWTAuthProvider jwtAuthProvider;
10    @Autowired
11    private TokenAuthProvider tokenAuthProvider;
12    @Autowired
13    @Override
14    protected void configure(AuthenticationManagerBuilder auth)
15        throws Exception {
16        /* SET PROVIDERS, EACH ONE FOR A SPECIFIC TOKEN */
17        auth
18            .authenticationProvider(jwtAuthProvider)
19            .authenticationProvider(tokenAuthProvider);
20    }
21    @Override
22    @Bean
23    public AuthenticationManager authenticationManagerBean()
24        throws Exception {
25        return super.authenticationManagerBean();
26    }
27    @Override
28    protected void configure(HttpSecurity http) throws
29        Exception {
30        http
31            .csrf()
32            .cors()
33            .disable()
34            .httpBasic().disable()
35            .authorizeRequests()
36            /* permitted endpoints */
37            .antMatchers(AUTHORIZED.toArray(new String[0]))
38            .permitAll()
39            /* any other request must be authenticated */
```

```
37     .anyRequest().authenticated()
38     .and()
39     .sessionManagement()
40     .sessionCreationPolicy
41             (SessionCreationPolicy.STATELESS)
42     .and()
43     /* add custom filter */
44     .addFilterBefore
45             (authenticationFilter,
46             UsernamePasswordAuthenticationFilter.class)
47     /* entrypoint in case of failure */
48     .exceptionHandling()
49     .authenticationEntryPoint
50             (authenticationEntryPoint);
51 }
52 }
```

A.2.6 Scelta dell'ambiente di sviluppo: EnvironmentConfiguration.java

```
1 package com.certimetergroup.qrestaurant.configuration;
2 @Configuration
3 @EnableMBeanExport(registration =
4     RegistrationPolicy.IGNORE_EXISTING)
5 @PropertySources({
6     @PropertySource("classpath:application.properties"),
7     @PropertySource(value =
8         "classpath:${env}/database.properties",
9         ignoreResourceNotFound = false),
10    @PropertySource(value =
11        "classpath:${env}/firebase.properties",
12        ignoreResourceNotFound = false),
13    @PropertySource(value =
14        "classpath:${env}/jwt_token.properties",
15        ignoreResourceNotFound = false),
16    @PropertySource(value =
17        "classpath:${env}/export.properties",
18        ignoreResourceNotFound = false),
19    @PropertySource(value =
20        "classpath:${env}/notification.properties",
21        ignoreResourceNotFound = false, encoding = "UTF-8"),
22    @PropertySource(value =
```

```
    "classpath:${env}/password_encoder.properties",
    ignoreResourceNotFound = false),
12 })
13 public class EnvironmentConfiguration {
14     final Logger logger =
            LoggerFactory.getLogger(this.getClass());
15     @Value("${env}")
16     private String env;
17     @PostConstruct
18     public void init() {
19         logger.info("ENVIRONMENT => " + env);
20     }
21 }
```

A.3 Deployment

A.3.1 DockerFile

```
1 FROM maven:latest as builder
2 COPY pom.xml /usr/local/pom.xml
3 COPY src /usr/local/src
4 WORKDIR /usr/local/
5 RUN mvn clean package
6
7 FROM openjdk:11
8 COPY --from=builder
    /usr/local/target/qrestaurant-0.0.1-SNAPSHOT.jar
    qrestaurant.jar
9 ENTRYPOINT exec java -jar /qrestaurant.jar
```

A.3.2 Docker Compose

```
1 version: "3.8"
2 volumes:
3   my-database:
4     name: my-database
5 networks:
6   qrestaurant-network:
7     name: qrestaurant-network
8     driver: bridge
9 services:
10 mysql:
11   image: mysql:latest
12   container_name: mysql
13   environment:
14     - MYSQL_ROOT_PASSWORD=root
15   ports:
16     - "3306:3306"
17   volumes:
18     - "my-database:/var/lib/mysql"
19   networks:
20     - qrestaurant-network
21   healthcheck:
22     test: [ "CMD", "curl", "-f", "http://mysql:3306" ]
23     interval: 1m30s
24     timeout: 10s
25     retries: 3
```

```
26      start_period: 40s
27 qrestaurant:
28   image: qrestaurant:latest
29   container_name: qrestaurant
30   depends_on:
31     - mysql
32   ports:
33     - "8080:8080"
34   networks:
35     - qrestaurant-network
36   build:
37     context: .
38   dockerfile: Dockerfile
39   healthcheck:
40     test: [ "CMD", "curl", "-f", "http://localhost:8080" ]
41     interval: 1m30s
42     timeout: 10s
43     retries: 3
44     start_period: 40s
```