



Università degli Studi di Milano Bicocca
**Dipartimento di Informatica, Sistemistica
e Comunicazione**
Corso di laurea in Informatica prova

Progettazione e sviluppo di un software come servizio: un caso pratico

Relatore: Prof. Pirola Yuri

Correlatore: Prof.ssa Rizzi Raffaella

Relazione della prova finale di:

Riccardo Aziani

Matricola 866037

Anno Accademico 2023-2024

*A mamma e papà, per avermi supportato economicamente e moralmente, ed
avermi spinto a proseguire nel mio percorso senza
preoccupazioni anche nei momenti più difficili.*

*A mia sorella, per aver sempre rappresentato un esempio da seguire
e da cui trarre ispirazione.*

*Ai nonni, per il loro sincero interesse e per avermi sempre ricordato
l'importanza dello studio, facendo in modo
che potessi metterlo sempre tra le mie priorità.*

*A nali, per essere stata indispensabile nei momenti più bui -e non solo-
con il mai mancato affetto, il continuo supporto e i numerosi consigli;
con le giornate passate a studiare assieme, spronandoci a vicenda,
hai reso i miei sforzi e sacrifici molto meno pesanti.*

*Agli amici che ho conosciuto durante questi anni, con cui ho condiviso momenti
che rimarranno impressi per sempre nei miei ricordi.*

*Grazie perché sono certo che questo traguardo senza di voi
non sarebbe stato possibile. Vi voglio bene.*

Sommario

Capitolo 1: Introduzione.....	4
Capitolo 2: Progettazione e sviluppo di software come servizio	5
2.1. Ingegneria dei requisiti	5
2.1.1. Storie e scenari	6
2.2. Progettazione della base di dati	7
2.2.1. Progettazione concettuale	7
2.2.2. Progettazione logica	8
2.3. Progettazione architetturale	9
2.3.1. Model-View-Controller.....	9
2.3.2. Java Servlet e i Presentation framework	10
2.4. Test del software	14
2.4.1. Sviluppo guidato da test	14
2.4.2. Test delle unità	15
2.4.3. Test dei componenti	15
2.4.4. Test del sistema	16
2.5. Deploy	17
2.5.1. Container	17
2.5.2. DevOps	18
2.5.3. Continuous Integration e Continuous Delivery	19
Capitolo 3: Esempio applicativo	20
3.1. Ingegneria dei requisiti	20
3.1.1. Storie e scenari, utente esterno	20
3.1.2. Storie e scenari, utente interno.....	30
3.2. Progettazione della base di dati	35
3.3. Progettazione architetturale	38
3.4. Test del software	40
3.5. Deploy	42
Capitolo 4: Bibliografia	43

Capitolo 1: Introduzione

La rapida diffusione della rete Internet è uno degli avvenimenti più significativi del settore dell'informatica. La tecnologia che più delle altre ha fatto registrare una crescita oltre ogni previsione è quella del World Wide Web: in principio, poteva essere definita come un sistema su scala planetaria per la distribuzione e l'accesso di documenti ipertestuali; successivamente, è diventata una piattaforma per lo sviluppo di sistemi informativi di ogni genere accessibili tramite un browser. È infatti con l'avvento di questa tecnologia, e la diffusione del calcolo cloud, che si è sviluppato il concetto di software come servizio (SaaS).

Per SaaS si intende il software offerto come servizio, a cui è possibile accedere tramite browser. Questo concetto porta con sé i seguenti elementi chiave:

- Il software è installato in un server, a cui si può accedere via web.
- Il software è gestito dal fornitore del software stesso, che ne è anche il proprietario.
- Gli utenti pagano il software in funzione dell'uso che ne fanno, o tramite una sottoscrizione mensile/annuale. In alternativa, può accadere che il software sia gratuito a condizione che l'utente accetti la visione di messaggi pubblicitari.

Nel corso della relazione verranno trattati i principali aspetti della progettazione e sviluppo di una applicazione concepita come SaaS, un processo che ho affrontato durante lo stage esterno in azienda. Durante questo periodo, ho seguito un corso formativo sull'utilizzo di Struts, uno dei principali framework MVC per lo sviluppo di applicazioni web Java.

Le competenze acquisite durante il percorso di studi universitari, unite alle nozioni tecniche apprese durante il corso di formazione, mi hanno permesso di sviluppare un progetto personale: lo sviluppo di un'applicazione per la gestione di un sistema di prenotazioni. Questo progetto ha rappresentato una sfida interessante e significativa, che mi ha dato possibilità di “tastare con mano” alcune tra le conoscenze acquisite, approfondendone alcuni aspetti.

La relazione è strutturata nel seguente modo:

- Nel Capitolo 2 vengono trattate le principali fasi che compongono la progettazione e lo sviluppo di un SaaS, tra cui:
 - ❖ Ingegneria dei requisiti, nella sezione 1
 - ❖ Progettazione della base di dati, nella sezione 2.2
 - ❖ Progettazione architetturale, nella sezione 2.3
 - ❖ Test del software, nella sezione 2.4
 - ❖ Deploy, nella sezione 2.5
- Nel Capitolo 3 viene utilizzato il progetto personale come caso di studio per trattare le tematiche del Capitolo 2.

Capitolo 2: Progettazione e sviluppo di software come servizio

L'ingegneria del software è una disciplina che si occupa di tutti gli aspetti della produzione di un software, dalla nascita, al funzionamento e alla manutenzione.

In questo capitolo vengono trattate le principali fasi che compongono il ciclo di vita di un sistema informativo, tra cui:

- **Ingegneria dei requisiti**, nella sezione 1
- **Progettazione della base di dati**, nella sezione 2.2
- **Progettazione architetturale**, nella sezione 2.3
- **Test del software**, nella sezione 2.4
- **Deploy**, nella sezione 2.5

Per ciascuna di queste fasi cercheremo di mettere in risalto, dove siano presenti, le attività che sono influenzate dal contesto web.

2.1. Ingegneria dei requisiti

L'ingegneria dei requisiti è la prima attività che viene svolta durante un processo di progettazione e sviluppo di un sistema. I requisiti di un sistema sono la descrizione dei servizi che deve fornire e i loro vincoli operativi, mentre il processo di ricerca, analisi, documentazione e verifica di questi servizi e vincoli è chiamato ingegneria dei requisiti [1].

Il termine “requisito” nell'ambito dell'ingegneria del software non è sempre usato in modo coerente: a volte potrebbe riferirsi a una formulazione astratta e di alto livello di ciò che dovrebbe fare un sistema; altre volte a una descrizione formale e dettagliata di una specifica funzione del sistema. È possibile fare una distinzione fra queste tipologie di descrizione, facendo riferimento alla fonte [1]:

- **Requisiti dell'utente:** dichiarano, con linguaggio naturale ad alto livello, quali servizi dovrebbe offrire il sistema
- **Requisiti del sistema:** sono descrizioni più dettagliate delle funzioni, dovrebbero definire esattamente il software che deve essere implementato

Anche se queste diverse definizioni del termine requisito possano sembrare a primo impatto confusionarie, si rivelano utili per fornire documenti ai diversi tipi di lettori con informazioni a diversi livelli di astrazione.

Nell'ambito web, l'attività di ingegneria dei requisiti pone attenzione sull'interazione dell'utente con il sistema; è importante individuare dei gruppi di utenti, ciascuno caratterizzato da esigenze simili. I contenuti memorizzati nella base di dati saranno mostrati in maniera differente in base alle esigenze di ciascun gruppo.

Un altro tema su cui è necessario riporre particolare attenzione quando si progetta un SaaS è l'interfaccia. Bisogna tenere conto che il sistema potrà essere utilizzato su

dispositivi molto diversi tra loro: ad esempio un PC fisso, un laptop o un cellulare; è importante fare in modo che l'interfaccia si adatti a ciascuno di essi in relazione alla dimensione dello schermo.

2.1.1. Storie e scenari

Dato che descrivere in modo dettagliato ed esaustivo i requisiti può essere un'operazione difficile, un modo efficace per definirli è quello di utilizzare le storie e gli scenari.

Le storie sono dei testi scritti in linguaggio naturale; utilizzando un esempio di vita reale, forniscono una descrizione ad alto livello di cosa deve fare il sistema, come si comporta in determinate situazioni, quali input prende e quali output produce, illustrando come l'utente può interagire con il sistema.

Una singola storia può essere approfondita in una serie di scenari più specifici. Gli scenari differiscono dalle storie perché seguono una struttura dettagliata, e forniscono informazioni più approfondite sui diversi casi di interazione tra il sistema e l'utente.

Le storie sono adatte per fornire una visione generale del sistema e su cosa deve compiere, mentre gli scenari sono in grado di fornire i dettagli necessari per capire come deve essere realizzato il sistema. [1]

2.2. Progettazione della base di dati

La progettazione di una base di dati è un processo che ha l'importante compito di rendere accessibile in maniera efficiente le informazioni necessarie al corretto funzionamento del sistema. Nonostante la base di dati sia soltanto uno dei vari componenti di un'applicazione, il ruolo centrale che risiede nei dati all'interno di un sistema, specialmente in ambito web, è tale da giustificare uno studio dedicato al modo in cui vogliamo rappresentarli; per fare ciò, in questa sezione del capitolo faremo affidamento alla fonte [2] per approfondire gli aspetti rilevanti della progettazione di basi di dati.

Nel corso degli anni si è consolidata una metodologia di progettazione fondata sul principio ingegneristico tanto semplice quanto efficace di separare le decisioni relative a “cosa” rappresentare in una base di dati (progettazione concettuale), da quelle relative a “come” farlo (progettazione logica).

2.2.1. Progettazione concettuale

La progettazione concettuale ha lo scopo di rappresentare il contenuto informativo dei dati, senza preoccuparsi dei vincoli implementativi. Il progettista deve soltanto occuparsi di rappresentare le informazioni, senza pensare a come effettivamente saranno implementati e codificati in un sistema reale, né all'efficienza.

La progettazione concettuale prende in input una specifica dei requisiti del sistema; è possibile fare una distinzione tra specifiche dei dati e specifiche delle operazioni. In questa fase bisogna fare attenzione alle specifiche sui dati, mentre le specifiche sulle operazioni servono a controllare che lo schema prodotto sia completo e che siano presenti tutti i dati necessari ad implementare le operazioni.

Il prodotto della progettazione concettuale è un diagramma entità-relazione: è un modello di rappresentazione concettuale dei dati che utilizza una serie di costrutti per rappresentare i dati senza preoccuparsi che questi siano comprensibili da una macchina (a differenza di quanto avviene quando viene definito uno schema con un linguaggio come SQL). Ne risulta uno schema che ha l'obiettivo di descrivere al meglio le specifiche sui dati dell'applicazione che si sta progettando.

Ci sono quattro proprietà che un buono schema concettuale dovrebbe avere:

- **Correttezza:** uno schema concettuale si dice corretto quando utilizza correttamente i costrutti messi a disposizione dal modello concettuale di riferimento.
- **Completezza:** uno schema concettuale è completo quando rappresenta tutti i dati di interesse e quando tutte le operazioni possono essere eseguite a partire dai concetti descritti nello schema.

- **Leggibilità:** uno schema concettuale si dice leggibile quando rappresenta i requisiti in maniera naturale e comprensibile; per fare ciò è consigliato assegnare dei nomi opportuni ai concetti e limitare le intersezioni.
- **Minimalità:** uno schema concettuale è minimale quando non sono presenti ridondanze e tutti i concetti sono rappresentati una sola volta nello schema.

2.2.2. Progettazione logica

La progettazione logica prende in input lo schema E-R prodotto nella fase precedente, con l'obiettivo di tradurlo in termini del modello di rappresentazione dei dati adottato dal sistema di gestione di basi di dati a disposizione. Il prodotto di questa fase viene chiamato schema logico fa riferimento a un modello logico dei dati; questa rappresentazione dei dati è ancora indipendente da dettagli di implementazione fisica, ma è concreta perché disponibile nei sistemi di gestione di dati di basi.

È possibile dividere la progettazione logica in due fasi distinte: la prima fase è quella di semplificazione dello schema E-R per rendere più facile la fase successiva; inoltre, è possibile che certe modifiche siano necessarie perché non tutti i costrutti del modello E-R hanno una naturale traduzione nel modello logico. La seconda fase è quella di traduzione verso il modello logico.

Inoltre, possono rendersi necessarie altre modifiche, perché a differenza della fase precedente in cui si ha il solo obiettivo di rappresentare in maniera esaustiva il significato dei dati, adesso viene presa in considerazione anche la prestazione. Lo schema logico rappresenta infatti la base di partenza per la vera e propria implementazione della base di dati; dunque, si ragiona anche in termini di efficienza delle transazioni.

2.3. Progettazione architetturale

La progettazione architetturale è un processo che si occupa dell'organizzazione di un sistema software e della progettazione della sua struttura complessiva. Identifica i principali componenti strutturali del sistema e loro relazioni, producendo in output un modello architetturale che descrive come il sistema è organizzato.

I singoli componenti di un sistema implementano i requisiti funzionali, ma è l'architettura che mettendoli in relazione ha l'influenza predominante sui requisiti non funzionali [1].

2.3.1. Model-View-Controller

Negli anni '90 nascono gli schemi architetturali come modo per rappresentare, condividere e riutilizzare le informazioni dei sistemi software. Possono essere immaginati come una descrizione stilizzata di una buona pratica, che è stata provata e verificata in altri sistemi e ambienti; dovrebbe includere informazioni su quando è opportuno è usarlo, e quali i sono i suoi vantaggi e svantaggi.

Lo schema MVC è la base per la gestione delle interazioni nei sistemi web ed è supportato da molti linguaggi; questi sono i suoi punti chiave, tratti dalla fonte [1]:

- **Descrizione:** ripartizione del sistema in tre moduli: il *Model* rappresenta lo stato dell'applicazione, sono i dati che vengono elaborati e presentati all'utente; la *View* si occupa di presentare all'utente i dati del Model e di fornire i comandi per l'interazione; il *Controller* traduce le interazioni dell'utente sui dati del Model e si occupa di mostrare la vista appropriata.
- **Quando si usa:** quando ci sono più modi di rappresentare i dati in base ad esigenze diverse; si usa anche quando non sono noti i requisiti futuri di interazione e presentazione dei dati.
- **Vantaggi:** la rappresentazione dei dati è indipendente dai dati stessi: questo significa che si possono apportare modifiche alla logica di gestione dei dati senza intaccare la loro presentazione e viceversa; sono supportati più modalità di rappresentazione dei dati senza che sia necessario modificarli. Le azioni dell'utente sono trattate esclusivamente dal controller che si occupa di formulare la risposta.
- **Svantaggi:** potrebbe richiedere del codice complesso anche quando le dati e le loro interazioni sono semplici.

Nella Figura 2.1 è rappresentata una vista concettuale del pattern architetturale MVC.

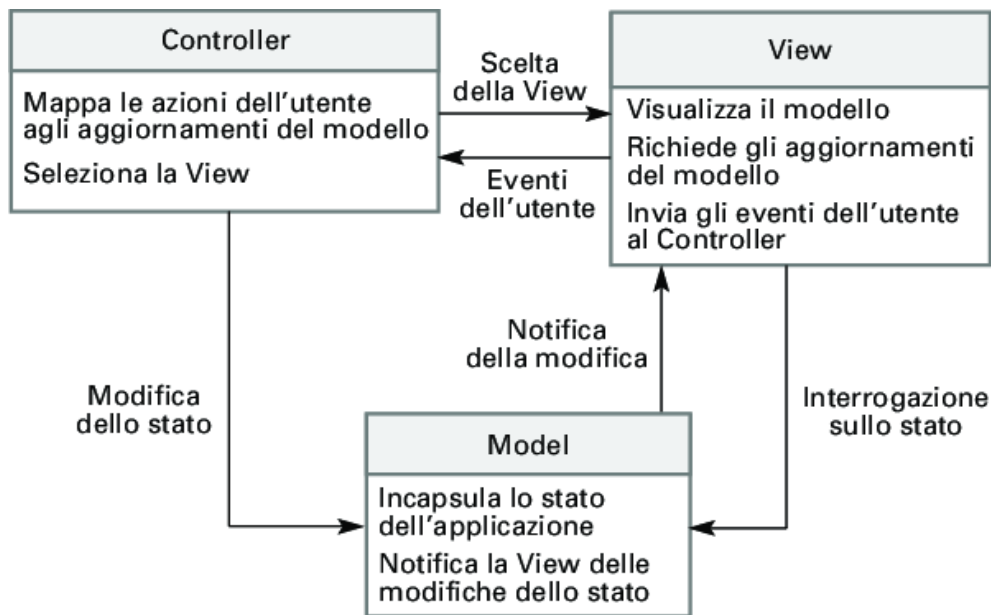


Figura 2.1 Organizzazione dello schema Model-View-Controller [1].

2.3.2. Java Servlet e i Presentation framework

In questa sezione del paragrafo viene trattata l'architettura Java Servlet e due framework che offrono delle soluzioni differenti per rispondere alla stessa esigenza, facendo affidamento alla fonte [2].

I presentation framework basati sull'architettura MVC sono nati per facilitare le operazioni di sviluppo ai programmatori. Il termine framework denota un insieme di moduli software; il termine presentation fa riferimento al fatto che tali moduli servono per la costruzione dinamica delle pagine da presentare all'utente.

Una delle tecnologie più diffuse tra i framework e usata per estendere il server web è l'architettura Java Servlet, nata per far fronte al fatto che il protocollo HTTP non ha memoria (stateless): il server non è in grado di recuperare informazioni sulle operazioni già effettuate.

Il vantaggio di questa architettura è l'utilizzo, ai fini del calcolo dinamico delle pagine, dell'ambiente di esecuzione di programmi Java chiamato *Java Virtual Machine*.

Il cuore dell'architettura è il *Servlet Container*, un programma speciale che espone una serie di funzionalità e oggetti come *request* e *response*, che incapsulano i dati della relativa richiesta e risposta HTTP, o l'oggetto *session* che funge da memoria condivisa per richieste dello stesso utente; queste funzionalità sono sfruttate dalle Servlet Java, che sono in grado di generare in modo dinamico una pagina per l'utente al ricevimento di una richiesta GET.

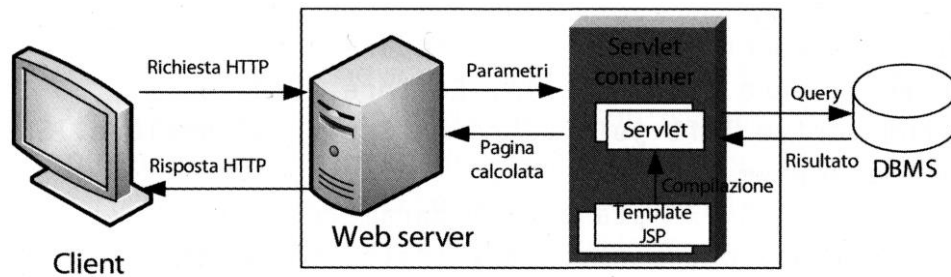


Figura 2.2 Architettura Servlet con JSP [2].

Una tecnologia che si unisce alle Servlet è quella delle Java Server Pages (JSP) : è una fusione di parti di codice HTML e Java, che facilita la scrittura di una pagina che viene generata dinamicamente: questo rende molto più netta la separazione tra la presentazione dei dati dalla logica e il calcolo. Quando un client richiede una pagina JSP, essa viene tradotta in una Servlet equivalente. Nella Figura 2.2 è rappresentata una vista di un sistema con architettura Java Servlet e Java Server Pages.

Uno dei primi framework che ha avuto un'ampia diffusione è stato Struts, prodotto dall'Apache Software Foundation. Struts può essere descritto come un'implementazione avanzata del Controller per la piattaforma Java Servlet, utilizzabile con oggetti del Model realizzati in Java. Il generatore delle richieste è il browser: quando l'utente invia una richiesta tramite un collegamento ipertestuale o compilando un form per l'inserimento di dati, la Servlet di Struts riceve la richiesta e provvede a popolare l'ActionForm associato ad essa, ovvero un contenitore di dati (dichiarato in precedenza) che viene riempito automaticamente dal framework. Una volta riempito l'ActionForm con i dati della richiesta, l'esecuzione viene delegata alla relativa Action; svolge la logica di business, interagendo con il Model, e poi restituisce alla Servlet di Struts un ActionForward contenente il path della vista da mostrare all'utente.

L'aspetto più rilevante del controller Struts è il suo carattere di natura parametrica: quale Action deve essere invocata per ogni richiesta, quale ActionForm deve essere popolato, e quale ActionForward deve fornire il path sono informazioni dichiarate in un file XML: questo rende possibile modificare il comportamento del Controller senza preoccuparsi delle pagine JSP o della logica di accesso ed elaborazione del Model. Una vista dell'architettura del framework Struts è mostrata in Figura 2.3.

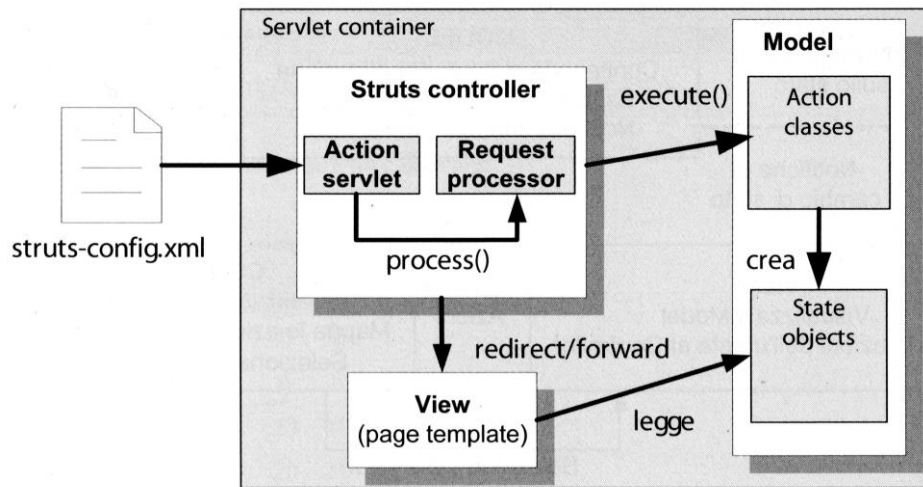


Figura 2.3 L'architettura di Apache Struts [2].

Una seconda realizzazione di MVC per il Web a lato server è fornita dal framework Spring MVC. Come mostrato in Figura 2.4, differisce da Struts per diversi motivi:

- Fa parte di un sistema complesso che comprende anche moduli per il controllo della sicurezza, gestione delle transazioni, accesso ai dati e altro ancora. Il funzionamento è basato sull'*Inversion of Control*, un meccanismo che prevede che un oggetto che ha bisogno di una risorsa dichiari una dipendenza anziché riferirsi direttamente ad essa; in questo i componenti sono resi maggiormente indipendenti e riusabili.
- Offre un meccanismo diverso per la realizzazione dei controller e il loro collegamento alle funzioni applicative: anziché programmare classi specifiche come le Action di Struts e definire regole di mappatura tra le richieste HTTP e le Action, il programmatore può arricchire direttamente le proprie classi applicative mediante il meccanismo delle annotazioni Java; queste direttive specificano quale metodo deve essere invocato al ricevimento di una richiesta HTTP avente uno specifico formato, e quale trattamento devono avere i suoi parametri.
- Spring MVC usa una serie di oggetti predefiniti per gestire in modo automatico la comunicazione tra i vari componenti del framework, come il passaggio dei parametri dalla richiesta HTTP al controller e agli oggetti del Model, o la notifica da parte del Controller sulla scelta della view da usare per produrre la risposta.

Struts e Spring MVC sono due soluzioni alternative per rispondere alla stessa esigenza: migliorare l'organizzazione, la manutenzione e il riutilizzo del software facilitando i compiti dello sviluppatore.

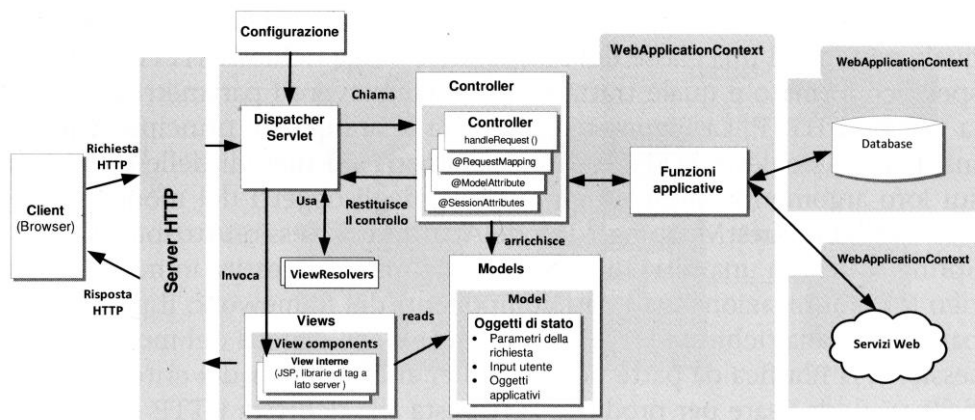


Figura 2.4 L'architettura di Spring MVC [2].

2.4. Test del software

L'obiettivo del test del software è quello di verificare che il sistema progettato soddisfi tutti i requisiti, trovando eventuali errori prima che il sistema venga reso disponibile al cliente.

In un caso ideale, vengono fatti dei test per ogni possibile input o condizione di lavoro; ovviamente fare dei test totalmente esaustivi, come ad esempio tutti i possibili input, non è possibile. Il testing può infatti mostrare la presenza di errori nel sistema, ma non può assicurare la loro assenza [3].

L'obiettivo del testing è, sapendo già quante saranno le risorse dedicate a questo processo dell'ingegneria del software, cercare di effettuare dei test il più possibile esaustivi e che possano garantire che il software è pronto per il rilascio.

2.4.1. Sviluppo guidato da test

Lo sviluppo guidato da test è un approccio in cui si interlacciano le attività di sviluppo e il testing del codice [4]. Il codice viene sviluppato in modo incrementale, accoppiato ad una serie di casi di test: non è possibile iniziare una nuova fase di sviluppo di codice fino a quando tutti i test non vengono superati. Questa metodologia nasce come parte del metodo di sviluppo agile, ed oggi è affermata e adottata da molti sviluppatori.

Le fasi principali dello sviluppo guidato da test, mostrate in Figura 2.5, sono:

- Identificazione del nuovo incremento; deve essere piccolo e sviluppabile in un numero contenuto di righe di codice.
- Scrittura del test per verificare se l'incremento viene implementato correttamente.
- Esecuzione di tutti i test. Può sembrare inutile verificare il nuovo test che ovviamente darà un esito negativo; tuttavia, in questo modo si dimostra che effettivamente viene aggiunta una nuova funzionalità al sistema.
- Si scrive il codice per implementare la nuova funzionalità, e poi si ripetono i test.
- Quando tutti i test sono soddisfatti, si può passare ad un nuovo incremento del sistema.

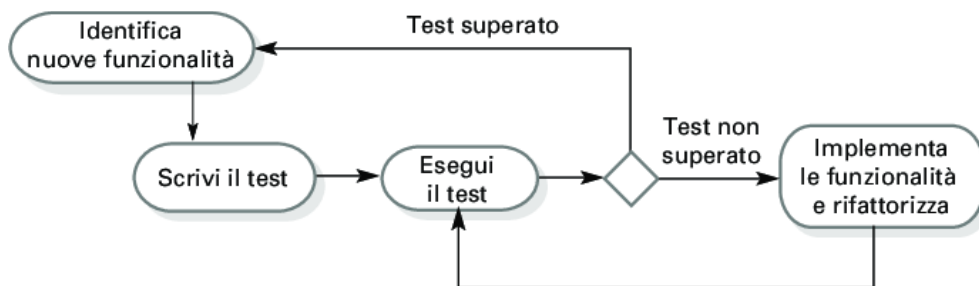


Figura 2.5 Sviluppo guidato da test.

Lo sviluppo guidato da test porta con sé una serie di vantaggi [1]:

- **Copertura del codice:** ogni riga di codice è coperta da almeno un caso di test, dato che ad ogni incremento (quindi ad ogni nuova scrittura di codice) viene implementato almeno un caso di test per verificarne la correttezza.
- **Test di regressione:** dato che i test sono scritti in maniera incrementale, è sempre possibile eseguire tutti i test per verificare che le nuove funzionalità non abbiano introdotto dei bug nel sistema.
- **Debugging:** quando un test fallisce, dovrebbe essere chiaro quale porzione di codice abbia introdotto l'errore.
- **Documentazione del sistema:** i test sono una forma di documentazione che descrive cosa dovrebbe fare il sistema; inoltre, la lettura dei test può agevolare la comprensione del codice.

Grazie a questi benefici, oggi lo sviluppo guidato da test è una pratica largamente utilizzata nel testing del software.

2.4.2. Test delle unità

Il test delle unità è la prima fase del testing di un software. Per unità si intende che si verifica il funzionamento delle componenti più semplici di un programma; in questa fase si devono testare tutti i metodi di un oggetto e tutti i suoi possibili stati.

Per la progettazione dei casi di test sono state definite delle linee guida da seguire [5]:

- Scegliere input che forzano il sistema a generare tutti i messaggi di errore.
- Ripetere lo stesso input o la stessa serie di input numerose volte.
- Forzare la produzione di output non validi.
- Forzare i calcoli in modo da ottenere valori molto grandi o molto piccoli.

2.4.3. Test dei componenti

Il test dei componenti va a verificare che le singole unità del sistema funzionano correttamente quando interagiscono tra di loro.

I test case in questa fase non si applicano alle singole unità, ma ai componenti creati combinando le singole unità. Per effettuare questa tipologia di test, si assume che i test delle unità prese in considerazione siano completati.

Alcuni esempi comuni di casi di test [1] sono quando i dati vengono passati da una funzione ad un'altra, casi in cui un componente incapsula una serie di procedure chiamate da altri componenti, casi in cui un componente chiede un servizio ad un altro componente passandogli un messaggio.

2.4.4. Test del sistema

Il test del sistema è l'ultima fase del testing di un software: viene testato il prodotto finale come un unico componente, con l'obiettivo di verifica che tutte le sue parti funzionino correttamente quando interagiscono tra di loro.

2.5. Deploy

Con il termine deploy si intende il processo attraverso cui si mette in esecuzione un'applicazione. In questo capitolo viene trattato il concetto di Container, una delle tecnologie più diffuse per la distribuzione di software, e le pratiche di DevOps.

2.5.1. Container

I container sono unità eseguibili di software in cui il codice dell'applicazione viene impacchettato insieme alle sue librerie e dipendenze, in modo che il codice possa essere eseguito ovunque, dal desktop del proprio PC al cloud; utilizzano al meglio una forma di virtualizzazione del sistema operativo, in cui le funzionalità kernel vengono utilizzate per isolare i processi e controllare la quantità di CPU, memoria e disco a cui tali processi possono accedere. [6]

Per capire meglio cos'è un container, possiamo fare un confronto con le tradizionali Virtual Machines (a cui d'ora in poi faremo riferimento con il termine VMs), facendo affidamento alla fonte [7].

Una prima differenza riguarda il modo in cui queste due tecnologie implementano la virtualizzazione. Le VMs ottengono una virtualizzazione a livello hardware; come mostrato in **Error! Reference source not found.**, al di sopra dell'hardware e del S.O. si trova l'hypervisor: colui che è responsabile della creazione di istanze virtuali di tutto ciò che compone una macchina, come processori, RAM, memoria, e così via.

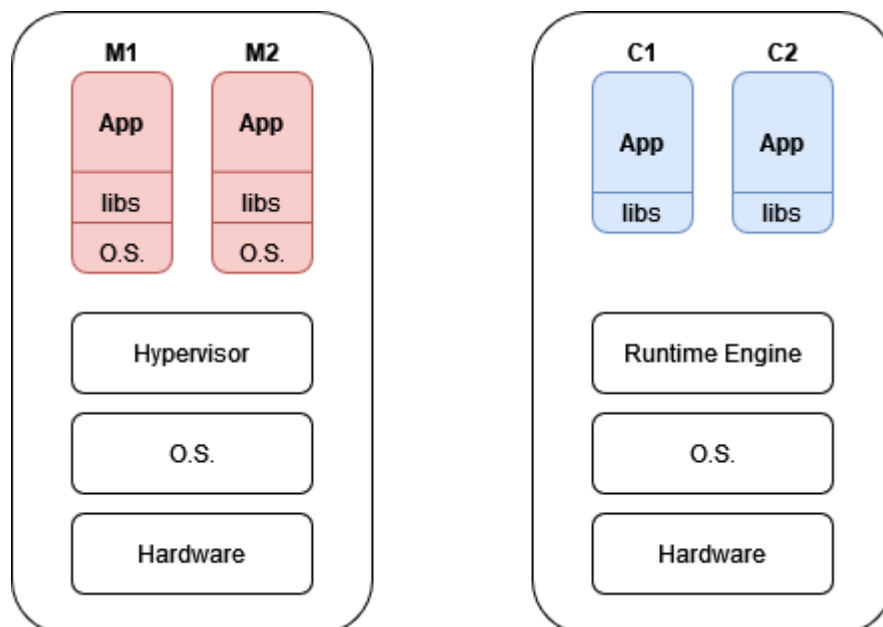


Figura 2.6 Virtual Machine e Container a confronto.

Nei container, invece, al di sopra dell'hardware e del S.O., troviamo un motore runtime: colui che si occupa di creare i container che sono in esecuzione; per un singolo sistema operativo, possono essere eseguiti molteplici container: per questo possiamo definire questa implementazione come una virtualizzazione a livello del sistema operativo.

Un'altra differenza è la tipologia di isolamento che si ottiene attraverso queste due tecnologie. Con le VMs otteniamo un isolamento a livello di macchina, ovvero otteniamo macchine (relativamente) indipendenti le une dalle altre. Invece, con i container, otteniamo un isolamento a livello di processo: essi condividono lo stesso sistema operativo, lo stesso kernel, ma ad ognuno di essi appare come se abbia il proprio sistema operativo; in questo modo, possono essere eseguiti contemporaneamente ed ogni container non conosce nulla degli altri, ottenendo un isolamento dei processi.

Alla luce di queste osservazioni, appare evidente che l'utilizzo dei container porta con sé diversi vantaggi, tra cui [6]:

- **Leggerezza:** dato che condividono il kernel del sistema operativo che fa da host, non è necessario avere un'istanza completa del sistema operativo per ogni applicazione; questo migliora le prestazioni in termini di risorse.
- **Portabilità:** i container contengono tutte le dipendenze di cui l'applicazione necessita, rendendo l'esecuzione del software indipendente dalla piattaforma.
- **DevOps:** supportano le pratiche di Continuous Integration e Continuous Delivery.

2.5.2. DevOps

La tecnologia dei container supporta DevOps, ovvero la combinazione di due pratiche che generalmente vengono trattate separatamente: lo sviluppo e le operazioni. Possiamo immaginare DevOps come una serie di processi di sviluppo iterativi e automatizzati, inseriti all'interno di un più ampio ciclo di sviluppo automatizzato per ottimizzare la rapida fornitura di software di alta qualità. Alcuni tra i principali flussi di lavoro sono [8]:

- **Progettazione:** viene valutato il potenziale delle nuove funzionalità nella release successiva, basandosi su casi di studio e feedback degli utenti.
- **Sviluppo:** fase di programmazione in cui gli sviluppatori testano, codificano e creano funzioni nuove e migliorate.
- **Build:** le pratiche di automazione più comuni includono l'incorporazione delle modifiche in una copia "master" del codice, l'estrazione di tale codice da un repository, e l'automazione di compilazione, test unitari e impacchettamento in un ambiente eseguibile.
- **Deploy:** distribuzione in un ambiente eseguibile.

- **Monitoraggio:** vengono monitorate le prestazioni e il comportamento delle nuove funzioni, assicurandosi che non presentino alcun problema e che non ci siano interruzioni nel servizio.
- **Apprendimento:** raccolta dei feedback che saranno utilizzati per la pianificazione delle funzionalità della release successiva.

Quando si parla di DevOps, assumono particolare importanza i concetti di CI (Continuous Integration) e CD (Continuous Delivery), che verranno trattati nella prossima sezione del capitolo.

2.5.3. Continuous Integration e Continuous Delivery

La pratica di CI si colloca tra la fase di sviluppo e di build descritte sopra; per comprendere cosa si intende per CI e la sua importanza, possiamo fare riferimento a quale problema si andava incontro prima che venisse adottata come pratica per produrre software di qualità: supponiamo che due (o più) sviluppatori stiano lavorando a degli aggiornamenti di una applicazione, lavorando separatamente per un lungo periodo di tempo; alcune righe di codice verranno modificate o eliminate da ciascuno dei due sviluppatori. Nel momento in cui dovranno integrare le loro modifiche, è molto probabile che riscontreranno dei conflitti di *merge*, che possono richiedere fino a giorni di lavoro per essere risolti.

Con la pratica di CI si intende un processo di sviluppo software in cui gli sviluppatori integrano il nuovo codice su base giornaliera; in questo modo, nel caso in cui si verificano dei conflitti di merge, essi riguardano soltanto il codice appena scritto e saranno più semplici da risolvere. Per individuare eventuali problemi di integrazione, ad ogni nuova build (effettuata ad ogni modifica al codice) vengono automatizzati una serie di test. [9]

Nel ciclo DevOps dove termina la pratica di CI subentra quella di CD; quest'ultima si concentra sulla fornitura agli utenti delle eventuali modifiche al codice, automatizzandone il processo. Questa pratica porta con sé una serie di vantaggi, tra cui:

- **Rilascio più rapido delle funzionalità**, dato che il processo automatizzato.
- **Maggiore affidabilità dei rilasci**, dato che nel processo di automatizzazione sono inclusi una serie di test.
- **Feedback immediato e continuo degli utenti**.

CI e CD sono dunque delle pratiche da adottare per migliorare la qualità del software, ridurre i rischi e i costi di integrazione ed accelerare il rilascio delle funzionalità.

Capitolo 3: Esempio applicativo

In questo capitolo vengono trattate le fasi di progettazione e sviluppo di un sistema già viste nel Capitolo precedente, trattandole utilizzando come caso di studio un sistema per gestire delle prenotazioni a dei campi da calcio.

Nello specifico, vengono trattati:

- **Ingegneria dei requisiti**, nella sezione 3.1
- **Progettazione della base di dati**, nella sezione 3.2
- **Progettazione architetturale**, nella sezione 3.3
- **Test del software**, nella sezione 3.4
- **Deploy**, nella sezione 3.5

3.1. Ingegneria dei requisiti

Quando si parla di applicazioni web, una delle prime attività da svolgere è l'individuazione dei gruppi di utenti: ciascun gruppo è caratterizzato da esigenze applicative simili e sarà dunque associato ad una specifica "versione" dell'applicazione; questa divisione in gruppi di utenti permette a ciascuno di essi di visualizzare i dati nella maniera corretta, e di effettuare le operazioni ad esso associate.

Nella progettazione dei requisiti del sistema sono stati individuati due gruppi di utenti, spesso presenti in ambito web: utenti esterni ed utenti interni. Gli utenti esterni sono i fruitori del servizio offerto dal sistema, mentre al contrario gli utenti interni rappresentano i gestori del sistema.

Per fare una distinzione tra quale utente sta utilizzando il sistema e per fornire la corretta vista dei dati e delle funzionalità, come spesso accade gli utenti interni devono superare un controllo di autenticazione.

Nelle prossime sezioni del capitolo segue la specifica dei requisiti, stilata seguendo la metodologia delle storie e scenari. Trattandosi di un sistema semplice e privo di criticità, è stata adottata questa scelta per spiegare in modo semplice quali servizi deve offrire il sistema, e come si deve comportare in determinate situazioni; inoltre, penso sia più gradevole ed intuitivo per il lettore comprendere la specifica dei requisiti.

3.1.1. Storie e scenari, utente esterno

In questa sezione sono presenti due storie, approfondite nei rispettivi scenari, per spiegare quali sono i servizi che il sistema deve offrire ai fruitori: effettuare una nuova prenotazione e gestire la propria prenotazione.

Requisiti utente esterno

Prenotazione di un campo di calcio

Riccardo vuole prenotare un campo da calcio per giocare una partita insieme ai suoi amici nel centro sportivo del suo paese, una struttura con diversi campi da calcio.

La compagnia ha deciso di giocare nel fine settimana: se ci fosse disponibilità il sabato vogliono giocare 12 amici, mentre nel caso in cui nella giornata di sabato non ci sia disponibilità, allora la domenica vorrebbero giocare 10 amici.

Riccardo prima controlla se è disponibile un campo per 12 persone nella giornata di sabato: si collega al sito web e seleziona il campo per 12 persone durante la fase di scelta del campo; successivamente, scorre i giorni della settimana fino ad arrivare a sabato, che però risulta occupato.

Riccardo a questo punto torna indietro, sceglie il campo per 10 persone e scorre fino alla giornata di domenica, dove trova disponibilità.

Per prenotare il campo, dopo aver cliccato sull'orario che preferisce, inserisce i dati che gli sono richiesti e conferma la prenotazione.

Per avere un promemoria dei dati della prenotazione, gli viene inviata una mail automatica al suo indirizzo di posta elettronica.

La storia viene approfondita in una serie di scenari; per ciascuno di essi, è fornita una schermata del sistema.

Scenario scelta del campo

Ipotesi iniziale: un utente vuole prenotare un campo. Si è collegato con successo al sistema e ha cliccato sul tasto "Prenota".

Normale: il sistema mostra all'utente una schermata con tanti bottoni quanti i sono i campi del gestore. Per ciascuno di essi è riportato il nome del campo e il numero di giocatori. L'utente può scegliere il campo che preferisce cliccando su uno dei bottoni.

Stato del sistema al completamento: dopo che l'utente ha scelto il campo che preferisce, gli viene mostrata la relativa pagina della disponibilità.



Figura 3.1 Schermata per lo scenario della scelta del campo.

Scenario scelta dell'orario

Ipotesi iniziale: un utente, dopo aver scelto il campo in cui giocare, vuole selezionare l'orario che preferisce per assicurarsi la prenotazione.

Normale: il sistema mostra all'utente una schermata con una tabella composta da tre colonne. La prima colonna rappresenta il giorno corrente, mentre le due successive rappresentano i due giorni successivi.

La prima riga viene riempita con la rispettiva data e giorno; quelle seguenti partono con il primo orario disponibile tra i 3 giorni e terminano con l'ultimo disponibile tra i 3 giorni. È possibile prenotare solo al minuto hh:00. Ciascuna riga rappresenta una fascia oraria.

Ciascuno slot orario è mostrato in una delle seguenti opzioni: verde se prenotabile e libero, rosso se prenotabile ma occupato, grigio se non prenotabile.

È possibile selezionare solo gli slot verdi, cliccandoci sopra.

A destra (e sinistra) della tabella sono posizionate delle frecce per scorrere avanti (e indietro) i giorni del calendario; se la tabella è posizionata sul giorno corrente, non è possibile scorrere ulteriormente indietro.

Stato del sistema al completamento: quando l'utente clicca su uno degli slot verdi, gli viene mostrato un form per raccogliere i dati necessari alla prenotazione.



Figura 3.2 Schermata per lo scenario della scelta dell'orario.

Scenario inserimento dei dati

Ipotesi iniziale: un utente ha scelto l'orario e deve inserire i dati per confermare la prenotazione.

Normale: il sistema mostra all'utente un form per l'inserimento dei dati necessari per la conferma della prenotazione: nome, cognome, e-mail e numero di telefono. Quando l'utente inserisce i dati, clicca su "conferma" per registrare la sua prenotazione e ricevere una mail al suo indirizzo come promemoria.

Che cosa può andare male: l'utente inserisce degli input non validi; in questo caso viene mostrato un messaggio all'utente per invitarlo a inserire correttamente i suoi dati.

Stato del sistema al completamento: dopo che l'utente ha confermato l'inserimento dei dati, si passa alla procedura di verifica e conferma della prenotazione.

Home

Progress indicator: 4 circles, 3rd circle filled.

Inserisci i dati richiesti per confermare la prenotazione

Form fields:

- Nome*
Nome
- Cognome*
Cognome
- Email*
email@email.com
- Telefono*
123456789

Conferma

Figura 3.3 Schermata per lo scenario di inserimento dati.

Scenario conferma della prenotazione

Ipotesi iniziale: un utente ha inserito i dati per effettuare una prenotazione e deve procedere alla fase di verifica e conferma.

Normale: il sistema invia all'utente (all'indirizzo e-mail inserito precedentemente) un codice di quattro cifre; tale codice viene a lui richiesto per confermare la sua prenotazione. L'utente può scegliere tramite l'apposito bottone di farsi spedire un nuovo codice, nel caso in cui lo desiderasse.

Che cosa può andare male: l'utente inserisce un codice diverso da quello che gli è stato inviato; in questo caso gli viene mostrato un messaggio di errore.

Che cosa può andare male: nel momento in cui l'utente invia il codice (corretto) lo slot da lui selezionato è nel frattempo stato prenotato da un altro utente; in questo caso la richiesta fallisce e l'utente viene indirizzato su una pagina di errore, dove con un messaggio gli viene spiegato che lo slot risulta occupato. Dalla pagina di errore tramite un collegamento è possibile tornare alla pagina principale.

Stato del sistema al completamento: dopo che l'utente ha inserito correttamente il codice a 4 cifre e ha confermato la prenotazione, gli viene mostrato un riepilogo che viene anche inviato automaticamente al suo indirizzo e-mail.

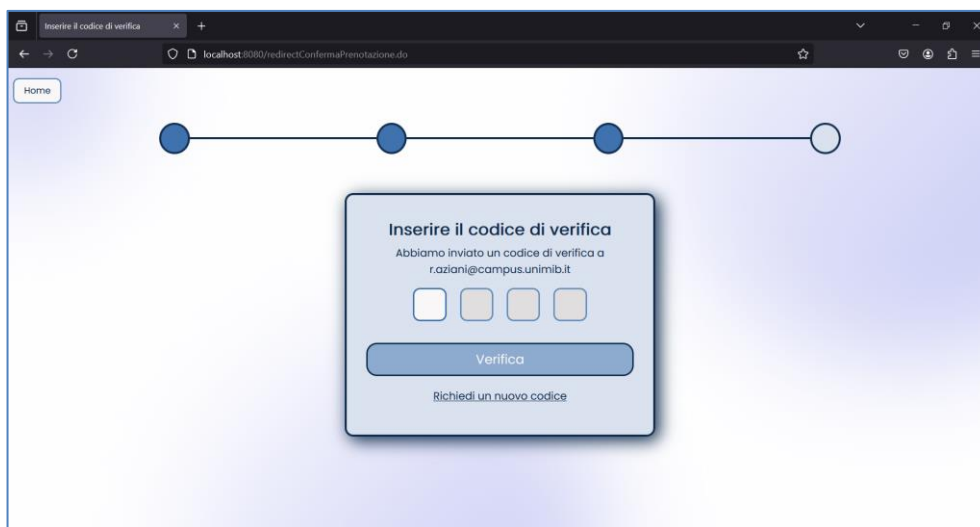


Figura 3.4 Schermata per lo scenario di conferma della prenotazione, inserimento del codice di conferma.

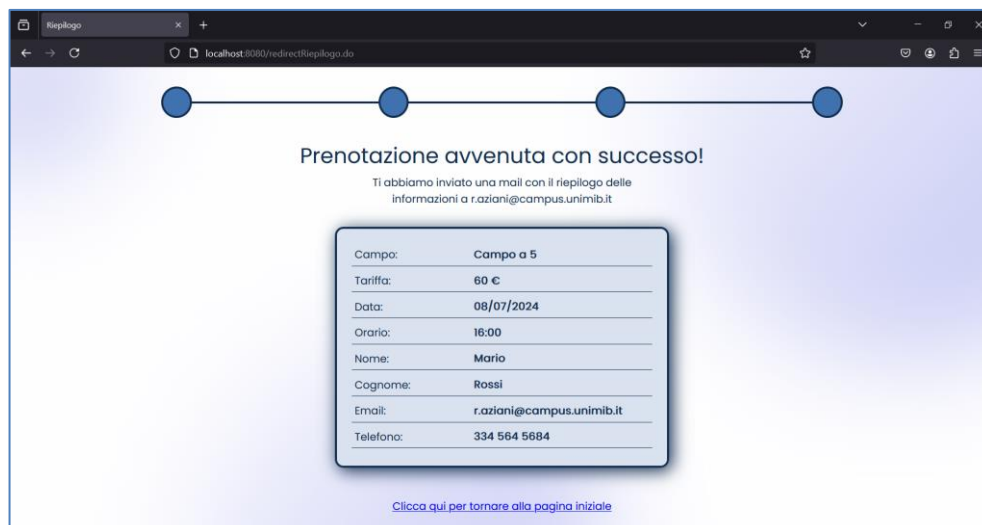


Figura 3.5 Schermata per lo scenario di conferma della prenotazione, riepilogo della prenotazione.

Segue adesso la storia, con i rispettivi scenari, per la gestione di una prenotazione.

Requisiti utente esterno

Gestione di una prenotazione

Riccardo ha effettuato una prenotazione, ma lui e i suoi amici hanno deciso di rimandare di sette giorni la loro partita.

Riccardo dalla schermata principale clicca sul pulsante “Gestisci prenotazione”: gli viene richiesto il codice associato alla sua prenotazione. A questo punto, recupera la mail di conferma che ha ricevuto quando ha effettuato la prenotazione, copia il codice e lo inserisce nel sistema; clicca sul bottone “Elimina prenotazione”, conferma la sua volontà di voler eliminare la prenotazione confermando nella finestra che viene aperta, e così la sua prenotazione viene cancellata.

Scenario inserimento codice della prenotazione

Ipotesi iniziale: un utente ha cliccato sul pulsante “gestisci la prenotazione” ed ha davanti a sé la schermata in cui gli viene chiesto di inserire il codice della sua prenotazione.

Normale: l’utente inserisce il codice della sua prenotazione e clicca sul tasto “conferma”.

Che cosa può andare male: l’utente inserisce un codice non valido; in questo caso il sistema gli mostra un messaggio per ricordargli che il codice della prenotazione ha una lunghezza di 12 caratteri alfanumerici.

Che cosa può andare male: l’utente inserisce un codice inesistente; in questo caso il sistema gli mostra un messaggio per fargli sapere che il codice da lui inserito non è associato a nessuna prenotazione.

Stato del sistema al completamento: dopo che l’utente ha inserito il suo codice e ha cliccato sul tasto “conferma”, viene indirizzato alla pagina in cui può scegliere quale operazione effettuare sulla sua prenotazione.

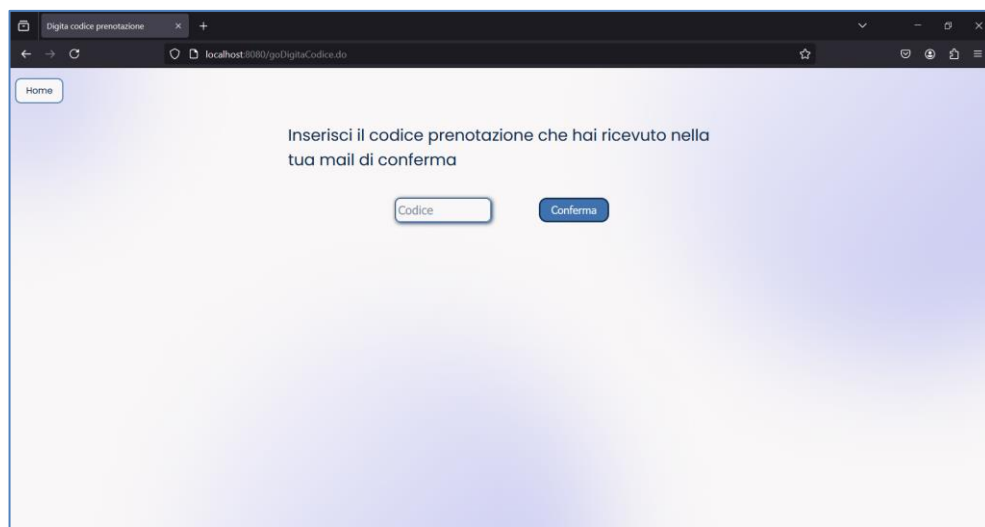


Figura 3.6 Schermata per lo scenario di inserimento del codice della prenotazione.

Scenario elimina prenotazione

Ipotesi iniziale: un utente ha inserito il codice della sua prenotazione correttamente.

Normale: l'utente clicca sul pulsante "Elimina prenotazione"; a questo punto viene aperta una finestra per chiedere conferma all'utente di eliminare la sua prenotazione; cliccando su "elimina" la prenotazione viene eliminata.

Che cosa può andare male: mancano meno di 24 ore alla sua prenotazione; in questo caso gli viene mostrato un messaggio di errore in cui gli viene ricordato che è possibile eliminare o spostare la prenotazione solo fino a 24 ore prima dell'inizio dell'evento.

Stato del sistema al completamento: dopo che l'utente ha eliminato la prenotazione gli viene mostrata una schermata che conferma l'esito positivo dell'operazione.

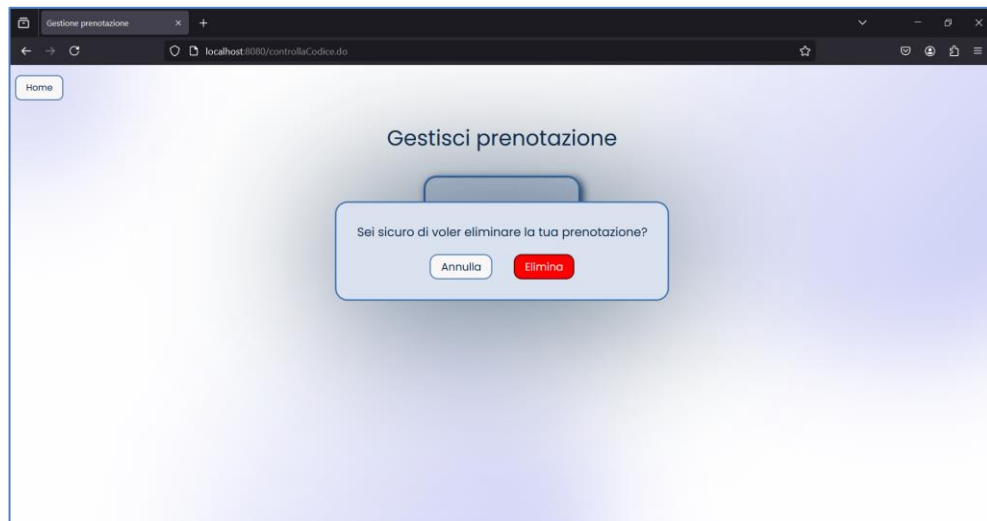


Figura 3.7 Schermata per lo scenario di eliminazione di una prenotazione.

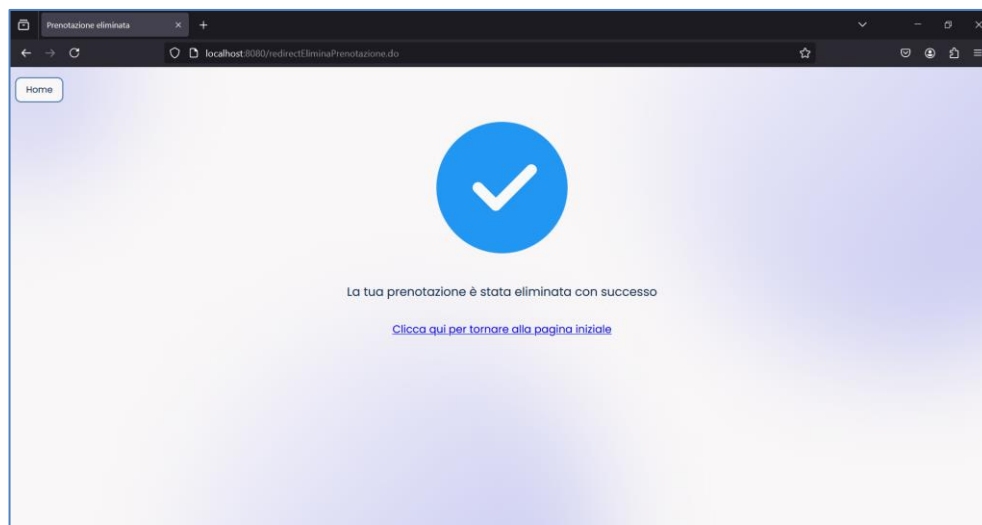


Figura 3.8 Schermata per lo scenario di eliminazione di una prenotazione, procedura avvenuta con successo.

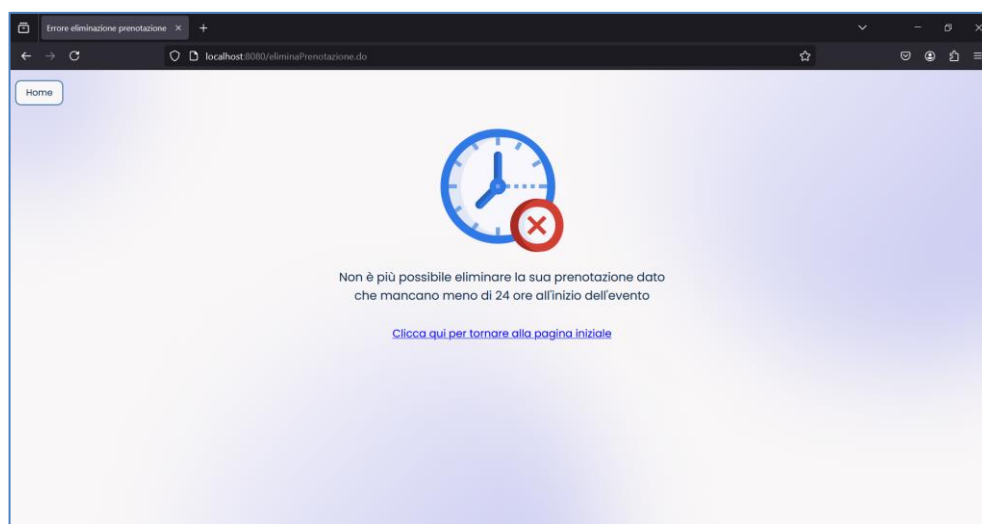


Figura 3.9 Schermata per lo scenario di eliminazione di una prenotazione, errore durante la procedura.

3.1.2. Storie e scenari, utente interno

In questa sezione è presente una storia, approfondita nei rispettivi scenari, per spiegare quali sono i servizi che il sistema deve offrire all'utente interno.

Requisiti utente interno

Gestione delle prenotazioni

Un utente interno può accedere all'area riservata superando un controllo di autenticazione: cliccando sul pulsante apposito, viene aperta una finestra di dialogo in cui sono chieste delle credenziali. Se vengono inserite correttamente, l'utente viene indirizzato alla pagina per gestire le prenotazioni. Tramite un menù può spostarsi tra tre sezioni: la pagina che mostra tutte le prenotazioni attive e che permette di eliminarle; la pagina che mostra tutte le prenotazioni che non sono state saldate e che permette di contrassegnarle come tali; la pagina che si occupa di modificare la disponibilità dei campi.

Se per uno slot è presente una prenotazione, l'utente non può togliere la disponibilità del campo.

Come per gli utenti esterni, la storia viene sviluppata in modo più approfondito con l'uso degli scenari.

Scenario login utente interno

Ipotesi iniziale: l'utente ha davanti a sé la schermata per effettuare l'autenticazione, che richiede un nome utente e una password.

Normale: l'utente inserisce correttamente le credenziali, e viene indirizzato alla pagina per gestire le prenotazioni.

Che cosa può andare male: l'utente inserisce delle credenziali sbagliate e fallisce il controllo, e viene fatto tornare alla pagina iniziale.

Stato del sistema al completamento: l'utente è nella pagina per gestire le prenotazioni

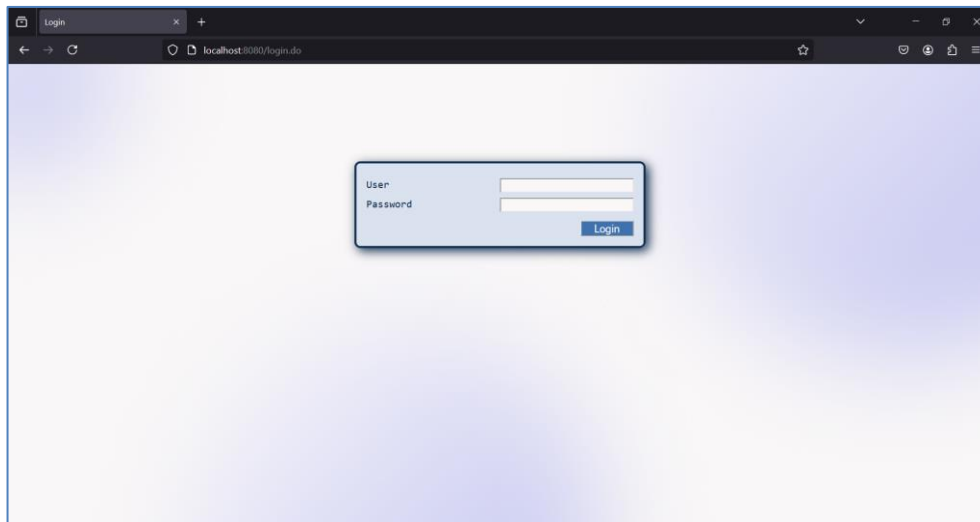


Figura 3.10 Schermata per lo scenario di login.

Scenario segnare una prenotazione come pagata

Ipotesi iniziale: l'utente ha davanti a sé la schermata che mostra una tabella contenente l'elenco di tutte le prenotazioni che ancora non sono state saldate.

Normale: l'utente clicca sul pulsante presente nell'ultima colonna, contrassegnando la prenotazione come saldata e quindi rimuovendola dall'elenco.

Stato del sistema al completamento: all'utente viene mostrata la tabella aggiornata, senza la prenotazione che ha segnato come pagata.

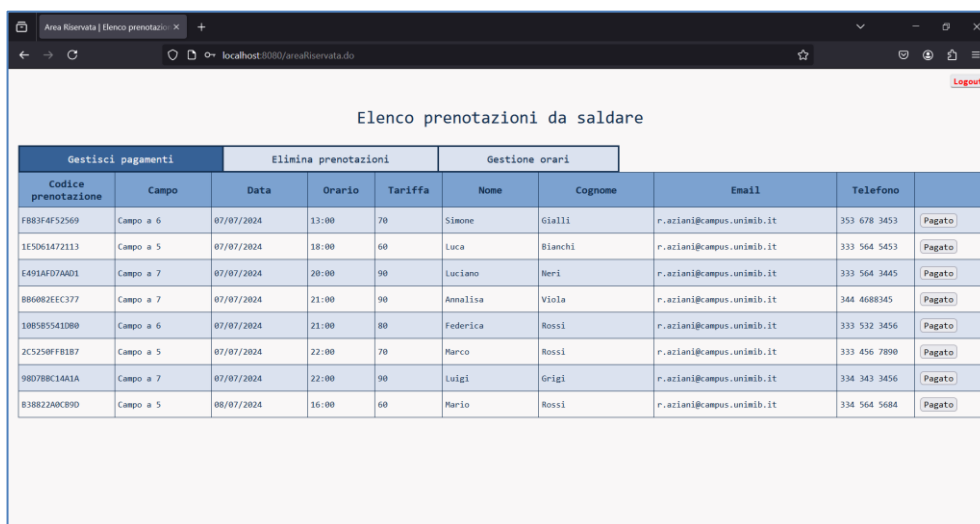


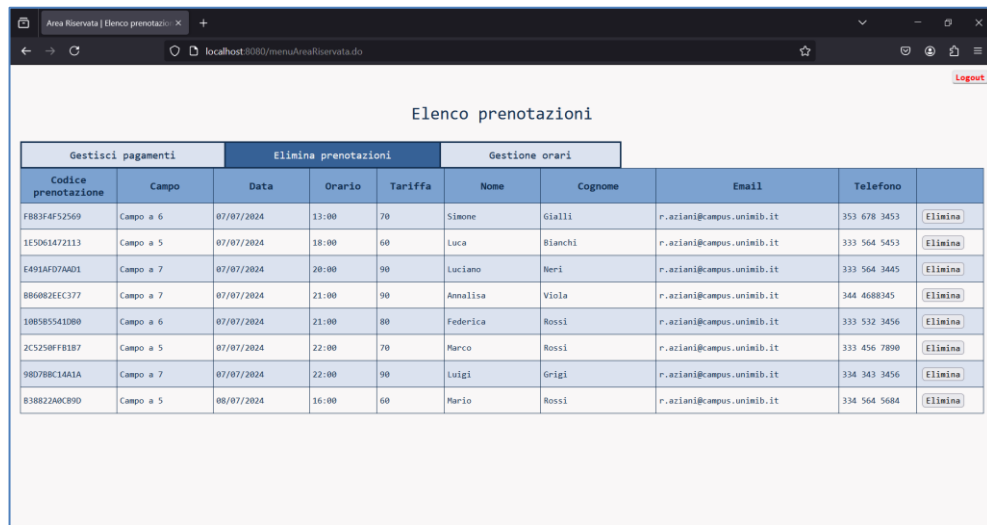
Figura 3.11 Schermata per lo scenario di contrassegnare una prenotazione come pagata.

Scenario elimina una prenotazione

Ipotesi iniziale: l'utente ha davanti a sé la schermata che mostra una tabella contenente l'elenco di tutte le prenotazioni.

Normale: l'utente clicca sul pulsante presente nell'ultima colonna, contrassegnando eliminando e quindi rimuovendola dall'elenco.

Stato del sistema al completamento: all'utente viene mostrata la tabella aggiornata, senza la prenotazione che è stata eliminata.



The screenshot shows a web browser window with the address bar displaying 'localhost:8080/menualnaRiservato.do'. The page title is 'Area Riservata | Elenco prenotazioni'. The main content area is titled 'Elenco prenotazioni' and contains a table with reservation details. Above the table, there are three tabs: 'Gestisci pagamenti', 'Elimina prenotazioni' (which is active), and 'Gestione orari'. The table has columns for 'Codice prenotazione', 'Campo', 'Data', 'Orario', 'Tariffa', 'Nome', 'Cognome', 'Email', 'Telefono', and an 'Elimina' button. The data rows show various reservations with their respective details.

Gestisci pagamenti		Elimina prenotazioni			Gestione orari				
Codice prenotazione	Campo	Data	Orario	Tariffa	Nome	Cognome	Email	Telefono	
FB83F4E52569	Campo a 6	07/07/2024	13:00	70	Simone	Gialli	r.aziani@campus.unimib.it	353 678 3453	Elimina
1E5D61A72113	Campo a 5	07/07/2024	18:00	60	Luca	Bianchi	r.aziani@campus.unimib.it	333 564 5453	Elimina
E491AFD7AAD1	Campo a 7	07/07/2024	20:00	90	Luciano	Neri	r.aziani@campus.unimib.it	333 564 3445	Elimina
8B6082EEC377	Campo a 7	07/07/2024	21:00	90	Annalisa	Viola	r.aziani@campus.unimib.it	344 4688345	Elimina
10F5855A1D80	Campo a 6	07/07/2024	21:00	80	Federica	Rossi	r.aziani@campus.unimib.it	333 532 3456	Elimina
2C5250FFB1B7	Campo a 5	07/07/2024	22:00	70	Marco	Rossi	r.aziani@campus.unimib.it	333 456 7890	Elimina
98078BC14A1A	Campo a 7	07/07/2024	22:00	90	Luigi	Grigi	r.aziani@campus.unimib.it	334 343 3456	Elimina
B38822A0CB9D	Campo a 5	08/07/2024	16:00	60	Mario	Rossi	r.aziani@campus.unimib.it	334 564 5684	Elimina

Figura 3.12 Schermata per lo scenario di eliminazione di una prenotazione.

Scenario scelta del giorno e campo di cui si vuole modificare la disponibilità

Ipotesi iniziale: l'utente ha davanti a sé la pagina per modificare la disponibilità.

Normale: il sistema chiede all'utente di selezionare un giorno del calendario e uno dei campi; successivamente clicca sul tasto conferma.

Che cosa può andare male: l'utente non seleziona alcun campo prima di cliccare sul tasto conferma; in questo caso gli viene mostrato un messaggio di errore che gli ricorda di selezionare uno dei campi.

Che cosa può andare male: l'utente seleziona una data del calendario antecedente alla data odierna; in questo caso gli viene mostrato un messaggio di errore che lo invita a correggere la scelta.

Stato del sistema al completamento: dopo che l'utente ha inserito i dati in modo coerente, cliccando su conferma l'utente viene indirizzato alla pagina per la gestione della disponibilità degli slot orari.

Modifica disponibilita

Gestisci pagamenti Elimina prenotazioni **Gestione orari**

Seleziona la data:

Seleziona il campo:

☐ Campo a 5

☐ Campo a 6

☐ Campo a 7

[Logout](#)

Figura 3.13 Schermata per lo scenario della gestione orari.

Scenario modifica della disponibilità

Ipotesi iniziale: l'utente ha selezionato un giorno e un campo, e ha davanti a sé la schermata per modificare la disponibilità degli slot orari.

Normale: il sistema mostra all'utente l'elenco dei 24 slot orari, a partire da 00:00 fino alle 23:00. Per ogni slot può spuntare o meno la casella selezionata per decidere se rendere tale slot disponibile o meno; a fianco della casella, può digitare l'importo in euro per quello slot orario.

Al caricamento della pagina gli slot che sono disponibili vengono spuntati automaticamente dal sistema, e lo stesso vale per i relativi importi.

Per gli slot che sono stati prenotati non sarà possibile rimuovere la disponibilità.

Che cosa può andare male: l'utente digita in una delle caselle degli importi un input non valido; in questo caso il sistema mostra un messaggio di errore all'utente.

Stato del sistema al completamento: dopo che l'utente ha cliccato sul tasto conferma, il sistema esegue le modifiche scelte dall'utente e lo reindirizza alla pagina in cui viene chiesto di scegliere un giorno del calendario e uno dei campi da modificare.

The screenshot shows a web browser window with the address bar displaying 'localhost:8080/scegliData.do'. The page title is 'Modifica disponibilità'. Below the title, there is a text prompt: 'Seleziona gli orari da rendere disponibili e inserisci il relativo importo:'. The interface contains a grid of 24 time slots, arranged in three columns and eight rows. Each slot consists of a checkbox and a text input field for the amount. The slots are labeled from 00:00 to 23:00. The following table represents the state of the slots as shown in the image:

Time Slot	Availability (Checkbox)	Amount (Input Field)
00:00	<input type="checkbox"/>	0
01:00	<input type="checkbox"/>	0
02:00	<input type="checkbox"/>	0
03:00	<input type="checkbox"/>	0
04:00	<input type="checkbox"/>	0
05:00	<input type="checkbox"/>	0
06:00	<input type="checkbox"/>	0
07:00	<input type="checkbox"/>	0
08:00	<input type="checkbox"/>	0
09:00	<input type="checkbox"/>	0
10:00	<input type="checkbox"/>	0
11:00	<input type="checkbox"/>	0
12:00	<input type="checkbox"/>	0
13:00	<input type="checkbox"/>	0
14:00	<input type="checkbox"/>	0
15:00	<input type="checkbox"/>	0
16:00	<input type="checkbox"/>	0
17:00	<input checked="" type="checkbox"/>	60
18:00	<input checked="" type="checkbox"/>	60
19:00	<input type="checkbox"/>	0
20:00	<input checked="" type="checkbox"/>	70
21:00	<input checked="" type="checkbox"/>	70
22:00	<input checked="" type="checkbox"/>	70
23:00	<input type="checkbox"/>	0

A 'Conferma' button is located at the bottom right of the form area. A 'Logout' link is visible in the top right corner of the page.

Figura 3.14 Schermata per lo scenario di modifica della disponibilità.

3.2. Progettazione della base di dati

Il primo passo per la progettazione di una base di dati è quello analizzare i requisiti, raccogliendo le specifiche sui dati e sulle operazioni che è necessario svolgere su di essi. In questa fase è stato utile prelevare dal testo le frasi relative agli stessi concetti e raggrupparle tra di loro.

Campo: *per un campo rappresentare il nome e il numero di giocatori.*

Prenotazione: *per una prenotazione rappresentare la data, l'orario, la tariffa, i dati dell'utente e un booleano per determinare se il pagamento è stato effettuato o no.*

Disponibilità: *per una disponibilità rappresentare la data, l'orario di inizio, l'orario di fine e la tariffa. Ogni slot che è possibile prenotare avrà la durata di un'ora.*

Oltre alle specifiche sui dati, è utile anche raccogliere tutte le operazioni che sarà necessario svolgere.

Operazione 1: *stampare tutti i campi.*

Operazione 2: *stampare gli slot orari per cui esiste una disponibilità associata.*

Operazione 3: *per gli slot per cui esiste una disponibilità associata, stampare se esiste anche una prenotazione.*

Operazione 4: *aggiungere una nuova prenotazione.*

Operazione 5: *stampare tutte le prenotazioni.*

Operazione 6: *stampare tutte le prenotazioni che devono ancora essere pagate.*

Operazione 7: *contrassegnare una prenotazione come pagata.*

Operazione 8: *eliminare una prenotazione.*

Operazione 9: *modificare la disponibilità di un campo.*

Dopo aver analizzato e raccolto i requisiti in questa forma strutturata, si può procedere a progettare un modello concettuale in grado di rappresentare tutte le specifiche. Una possibile implementazione è rappresentata in Figura 3.15 .

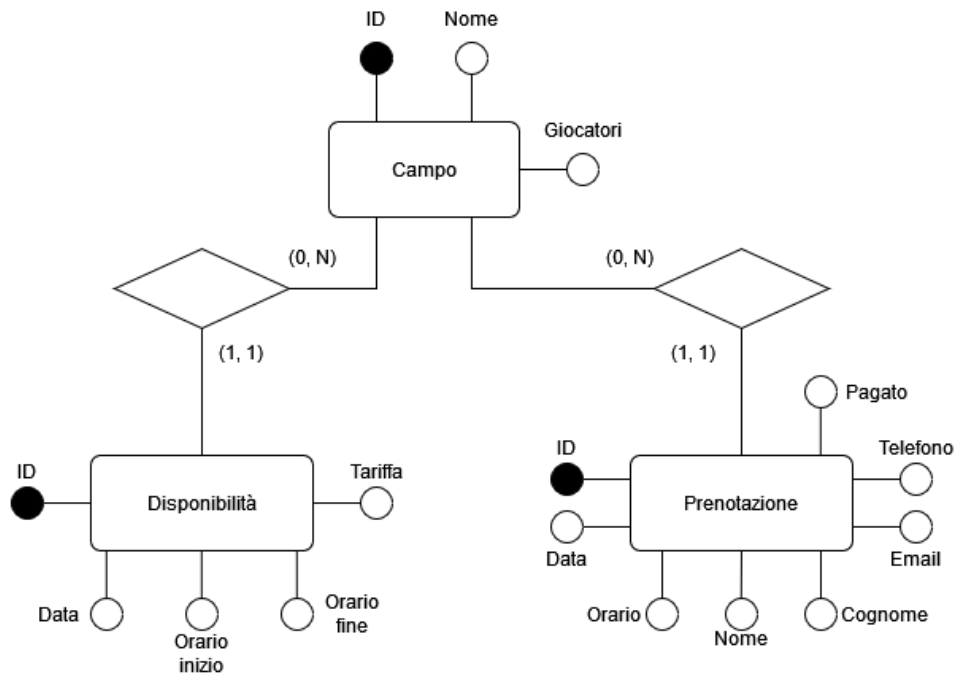


Figura 3.15 Un possibile schema concettuale secondo il modello ER.

La seconda fase della progettazione di una base di dati è la progettazione logica, in cui lo schema concettuale viene tradotto in un modello logico. Come si vede in Figura 3.16, in questo caso non ci sono particolari modifiche da apportare; è stata solamente aggiunta una tabella con lo scopo di memorizzare il codice hash delle credenziali necessarie all'autenticazione degli utenti interni.

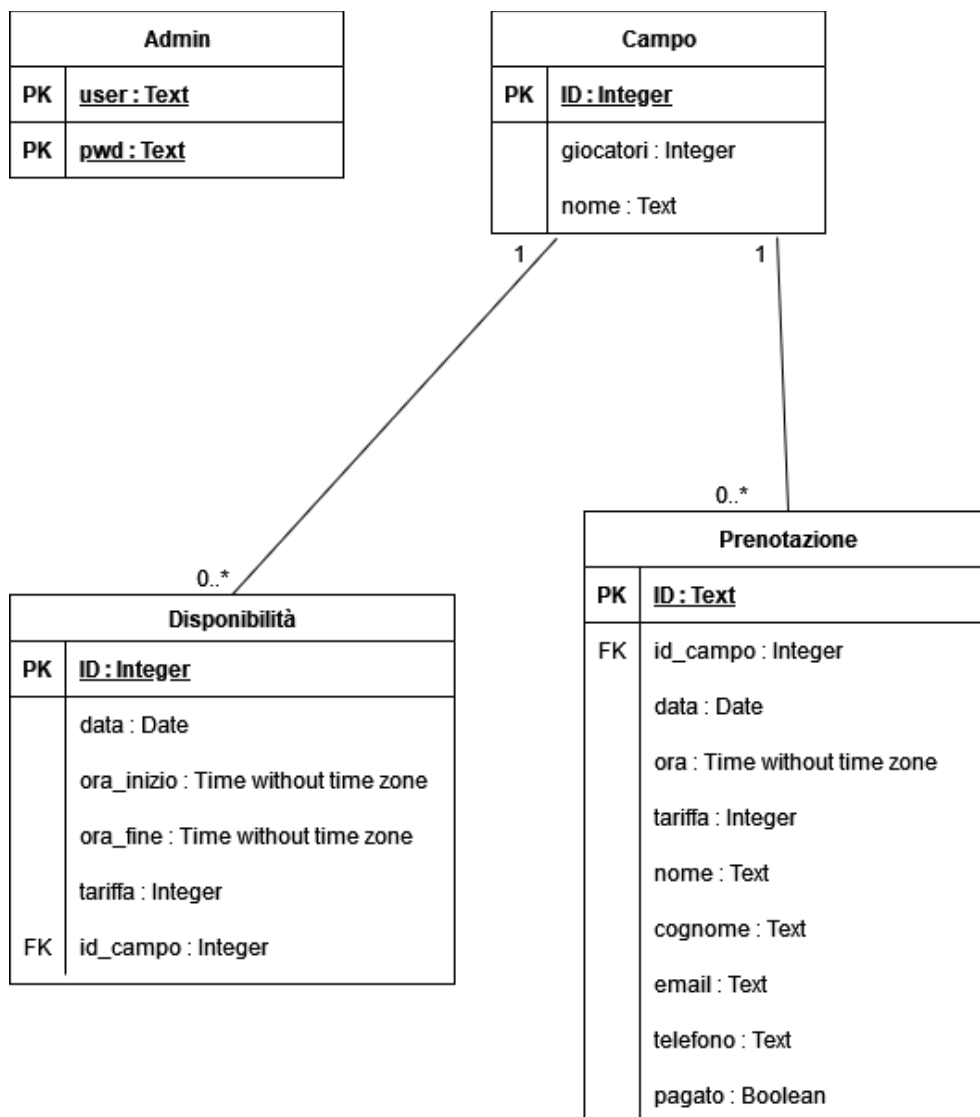


Figura 3.16 Traduzione nello schema logico.

Per implementare fisicamente il database è stato usato PostgreSQL; si tratta di un DBMS che ha guadagnato una solida reputazione per la sua architettura comprovata, affidabilità, integrità dei dati, ricca gamma di funzionalità, estensibilità e la dedizione della comunità open-source che supporta il software nel fornire costantemente soluzioni performanti e innovative. [10]

3.3. Progettazione architetturale

Per sviluppare l'applicazione è stato utilizzato il framework Struts, in quanto implementa una distinzione dei compiti dei vari componenti dell'architettura più chiara e fedele alle linee guida del pattern MVC.

La sua natura dichiarativa permette di concentrarsi soltanto sulla gestione del controller e dello sviluppo delle classi per rispondere alle richieste, mentre l'elaborazione delle viste nelle pagine JSP e la logica del Model è totalmente indipendente da esso. Questo significa che si può cambiare la logica di controllo senza intaccare il codice sorgente, a differenza di quanto avviene con il meccanismo delle annotazioni usato da Spring MVC. Una rappresentazione dell'architettura dell'applicazione è mostrata in Figura 3.17.

Come spiegato nella sezione 2.3.2, il controller Struts si occupa essenzialmente di tre attività quando viene invocata un'azione:

- Riempie il form-bean associato (che estende la classe `ActionForm`), ovvero un oggetto che funge da contenitore di dati per quella specifica azione, con lo scopo di raccogliere i dati della richiesta.
- Successivamente delega l'esecuzione alla classe `Action` che gestisce quella determinata azione, che ritornerà un forward.
- Il controller Struts verifica quale forward è stato ritornato e mostra all'utente la pagina JSP associata a tale forward.

Il controller viene configurato nel file *struts-config.xml*, usando una struttura dichiarativa: per ogni azione viene mappata quale classe ha il compito di gestirla, quale form-bean viene popolato (automaticamente) e quali viste devono essere mostrate all'utente in base all'esito della richiesta. Un esempio di dichiarazione di una generica azione è mostrato nella Figura 3.18.

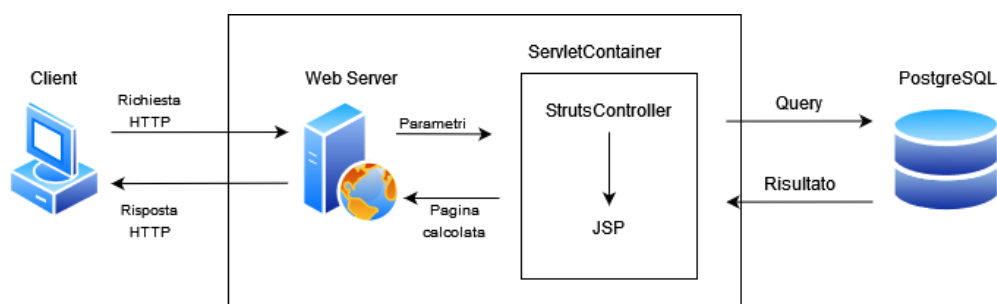


Figura 3.17 Architettura dell'applicazione.

```
<struts-config>
  <form-beans>
    <form-bean name="myBean" type="<class>"/>
  </form-beans>
  <action-mappings>
    <action          path="<path>"          type="<class>"
name="myBean">
      <forward name="success" path="<pathSuccess>"/>
      <forward name="fail" path="<pathFail>"/>
    </action>
  </action-mappings>
</struts-config>
```

Figura 3.18 Esempio di dichiarazione di una azione nel file struts-config.xml.

3.4. Test del software

Per testare il software è opportuno automatizzare la procedura di testing. Questo permette di verificare la correttezza del codice in maniera continua e sistematica, offrendo la possibilità di eseguire dei test di regressione.

Per progettare ed eseguire i test di unità è stato utilizzato il framework open-source JUnit, che permette di eseguire i test direttamente sul JVM.

In aggiunta è stato usato anche Mockito, uno tra i più usati framework di mocking per Java: significa che permette di simulare degli oggetti per testarne il comportamento [11]. Mockito è stato indispensabile per poter testare le funzioni che interagiscono con il database, permettendo di simulare il loro comportamento e verificarne la correttezza, senza effettuare delle vere e proprie transazioni sul database fisico. Un esempio di caso di test che sfrutta il mocking degli oggetti è mostrato in Figura 3.19: in particolare, in questo caso si vuole verificare un metodo che collegandosi al database elimina una prenotazione, passando la sua chiave primaria (id) come argomento della funzione; come si può vedere, sfruttando gli oggetti mock è possibile verificare che su di essi vengano fatto le corrette invocazioni dei metodi.

Per i test di integrazione è stato utilizzato StrutsTestCase, un'estensione della libreria standard JUnit che fornisce delle funzionalità per testare il codice basato sul framework Struts [12]. StrutsTestCase permette di testare la logica, il mapping e il forward delle azioni; per fare ciò, come si vede in Figura 3.20 si utilizzano dei metodi per simulare i parametri della richiesta, invocare l'azione, verificare quale forward viene ritornato e che la vista associato ad esso sia corretta.

Con lo strumento EcEmma, integrato nell'ambiente di sviluppo Eclipse, è stata calcolata una copertura del codice del 66%, con un totale di 41 casi di test.

```
@Test
public void eliminaPrenotazione() throws SQLException {
    Connection mockConnection = mock(Connection.class);
    PreparedStatement mockSt = mock(PreparedStatement.class);

    when(mockConnection.prepareStatement("DELETE FROM "
        + "public.\"Prenotazioni\" WHERE id = ?")).thenReturn(mockSt);

    PrenotazioneDAO.eliminaPrenotazione(mockConnection, "id");

    verify(mockSt).setString(1, "id");
    verify(mockSt).executeUpdate();
    verify(mockSt).close();
}
```

Figura 3.19 Un caso di test che utilizza Mockito.

L'automatizzazione del test si è rivelata utile non solo per verificare il corretto funzionamento del codice, ma anche per velocizzare i processi di sviluppo, permettendo di individuare e risolvere rapidamente i problemi.

Per testare il sistema è stato utilizzato Apache Tomcat; è un software open-source che funge da Application Server e Servlet Container per eseguire applicazioni web basate su Java. In questo modo è possibile eseguire e verificare la correttezza del funzionamento dell'intero sistema.

```
@Test
public void testExecuteSuccess() {
    addRequestParameter("id", "E3493541AB2A");

    setRequestPathInfo("/controllaCodice");

    actionPerform();

    verifyForward("success");
    verifyTilesForward("success", "tiles.gestisciPrenotazione");
}

@Test
public void testExecuteFail() {
    addRequestParameter("id", "fail");

    setRequestPathInfo("/controllaCodice");

    actionPerform();

    verifyForward("fail");
    verifyTilesForward("fail", "tiles.codiceNonTrovato");
}
```

Figura 3.20 Un caso di test che utilizza StrutsTestCase.

3.5. Deploy

Per il deploy è stato utilizzato Docker, uno tra i più popolari software per il deployment di applicazioni in dei container di dimensioni contenute. Per descrivere il funzionamento di Docker, è indispensabile trattare due concetti: le immagini Docker e il Dockerfile.

Possiamo definire le immagini Docker come una istantanea della nostra applicazione contenente tutto ciò che è necessario per la sua esecuzione, come ad esempio il codice, le librerie e le dipendenze. Il Dockerfile è un file contenente le istruzioni per creare tale immagine Docker. Un container Docker invece è sostanzialmente una istanza di una immagine in esecuzione. Questo processo di costruzione ed esecuzione di una immagine è visibile in Figura 3.21 .

Il primo passo dunque è la creazione delle immagini necessarie al deploy; nel nostro caso si tratta di una immagine per l'applicazione ed una immagine per il database.

Per la creazione dell'immagine dell'applicazione quest'ultima viene esportata nel file .WAR, ovvero un file contenente tutti i file, le librerie e le dipendenze necessarie alla sua esecuzione; successivamente, viene configurato il Dockerfile e creata una immagine. Allo stesso modo, viene creata una immagine per il database; in questo caso, è stata direttamente scaricata la immagine più recente di PostgreSQL, disponibile su DockerHub. Una volta create, le immagini sono immutabili nel tempo.

L'ultimo passo è la creazione e l'esecuzione dei container, ovvero l'istanza in esecuzione delle immagini appena create. Una volta creati e lanciati i container, è possibile eseguire l'applicazione sfruttando i vantaggi della virtualizzazione ottenuta tramite container.

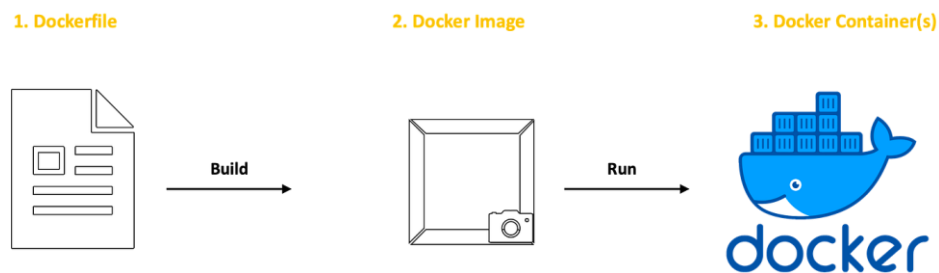


Figura 3.21 Funzionamento di Docker [13]

Capitolo 4: Bibliografia

- [1] I. Sommerville, Ingegneria del software, Decima edizione, 2017.
- [2] P. Atzeni, S. Ceri, P. Fraternali, S. Paraboschi e R. Torlone, Basi di dati, V Edizione, 2018.
- [3] D. E. W., The Humble Programmer, 1972.
- [4] K. Beck, Test Driven Development: By Example, 2002.
- [5] J. Whittaker, Exploratory Software Testing. Tips, Tricks, Tours, and Techniques to Guide Test Design, 2009.
- [6] IBM, «Cosa sono i contenitori?,» [Online]. Available: <https://www.ibm.com/it-it/topics/containers>. [Consultato il giorno Luglio 2024].
- [7] IBM, «Container vs VM: qual è la differenza?,» 10 Settembre 2020. [Online]. Available: <https://www.youtube.com/watch?v=cjXI-yxqGTI>. [Consultato il giorno Luglio 2024].
- [8] IBM, «Cos'è DevOps?,» [Online]. Available: <https://www.ibm.com/it-it/topics/devops>. [Consultato il giorno Luglio 2024].
- [9] IBM, «Cos'è l'Integrazione Continua?,» [Online]. Available: <https://www.ibm.com/it-it/topics/continuous-integration>. [Consultato il giorno Luglio 2024].
- [10] PostgreSQL, «PostgreSQL: The world's most advanced open source database,» Maggio 2024. [Online]. Available: <https://www.postgresql.org/about/>.
- [11] Mockito, «Mockito FAQ,» [Online]. Available: <https://github.com/mockito/mockito/wiki/FAQ>. [Consultato il giorno Maggio 2024].
- [12] StrutsTestCase, «StrutsTestCase for JUnit,» [Online]. Available: <https://strutstestcase.sourceforge.net/>.
- [13] freeCodeCamp, «How Docker Containers Work,» [Online]. Available: <https://www.freecodecamp.org/news/how-docker-containers-work/>. [Consultato il giorno Luglio 2024].