

UNIVERSITA' DEGLI STUDI DI
NAPOLI FEDERICO II

Scuola Politecnica e delle Scienze di Base
Corso di Laurea in Ingegneria Informatica

Elaborato finale in **Software Architecture Design**

Applicazione Web: *bookCamping*

Anno Accademico 2021/2022

Candidato:

Francesco Crescenzo Grasso

matr. M63001244

bookCamping

Prenotazione Campeggi ed Escursioni al Camping di
Villetta Barrea (AQ), Abruzzo

Indice

<u>Introduzione</u>	5
<u>Capitolo 1</u> : Processo di sviluppo UP e organizzazione.....	6
1.1 Sezione generale	6
1.2 Fasi di lavoro	7
1.3 Ordinamento	9
1.4 Fabbisogno delle risorse.....	9
<u>Capitolo 2</u> : Documenti e modelli d'avvio	10
2.1 Requisiti generali	10
2.2 Assunzioni	10
2.3 Parti interessate	11
2.4 Vincoli di carattere generale	11
<u>Capitolo 3</u> : Specifica dei Requisiti.....	13
3.1 Specifica dei Requisiti Funzionali	13
3.2 Specifica dei Requisiti Non-Funzionali	14
<u>Capitolo 4</u> : Analisi dei Requisiti.....	15
4.1 Use Cases Diagram	15
4.2 Descrizione dei Casi D'Uso	16
4.3 Stima Costi dei Casi D'Uso.....	24
<u>Capitolo 5</u> : Architettura Software	28
5.1 System Domain Model.....	30
5.2 Sequence Diagram d'Analisi	31
5.3 Activity Diagram con <i>Swimlanes</i>	32
5.4 Component Diagram	33
5.5 Package Diagram	34
5.6 Sequence Diagram di Dettaglio	37
5.7 Deployment Diagram.....	39
5.8 Manifestation of Component by <i>Artifact</i>	39

<u>Capitolo 6:</u> Strumentazione ed implementazione	40
6.1 Ambiente di Sviluppo: <i>Eclipse for Java and Web Developers</i>	40
6.1.1 Installazione	40
6.1.2 Configurazione	40
6.2 Database <i>XAMPP</i>	41
6.2.1 Installazione	41
6.2.2 Configurazione	41
6.3 Server <i>Apache Tomcat 9.0</i>	42
6.3.1 Installazione	42
6.3.2 Configurazione	42
6.4 Implementazione	43
6.4.1 Linguaggi di Programmazione	49
6.4.2 Design Framework: <i>Bootstrap</i>	49
6.4.3 Esecuzione del Software prodotto	50
6.4.4 <i>GitHub</i>	53
<u>Capitolo 7:</u> Test e prove eseguite	55
<u>Glossario</u>	59
<u>Conclusioni</u>	60

Introduzione

bookCamping

Il progetto vuole invogliare gli utenti a viaggiare, ad apprezzare più da vicino le bellezze che la natura ci offre e che talvolta ignoriamo, distratti dalla sete di voler essere costantemente al centro del mondo, nelle nostre grigie ed inquinate città. Questo progetto è un modo per avvicinarsi alla natura attraverso la tecnologia, si rende promotore di un movimento che ragiona attraverso la massima “*Less is More*”, secondo cui meno si ha, più si può aspirare ad essere felici in un mondo migliore.

L'*applicazione Web* ideata, documentata, progettata ed implementata si chiama **bookCamping**. L'obiettivo di questo progetto è quello di intermediare tre due diverse realtà: la realtà del Camping di *Villetta Barrea* (AQ) situato nel *Parco Regionale dell'Abruzzo* a circa un chilometro d'altitudine e la realtà di chi necessita di evadere dalla città, di chi ha bisogno di ritrovare la propria essenza entrando a contatto con l'entità da cui tutti proveniamo: la *natura*.

Se stai cercando, il campeggio ideale per vivere un'esperienza indimenticabile nelle montagne dell'Abruzzo questa è la soluzione che fa per te. Lasciati trasportare dalla natura incontaminata, dai paesaggi spettacolari degli Appennini e dalle molteplici avventure ed escursioni guidate da esperti.

Il campeggio rispetta la natura e mette a disposizione postazioni per accendere il fuoco e dotate di tavoli e sedie di legno. Esso è situato tra le montagne dell'Abruzzo che comprendono la “*Riserva Naturale La Camosciara*” e “*L'Area faunistica del Lupo*”. Camosci, cerbiatti e cervi fanno spesso visita al campeggio lasciando tutti gli ospiti del Camping senza parole. Un fiume costeggia, delimita il campo e sfocia nel lago di Barrea. Quest'ultimo presenta delle dimensioni molto estese e da anni attira turisti da tutta la penisola e non solo.

In particolare, tramite il *Web Site* è possibile prenotare *online* un soggiorno al Camping di Villetta Barrea scegliendo una o più piazzole di qualsiasi tipologia in base alle esigenze dell'utente, prenotare *online* delle escursioni guidate di diverse tipologie per una o più persone.

Capitolo 1: Processo di Sviluppo ed Organizzazione

1.1 Sezione generale

Per lo sviluppo del sistema software ho deciso di utilizzare come processo di sviluppo software un processo iterativo, flessibile ed evolutivo, chiamato **UP** che sta per *Unified Process*.

Per la produzione della Web Application sono state necessarie **due iterazioni**, a ciascuna delle quali è stato dedicato un tempo prefissato, noto come *timeboxing*, pari a **due settimane**. Il timeboxing mi ha obbligato a integrare, testare e stabilizzare il sistema entro i tempi prefissati. Eventuali requisiti non implementati in una iterazione, sono stati spostati alle successive iterazioni. Le iterazioni rispettano i principi di UP secondo cui iterazioni troppo lunghe sarebbero rischiose e contro il modello iterativo, mentre iterazioni troppo brevi potrebbero non consentire uno sviluppo sufficiente per ottenere feedback.

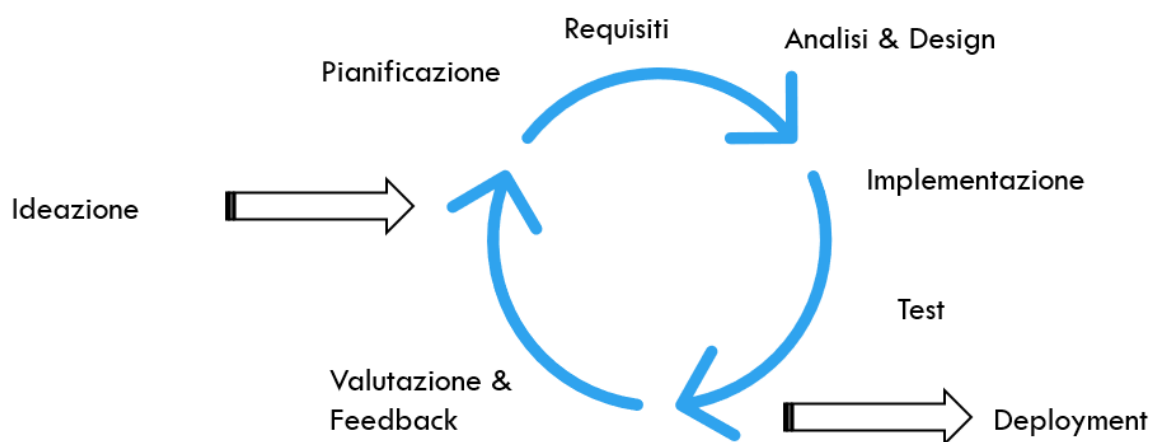
Fase	Inizio	Fine	Iterazione	Lavoro (gg/uomo)
Ideazione	01/09/2021	02/09/2021	Iterazione 1	2
Elaborazione 1°	03/09/2021	08/09/2021	Iterazione 1	6
Costruzione 1°	09/09/2021	13/09/2021	Iterazione 1	5
Transizione 1°	14/09/2021	15/09/2021	Iterazione 1	2
Elaborazione 2°	16/09/2021	21/09/2021	Iterazione 2	6
Costruzione 2°	22/09/2021	26/09/2021	Iterazione 2	5
Transizione 2°	27/09/2021	30/09/2021	Iterazione 2	4

La fase di *ideazione* presenta una breve durata, poiché avevo già avuto modo di discutere e pensare a quale genere di sistema implementare. Successivamente, attraverso studio di fattibilità e stime di costi e tempi, sono arrivato al concepimento dell'idea definitiva.

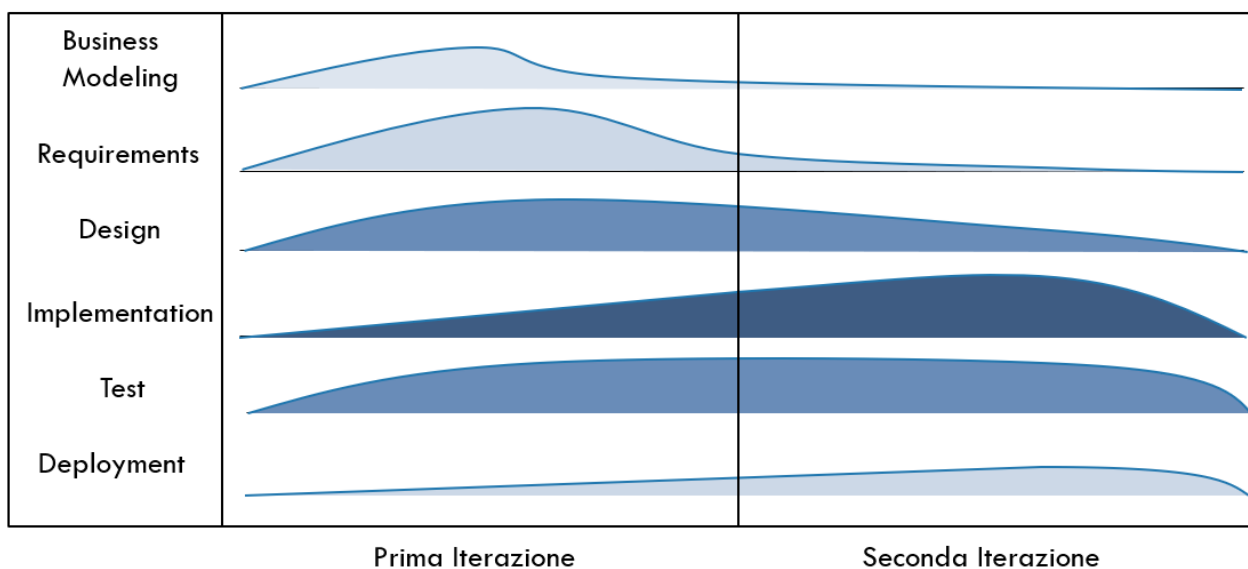
Inoltre, è facile notare che la prima fase di *transizione* è stata molto breve mentre la seconda fase di transazione si è compiuta nel doppio dei giorni. Questo è successo perché all'inizio della transizione associata alla seconda iterazione il progetto si è ingrandito e quindi le operazioni di testing e di integrazione hanno avuto una complessità maggiore.

1.2 Fasi di lavoro

Il processo di sviluppo adottato UP mi ha consentito di svolgere in ogni iterazione il lavoro di ogni *disciplina*: *Requirements* che ha come obiettivo la modellazione dei Casi d'uso e la specifica dei requisiti, *Business Modeling* che ha come scopo la realizzazione del Modello di Dominio, *Design* che si pone come scopo la realizzazione del modello di progetto, *Implementation* ovvero l'implementazione del sistema, *Test* e *Deployment*.



Il **Workflow** può mostrare in maniera intuitiva la distribuzione delle principali discipline nel tempo suddiviso in due iterazioni di uguale durata:



Risulta però necessario comprendere quali sono state le effettive attività svolte in ogni fase, quali sono stati i *milestones* e quali risorse sono state utilizzate.

Fase	Descrizione	Milestones
Ideazione	<ul style="list-style-type: none"> Esplorazione e concepimento dell'idea di base del progetto. Verifica della fattibilità con stime di costi e tempi. 	<ul style="list-style-type: none"> Documentazione su requisiti generali, parti interessate e assunzioni.
Elaborazione 1°	<ul style="list-style-type: none"> Generazione dei Casi d'uso con relativa descrizione, del System Domain Model e dei primi Sequence Diagram d'Analisi. Inizializzazione <i>Dynamic Web Project</i> su Eclipse. Configurazione Server e Database e creazione schemi per DB. Implementazione prime classi Model, classi Dao, pagine JSP e Servlet relative a <i>login, logout, registrazione e prenotazione</i>. 	<ul style="list-style-type: none"> Ambiente di sviluppo, DB e Server configurati e pronti per l'implementazione. Codice sorgente con le prime funzionalità.
Costruzione 1°	<ul style="list-style-type: none"> Costruzione della documentazione di progetto. 	<ul style="list-style-type: none"> Documenti progettuali parziali.
Transizione 1°	<ul style="list-style-type: none"> Verifica e testing delle funzionalità implementate. Ricevimento con la professoressa. 	<ul style="list-style-type: none"> Codice eseguibile: "bookCamping" versione 1.0
Elaborazione 2°	<ul style="list-style-type: none"> Cambiamento progettuale del sistema a causa di comportamenti impropri per un'architettura MVC. Prosecuzione del documento progettuale. Generazione di Activity Diagram, Sequence Diagram di dettaglio, Package Diagram. Implementazione pagine JSP e Servlet relative a <i>funzioni amministrative, profilo e pagamento</i>. 	<ul style="list-style-type: none"> Documenti progettuali in fase finale. Codice sorgente con tutte le funzionalità dei Casi d'uso.
Costruzione 2°	<ul style="list-style-type: none"> Correzione diagrammi ed implementazioni relative alle prenotazioni. Generazione Component, Deployment e Manifestation Diagram. 	<ul style="list-style-type: none"> Diagrammi UML. Codice sorgente rifinito con tutte le funzionalità. Documenti progettuali.
Transizione 2°	<ul style="list-style-type: none"> Integrazione e testing delle funzionalità implementate. 	<ul style="list-style-type: none"> Codice eseguibile: "bookCamping" versione 2.0

1.3 Ordinamento delle fasi di lavoro

ID	Fase	Iterazione	Predecessore
1	Ideazione	Iterazione 1	Nessuna
2	Elaborazione 1°	Iterazione 1	Nessuna
3	Costruzione 1°	Iterazione 1	Nessuna
4	Transizione 1°	Iterazione 1	Nessuna
5	Elaborazione 2°	Iterazione 2	Iterazione 1
6	Costruzione 2°	Iterazione 2	Iterazione 1
7	Transizione 2°	Iterazione 2	Iterazione 1

1.4 Fabbisogno delle Risorse

ID	Fase	Iterazione	Risorse Software
1	Ideazione	Iterazione 1	<i>PowerPoint, Microsoft Word, Web Browser</i>
2	Elaborazione 1°	Iterazione 1	<i>Visual Paradigm, Eclipse, XAMP, Apache Server Tomcat, Web Browser, Blocco Note, Microsoft Word, GitHub</i>
3	Costruzione 1°	Iterazione 1	<i>Visual Paradigm, Eclipse, XAMP, Apache Server Tomcat, Web Browser, Blocco Note, Microsoft Word, GitHub</i>
4	Transizione 1°	Iterazione 1	<i>Visual Paradigm, Eclipse, XAMP, Apache Server Tomcat, Web Browser, Blocco Note, PowerPoint, Microsoft Word</i>
5	Elaborazione 2°	Iterazione 2	<i>Visual Paradigm, Eclipse, XAMP, Apache Server Tomcat, Web Browser, Blocco Note, Microsoft Word, GitHub</i>
6	Costruzione 2°	Iterazione 2	<i>Visual Paradigm, Eclipse, XAMP, Apache Server Tomcat, Web Browser, Blocco Note, Microsoft Word</i>
7	Transizione 2°	Iterazione 2	<i>Visual Paradigm, Eclipse, XAMP, Apache Server Tomcat, Web Browser, Blocco Note, Microsoft Word, GitHub</i>

Capitolo 2: Documenti e Modelli d'avvio

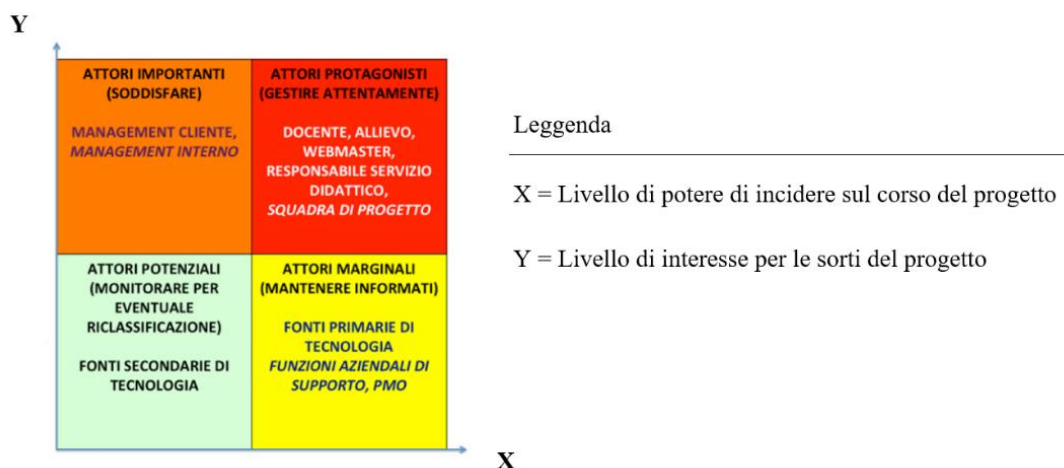
2.1 Requisiti generali

Il sistema software deve realizzare la gestione automatizzata di prenotazione di piazzole ed escursioni di un campeggio. Il sistema deve consentire agli utenti di poter prenotare sia le piazzole che le escursioni ed all'amministratore di poter monitorare le prenotazioni e la situazione economica corrente del camping. Per cui, i *requisiti generali* del nostro sistema sono:

- consentire agli utenti di prenotare on-line un soggiorno al Camping di Villetta Barrea scegliendo una o più piazzole di qualsiasi tipologia, in base alle esigenze dell'utente
- consentire agli utenti di prenotare *online* delle escursioni guidate di diverse tipologie per una o più persone
- consentire all'amministratore del Camping di visualizzare i dati delle prenotazioni relative a piazzole ed a escursioni
- consentire all'amministratore del Camping di monitorare gli incassi totali, gli incassi associati alle prenotazioni delle piazzole e gli incassi alle prenotazioni delle escursioni

2.2 Parti interessate

Definiamo ora le *parti interessate*, ovvero gli individui o dei gruppi che hanno, o si aspettano, proprietà, diritti o interessi nei confronti di bookCamping e delle sue attività, il cui contributo è essenziale per il raggiungimento degli obiettivi dell'organizzazione.



Tra gli *stakeholders* di bookCamping, riconosciamo:

1. i **proprietari** del Camping che, secondo la tabella, rivestono il ruolo di Attori Protagonisti
2. i **dipendenti** del Camping tra cui gli **escursionisti** esperti e l'**amministratore** del Camping che rivestono il ruolo di Attori Protagonisti
3. i **clienti** del Camping che rivestono il ruolo di Attori Importanti
4. gli **utenti finali** dell'Applicazione Web che rivestono il ruolo di Attori Protagonisti

2.3 Assunzioni

Le *assunzioni* concettuali riguardanti il comportamento di *bookCamping* sono state concepite in parte durante la fase iniziale di *Ideazione* ed in parte durante le successive iterazioni. Quest'ultime sono elencate di seguito:

1. si suppone che per effettuare una prenotazione di qualsiasi genere sia necessario essersi prima *registrati* e successivamente identificati attraverso il *login*
2. si suppone che l'utente al momento della prenotazione di una piazzola selezioni un soggiorno che includa almeno una notte
3. si suppone che l'amministratore sia unico e che gli siano assegnate delle credenziali univoche e non modificabili se non dagli sviluppatori del sistema
4. si suppone che le piazzole e le escursioni siano di un numero ben preciso, che siano stabilite a priori e che non siano modificabili dall'amministratore ma solo dagli sviluppatori del sistema
5. si suppone che gli utenti abbiano raggiunto almeno la maggiore età per poter effettuare la registrazione, in caso contrario il sistema non consente né la registrazione tanto meno la prenotazione
6. si suppone che il pagamento debba essere effettuato esclusivamente *on-line*

2.4 Vincoli di carattere generale

Il progetto in realtà presenta delle *limitazioni*, dei vincoli che sarebbe stato possibile oltrepassare solo attraverso ulteriori iterazioni di lavoro. In particolare, ho riscontrato che sarebbe stato necessario implementare ulteriori funzionalità per poter effettivamente lanciare questo prodotto software in ambito commerciale.

Degli esempi di funzionalità che avremmo potuto implementare per rendere ancora più completo il sistema sono le seguenti:

- cancellare le prenotazioni relative a date passate per non intasare la bacheca dell'amministratore e spostarle in uno *storico* in modo tale da averne sempre traccia
- consentire all'utente, come anche all'amministratore, di poter modificare ogni informazione relativa al proprio profilo tra cui l'immagine di riconoscimento
- consentire all'utente di poter prenotare più piazzole contemporaneamente, attraverso un'unica prenotazione
- consentire all'utente di poter cancellare la propria prenotazione
- consentire all'amministratore di modificare, aggiungere o eliminare le escursioni e le loro caratteristiche
- prevedere che ogni prenotazione sia caratterizzata da ora e data di *avvenuta prenotazione* per consentire all'amministratore di capire quando sono avvenute le prenotazioni
- prevedere un accesso tramite *Applicazione Desktop* per gli amministratori

Capitolo 3: Specifica dei Requisiti

Si vuole realizzare un sistema software che consente agli **utenti**, caratterizzati da id, username, password, nome, cognome, data di nascita, indirizzo, residenza, cellulare ed e-mail di effettuare prenotazioni *on-line* per piazzole e per escursioni relativa ad un **Camping**.

Per quanto concerne la gestione automatizzata delle prenotazione delle **piazzole** di un camping, sappiamo che quest'ultimo è costituito da dieci piazzole, le quali sono caratterizzate da un identificativo e sono di differente tipologia:

- tre piazzole sono di tipologia Tenda Small e si estendono su un'area di *trenta metri quadri*
- tre piazzole sono di tipologia Tenda Large e si estendono su un'area di *quaranta metri quadri*
- due piazzole sono di tipologia Roulotte e si estendono su un'area di *sessanta metri quadri*
- due piazzole sono di tipologia Camper e si estendono su un'area di *settantadue metri quadri*

Ogni **prenotazione di una piazzola** necessita delle seguenti informazioni: id, data di check-in, data di check-out, numero di adulti, numero di bambini, costo totale, note e tre optional che sono posto auto, posto moto e corrente. Il costo totale di una prenotazione di una piazzola viene calcolato tenendo conto di una quota al giorno per persona ed una quota al giorno per postazione ed optional:

Tipologia Piazzola	Costo al giorno
Per Tenda Small	11 €
Per Tenda Large	18 €
Per Roulotte	22 €
Per Camper	24 €

Optional	Costo al giorno
Corrente	3 €
Posto Auto	5 €
Posto Moto	3 €

Persona	Costo al giorno
Adulto	13 €
Bambino (4-10)	10 €
Bambino (0-3)	Gratuito

Il sistema software, inoltre, deve realizzare la gestione di prenotazioni relative a delle **escursioni**. Le escursioni sono caratterizzate da: identificativo, nome, ora di inizio, ora di fine, guida esperta, prezzo, luogo di partenza, durata e capienza massima. Esse sono organizzate dal Camping ogni giorno e sono di due tipologie:

- Escursione guidata a cavallo il cui prezzo è quindici euro
- Trekking guidato il cui prezzo è dieci euro

Ogni **prenotazione di un'escursione** necessita delle seguenti informazioni: id, data, numero di persone e costo totale. Il costo totale tiene conto del costo della tipologia di escursione e del numero di persone che partecipano.

Il sistema software deve consentire agli utenti di effettuare il pagamento delle prenotazioni *on-line*. Infine, il sistema deve permettere ad un unico **amministratore**, identificabile attraverso username e password, di effettuare operazioni di consultazione delle prenotazioni effettuate e monitoraggio degli incassi relativi alle prenotazioni di escursioni, piazzole e del totale.

3.1 Specifica dei requisiti funzionali

Attraverso la descrizione dei *requisiti funzionali* del sistema, voglio definire in maniera concisa ed efficace quali sono i servizi che il sistema offre:

Nome Requisito	Registrazione Utente
ID	RF01
Funzione	Registrazione di un utente nel sistema
Descrizione	Consiste nel salvataggio dei dati relativi ad un utente
Input	Dati dell'utente
Output	Utente registrato e messaggio di conferma
Elaborazione	Verifica della Validità dei dati inseriti
Pre-condizioni	Utente Non registrato
Post-condizioni	Utente registrato

Nome Requisito	Prenotazione Piazzola
ID	RF02
Funzione	Prenotazione di una piazzola del Camping
Descrizione	Consiste nel consentire all'utente di prenotare un soggiorno presso il Camping
Input	Dati relativi alla prenotazione inseriti dall'utente
Output	Prenotazione Piazzola temporaneamente salvata e riepilogo
Elaborazione	Verifica della validità dei dati inseriti e della disponibilità del Camping
Pre-condizioni	Utente registrato e loggato
Post-condizioni	Salvataggio Temporaneo della Prenotazione

Nome Requisito	Prenotazione Escursione
ID	RF03
Funzione	Prenotazione di un'escursione organizzata dal Camping
Descrizione	Consiste nel consentire all'utente di prenotare un soggiorno presso il Camping
Input	Dati relativi alla prenotazione inseriti dall'utente
Output	Prenotazione Escursione temporaneamente salvata e riepilogo
Elaborazione	Verifica della validità dei dati inseriti e della disponibilità del Camping
Pre-condizioni	Utente registrato e loggato
Post-condizioni	Salvataggio Temporaneo della Prenotazione

Nome Requisito	Consulta Escursioni
ID	RF04
Funzione	Visualizzazione delle escursioni da poter prenotare
Descrizione	Consiste nel far visualizzare all'utente le varie escursioni da poter prenotare
Input	Nessuno
Output	Elenco delle escursioni
Elaborazione	Estrarre i dati associati alle escursioni
Pre-condizioni	Utente non necessariamente registrato o loggato
Post-condizioni	L'utente visualizza le escursioni

Nome Requisito	Consulta Prenotazioni
ID	RF05
Funzione	Visualizzazione di tutte le prenotazioni relative ad escursioni e piazzole
Descrizione	Consiste nel far visualizzare all'amministratore tutte le prenotazioni relative ad escursioni e piazzole
Input	Nessuno
Output	Elenco delle prenotazioni relative ad escursioni e piazzole
Elaborazione	Estrarre i dati associati a tutte le prenotazioni
Pre-condizioni	L'amministratore deve aver effettuato il login
Post-condizioni	L'amministratore visualizza tutte le prenotazioni

Nome Requisito	Monitoraggio degli Incassi
ID	RF06
Funzione	Visualizzazione degli incassi relativi al Camping, alle escursioni ed al totale
Descrizione	Consiste nel far visualizzare all'amministratore gli incassi relativi al Camping, alle escursioni ed al totale
Input	Nessuno
Output	Incassi relativi al Camping, alle escursioni ed al totale
Elaborazione	Calcolare la somma dei costi di tutte le prenotazioni effettuate relative ad escursioni e piazzole
Pre-condizioni	L'amministratore deve aver effettuato il login
Post-condizioni	L'amministratore visualizza gli incassi

Nome Requisito	Pagamento
ID	RF07
Funzione	Pagamento di una quota pari al costo delle prenotazioni
Descrizione	Consiste nel consentire all'utente di poter pagare le prenotazioni effettuate
Input	Dati della carta di credito dell'utente
Output	Messaggio di avvenuto pagamento e registrazione della prenotazione
Elaborazione	Estrarre i dati associati alle escursioni
Pre-condizioni	L'utente che ha effettuato una prenotazione
Post-condizioni	La prenotazione viene registrata

3.2 Specifica dei requisiti non funzionali

Ho suddiviso i *requisiti non-funzionali*, ovvero di *qualità*, del progetto in:

- parametri *esterni*, ovvero quei parametri che misurano la qualità del software così come è percepita dai suoi utenti
- parametri *interni*, ovvero quei parametri che misurano la qualità del software così come è percepita dagli sviluppatori.

Per quanto concerne i parametri esterni, il sistema presenta:

- **affidabilità**: il prodotto software preserva un alto livello di prestazione nel tempo
- **usabilità**: le interfacce proposte ai nostri utenti sono *user-friendly* e di facile utilizzo anche per i non esperti
- **robustezza**: il sistema si comporta in modo adeguato ad eventuali input non validi o ad imprevisti software o hardware esterni al sistema
- **persistenza**: l'utilizzo di un Database consente che le informazioni vengano salvate in maniera *persistente* su uno spazio ad esse riservato

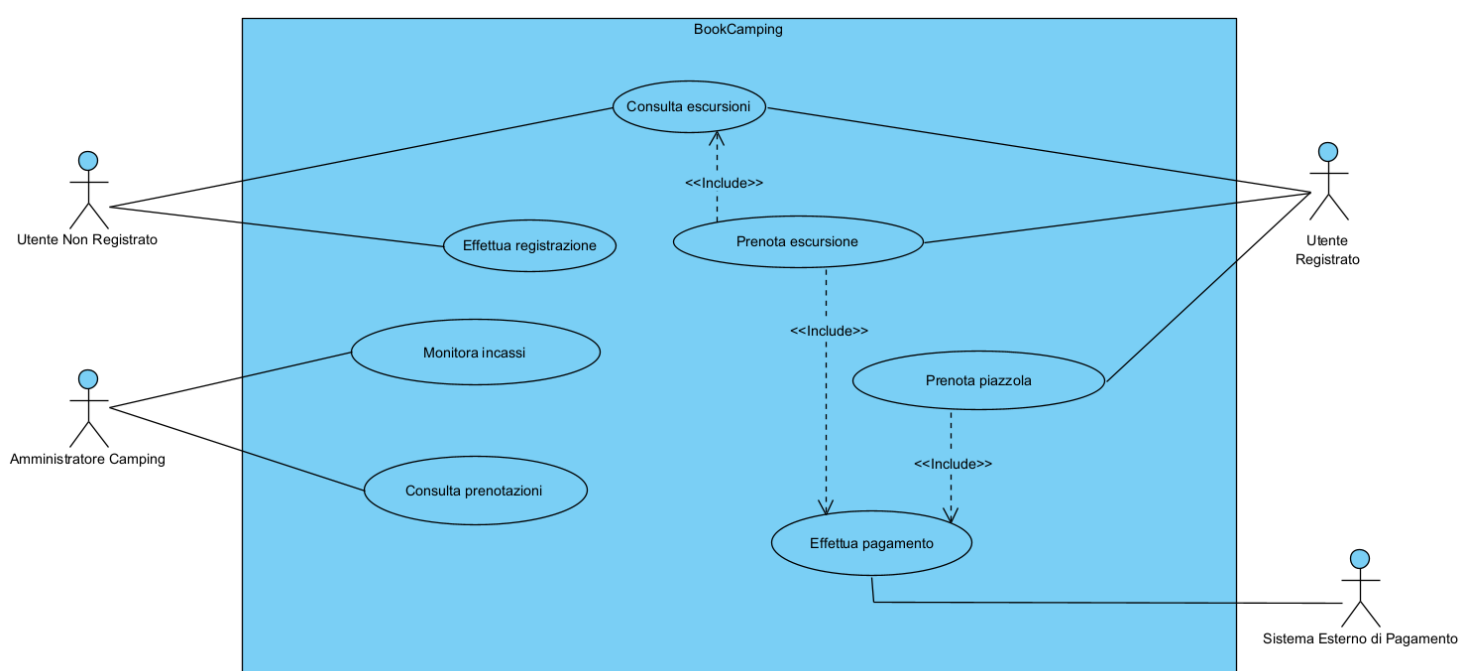
Per quanto concerne i parametri interni, il sistema presenta:

- **portabilità**: le semplici tecnologie alla base della realizzazione di questo sistema consentono al software di essere facilmente sottoposto al *porting*, cioè di essere utilizzato su piattaforme diverse da quella originale
- **manutenibilità**: il sistema è facilmente ripristinabile, qualora sia necessario effettuare un intervento di manutenzione
- **modificabilità**: il software possiede la capacità di consentire lo sviluppo di modifiche al software originale in maniera ottimale nel caso ce ne fosse bisogno
- **riusabilità**: gran parte del codice sorgente relativo a questa implementazione potrà essere riutilizzata in altri progetti software basati sulle stesse tecnologie e sugli stessi modelli progettuali
- **modularità**: il prodotto software è stato implementato attraverso un alto numero di moduli dipendenti tra loro dello stesso package affinché il sistema aumenti il grado di coesione

Capitolo 4: Analisi dei requisiti

4.1 Use Cases Diagram

Il seguente diagramma realizzato utilizzando *Visual Paradigm* definisce quali sono i Casi d'Uso relativi al sistema implementato. Attraverso questa tecnica usata nei processi dell'ingegneria del software garantisco una raccolta coincisa ed esaustiva dei requisiti al fine di produrre un software di qualità.



Elenco attori:

- Utente Registrato: utente che ha effettuato almeno una volta la registrazione
- Utente Non Registrato: utente che non ha mai effettuato la registrazione
- Amministratore Camping: utente che provvede alla gestione delle prenotazioni
- Sistema Esterno di Pagamento: sistema che consente di effettuare il pagamento

Elenco casi d'uso:

- Effettua Registrazione
- Prenota Escursione
- Consulta Escursione
- Prenota Piazzola
- Effettua Pagamento
- Monitora Incassi
- Consulta Prenotazioni

4.2 Descrizione dei Casi D'Uso

Per raggiungere una completa e non ambigua analisi dei requisiti necessitiamo di una descrizione più profonda di ogni caso d'uso elencato.

Effettua registrazione

- **Portata:** Applicazione Book Camping
- **Livello:** Obiettivo utente
- **Attore Primario:** Utente Non Registrato
- **Parti interessate:** Utente (vuole registrarsi)
- **Pre-condizioni:** L'utente non si è ancora registrato.
- **Garanzia di successo:** La registrazione viene effettuata.
- **Scenario Principale di successo:**
 1. L'utente inserisce i propri dati.
 2. Il sistema verifica i dati inseriti.
 3. Il sistema registra l'utente.
- **Flussi alternativi:**
 - 2.a) L'utente non inserisce tutti i campi obbligatori.
 1. Il sistema segnala l'errore e richiede i dati all'utente.
 - 2.b) L'utente inserisce una password con meno di nove caratteri.
 1. Il sistema segnala l'errore e richiede nuovamente i dati all'utente.
 - 2.c) L'utente inserisce una password con meno di due cifre.
 1. Il sistema segnala l'errore e richiede nuovamente i dati all'utente.
 - 2.d) L'utente inserisce una password con simboli.
 1. Il sistema segnala l'errore e richiede nuovamente i dati all'utente.
 - 2.e) L'utente inserisce le due password ma non combaciano.
 1. Il sistema segnala l'errore e richiede nuovamente i dati all'utente.
 - 2.f) L'utente inserisce uno username già esistente.
 1. Il sistema segnala l'errore e richiede nuovamente i dati all'utente.
 - 2.g) L'utente inserisce una e-mail non valida.
 1. Il sistema segnala l'errore e richiede nuovamente i dati all'utente.
 - 2.h) L'utente inserisce un cellulare non valido.
 1. Il sistema segnala l'errore e richiede nuovamente i dati all'utente.
 - 2.i) L'utente inserisce una data di nascita successiva alla data odierna.
 1. Il sistema segnala l'errore e richiede nuovamente i dati all'utente.
 - 2.l) L'utente non ha diciotto anni .
 1. Il sistema segnala l'errore e richiede nuovamente i dati all'utente.

Consulta escursioni

- **Portata:** Applicazione Book Camping
- **Livello:** Obiettivo Utente
- **Attore Primario:** Utente Registrato o Utente Non Registrato
- **Parti interessate:** Utente (vuole consultare l'elenco delle escursioni disponibili)
- **Garanzia di successo:** L'elenco delle escursioni viene visualizzato dall'utente.
- **Scenario Principale di successo:**
 1. L'utente richiede di visualizzare le escursioni disponibili.
 2. Il sistema mostra l'elenco delle escursioni.

Effettua login

- **Portata:** Applicazione Book Camping
- **Livello:** Sottofunzione
- **Attore Primario:** Utente Registrato
- **Parti interessate:** Utente (vuole effettuare il login)
- **Pre-condizioni:** L'utente è registrato.
- **Garanzia di successo:** Il login viene effettuato.
- **Scenario Principale di successo:**
 1. L'utente inserisce username e password.
 2. Il sistema verifica i dati inseriti.
 3. Il sistema identifica l'utente.
- **Flussi alternativi:**
 - 2.a) L'utente non inserisce tutti i campi obbligatori.
 1. Il sistema segnala l'errore e richiede i dati all'utente.
 - 2.b) L'utente inserisce username o password non validi.
 1. Il sistema segnala l'errore e richiede nuovamente i dati all'utente.

Prenota escursione

- **Portata:** Applicazione Book Camping
- **Livello:** Obiettivo Utente
- **Attore Primario:** Utente Registrato
- **Attore secondario:** Sistema di pagamento esterno
- **Parti interessate:** Utente (vuole prenotare un'escursione), sistema di pagamento esterno
- **Pre-condizioni:** L'utente ha effettuato il login.
- **Garanzia di successo:** La prenotazione viene registrata.
- **Scenario Principale di successo:**
 1. L'utente seleziona escursione, giorno e numero di persone.
 2. Il sistema verifica la validità dei dati e controlla se l'escursione è disponibile nel giorno indicato.
 3. Il sistema memorizza le informazioni associate alla prenotazione e mostra il riepilogo con prezzo totale.
 4. L'utente effettua il pagamento con sistema esterno.
 5. Il sistema notifica che il pagamento è avvenuto con successo.
- **Flussi Alternativi:**
 - 2.a) L'utente non inserisce tutti i campi obbligatori.
 1. Il sistema segnala l'errore e richiede i dati all'utente.
 - 2.b) L'utente inserisce una data precedente alla data odierna.
 1. Il sistema segnala l'errore e chiede di reinserire i dati all'utente.
 - 2.c) L'escursione ha raggiunto il numero massimo di persone.
 1. Il sistema segnala errore e chiede il reinserimento dei dati.

Prenota piazzola

- **Portata:** Applicazione Book Camping
- **Livello:** Obiettivo Utente
- **Attore Primario:** Utente Registrato
- **Attore secondario:** Sistema di pagamento esterno
- **Parti interessate:** Utente (vuole prenotare una piazzola), sistema di pagamento esterno
- **Pre-condizioni:** L'utente ha effettuato il login.
- **Garanzia di successo:** La prenotazione viene registrata.
- **Scenario Principale di successo:**
 1. L'utente inserisce il periodo, la tipologia della piazzola, il numero di adulti e di bambini e delle scelte opzionali.
 2. Il sistema verifica la validità dei dati e controlla la disponibilità delle piazzole nel periodo indicato.
 3. Il sistema memorizza le informazioni associate alla prenotazione e mostra il riepilogo con prezzo totale.
 4. L'utente esegue Effettua Pagamento attraverso Sistema esterno di pagamento.
 5. Il sistema notifica che il pagamento è avvenuto con successo.
- **Flussi Alternativi:**
 - 2.a) L'utente non inserisce tutti i campi obbligatori.
 1. Il sistema segnala l'errore e richiede i dati all'utente.
 - 2.b) L'utente inserisce la data di check-in successiva al check-out.
 1. Il sistema segnala l'errore e chiede di reinserire i dati all'utente.
 - 2.c) L'utente inserisce la data di check-in precedente alla data odierna.
 1. Il sistema segnala l'errore e chiede di reinserire i dati all'utente.
 - 2.d) L'utente inserisce la data di check-in pari alla data di check-out.
 1. Il sistema segnala l'errore e chiede di reinserire i dati all'utente.
 - 2.e) L'utente sceglie un periodo in cui non ci sono piazzole libere.
 1. Il sistema segnala l'errore e chiede di reinserire i dati all'utente.

Consulta prenotazioni

- **Portata:** Applicazione Book Camping
- **Livello:** Obiettivo Utente
- **Attore Primario:** Amministratore
- **Parti interessate:** Amministratore (vuole visualizzare le prenotazioni effettuate)
- **Pre-condizioni:** L'amministratore ha effettuato il login.
- **Garanzia di successo:** Vengono mostrati i dettagli delle prenotazioni *online* effettuate presso il campeggio.
- **Scenario Principale di successo:**
 1. L'amministratore richiede di visualizzare l'elenco delle prenotazioni relative alle prenotazioni delle piazzole e delle escursioni.
 2. Il sistema recupera le informazioni e le mostra quindi all'amministratore

Monitora incassi

- **Portata:** Applicazione Book Camping
- **Livello:** Obiettivo Utente
- **Attore Primario:** Amministratore
- **Parti interessate:** Amministratore (vuole visualizzare i profitti del campeggio)
- **Pre-condizioni:** L'amministratore ha effettuato il login.
- **Garanzia di successo:** Viene mostrato il totale degli incassi del campeggio.
- **Scenario Principale di successo:**
 1. L'amministratore richiede di visualizzare gli incassi dovuti alle prenotazioni del campeggio, alle prenotazioni delle escursioni ed al totale.
 2. Il sistema recupera le informazioni relative a tutte le prenotazioni registrate presso il campeggio.
 3. Il sistema calcola l'ammontare degli incassi.
 4. Il sistema mostra all'amministratore l'incasso totale.

Effettua pagamento

- **Portata:** Applicazione Book Camping
- **Livello:** Obiettivo Utente
- **Attore Primario:** Utente Registrato
- **Attore secondario:** Sistema di pagamento esterno
- **Parti interessate:** Utente (vuole prenotare una piazzola), sistema di pagamento esterno, amministratore
- **Pre-condizioni:** L'utente ha effettuato una prenotazione.
- **Garanzia di successo:** Il pagamento viene effettuato con successo.
- **Scenario Principale di successo:**
 1. L'utente inserisce i dati bancari.
 2. Il sistema di pagamento esterno verifica i dati.
 3. Il sistema esterno conferma che il pagamento è avvenuto con successo.
- **Flussi alternativi:**
 - 2.a) L'utente non inserisce tutti i campi obbligatori.
 1. Il sistema segnala l'errore e richiede i dati all'utente.
 - 2.b) L'utente inserisce CVV e Card Number che non hanno un formato valido.
 1. Il sistema segnala l'errore e richiede i dati all'utente.
 - 2.c) L'utente inserisce una carta di credito scaduta.
 1. Il sistema segnala l'errore e richiede i dati all'utente.
 - 2.d) L'utente inserisce una carta di credito associata a circuiti di credito non riconosciuti.
 1. Il sistema segnala l'errore e richiede i dati all'utente.

4.3 Stima Costi dei Casi D'Uso

Per stimare i costi utilizzo la metrica degli **Use Case Points**, basata sul concetto di complessità dei casi d'uso. Un caso d'uso complesso necessiterà di maggiore tempo per la sua implementazione.

Gli UCP dipendono dal numero e dalla complessità dei casi d'uso, dagli attori e dai requisiti non-funzionali. Per cui, per il calcolo degli UCP sono necessari: **Unadjusted Use Case Point**, **Technical Complexity Factor**, **Environmental Factor**.

Per calcolare gli Unadjusted Use Case Point sono necessari due ulteriori parametri: *Unadjusted Use Case Weight* e *Unadjusted Actors Weight*.

Sapendo che:

Complessità	Numero Transizioni	Peso
Facile	3 o meno	5
Medio	4-7	10
Complesso	8 o più	15

associo ad ogni caso d'uso un grado di complessità

Caso d'uso	Complessità
Effettua registrazione	Complesso
Consulta escursione	Facile
Effettua login	Medio
Prenota escursione	Complesso
Prenota piazzola	Complesso
Consulta prenotazioni	Facile
Monitora incassi	Medio
Effettua Pagamento	Medio

e calcolo **Unadjusted Use Case Weight**.

Complessità	Peso	Numero Casi d'uso	Prodotto
Facile	5	2	10
Medio	10	3	30
Complesso	15	3	45
Totale			85

Sapendo che:

Complessità	Tipo di attore	Peso
Semplice	Sistema esterno che interagisce usando una API ben definita	1
Medio	Sistema esterno che interagisce un protocollo standard di comunicazione	2
Complesso	Attore umano che interagisce tramite interfaccia GUI	3

associo ad ogni caso d'uso un grado di complessità

Attore	Complessità
Utente registrato	Complesso
Utente non registrato	Complesso
Amministratore	Complesso
Sistema di pagamento esterno	Semplice

e calcolo **Unadjusted Actor Weight**.

Tipo di attore	Peso	Numero Attori	Prodotto
Semplice	1	1	1
Medio	2	0	0
Complesso	3	3	9
Totale			10

Ora è possibile calcolare gli *UUCP* come: $UUCP = UUCW + UAW = 85 + 10 = 95$

Tuttavia, come si può immaginare, gli UCP non dipendono solo dai casi d'uso, dagli attori e dalla loro complessità, bensì dipendono anche da vari requisiti non-funzionali che non sono stati scritti come casi d'uso. L'impatto della complessità tecnica è valutato attraverso tredici fattori a cui viene dato un punteggio da 0 a 5. Questo parametro viene chiamato **Technical Complexity Factor**.

Ogni fattore presenta un peso stabilito a priori che viene moltiplicato per la nostra valutazione (0-5) per ottenere l'impatto effettivo.

Fattori	Peso	Valutazione	Impatto
Sistema Distribuito	2	3	6
Obiettivi di performance	2	3	6
Efficienza end-user	1	3	3
Elaborazione complessa	1	2	2
Codice riusabile	1	2	2
Facile da installare	0.5	0	0
Facile da usare	0.5	4	2
Portabile	2	2	4
Facile da modificare	1	5	5
Uso concorrente	1	5	5
Sicurezza	1	5	5
Accesso a terze parti	1	3	3
Necessità di Training	1	0	0
Totale			43

Il totale viene chiamato **Tfactor = 43**.

A questo punto otteniamo il **TCF**.

$$\text{TCF} = 0.6 + (0.01 \times \text{Tfactor}) = 1.03$$

Un ultimo fattore per il calcolo degli UCP è un parametro chiamato **Environmental Factor** ed è influenzato da fattori come la motivazione, la familiarità al processo di sviluppo ecc...

Fattori	Peso	Valutazione	Impatto
Familiarità con il processo di sviluppo	1.5	2	3
Esperienza sull'applicazione	0.5	2	2
Esperienza sull'O-O	1	4	4
Capacità dell'analista	0.5	3	1.5
Motivazione	1	4	4
Requisiti stabili	2	3	6
Staff part-time	-1	0	0
Linguaggio di programmazione difficile	-1	2	-2
Totale			18.5

Il totale è chiamato EFactor. L' ET è il seguente **EF = 1.4 + (-0.03 x EFactor) = 0.85**.

Per il calcolo degli UCP bisogna effettuare il seguente calcolo:

$$\text{UCP} = \text{UUCP} \times \text{TCF} \times \text{EF} = 95 \times 1.03 \times 0.85 = 83,17$$

Questo punteggio è la stima della dimensione del progetto. Per poter stimare il tempo che si impiegherebbe a sviluppare UCP, basterebbe conoscere la velocità di sviluppo. In questo caso basta conoscere il numero di ore che sono state necessarie a realizzare il progetto. Avendo lavorato per un mese, in maniera assidua sette giorni su sette le ore totalizzate sono: **224**. Per cui, conoscendo il valore UCP prodotto, la mia media sarà di *circa tre ore per UCP*.

Capitolo 5: Architettura Software

In questo capitolo, descrivo l'architettura del software sviluppato definendo l'insieme degli *elementi architetturali* da cui è costituita (“*what*”), la particolare *forma* che struttura il sistema (“*how*”) e le *giustificazioni logiche* per la scelta degli elementi e della forma (“*why*”).

L'architettura software adottata per la *Web Application* bookCamping riprende la sua struttura di tipo *layered*, le sue caratteristiche relazionali e componentistiche da uno dei pattern architetturali maggiormente utilizzato per le applicazioni Web, il pattern **MVC**. La struttura principale prevede tre macro-moduli: **Model**, **View** e **Controller**. A questi tre è stato necessario aggiungerne un quarto per l'accesso indipendente ai dati del database, il **Data Access Object**.

La **motivazione** che mi ha spinto ad utilizzare il modello MVC è che necessitavo di una soluzione architetturale che mi consentisse di separare in maniera netta la *logica applicativa*, alla base del sistema, dalle numerose *interfacce utente* nelle quali vengono mostrati i dati del sistema.

Questo è stato un vantaggio poiché il software risulta in gran parte *riusabile* a causa dell'indipendenza che intercorre tra View, logica applicativa che risiede nel Controller e Model che rappresenta la struttura del modello di dominio.

Le **componenti** software che ho maggiormente sfruttato per definire l'architettura software del sistema sono:

- per la **View**, le *JavaServer Page*, anche abbreviate in *JSP*, le quali sono dei documenti HTML con estensione *.jsp* che includono codice Java. Le JSP sono elaborate dal *servlet container Tomcat* a seguito di richieste utente e consentono la generazione dinamica di contenuto HTML. Il loro compito architetturale nel nostro sistema è:
 - Mostrare i dati del Model
 - Inviare le azioni utente al Controller
 - Consentire che il Controller scelga la View
 - Richiedere dati aggiornati al Model
- per il **Controller**, le *Servlet*, particolari classi Java estensioni di classi *HttpServlet*, le quali operano all'interno di un server web, nel nostro caso Tomcat, e che consentono di ricevere e rispondere a delle richieste Http permettendo di interagire da un lato con le JSP e dall'altro con le classi Java appartenenti al Model. Il loro compito architetturale nel nostro sistema è:
 - Incapsulare la logica applicativa del sistema
 - Selezionare la View per fornire gli appropriati output all'utente
 - Mappare gli input dell'utente come modifiche al Model

- per il **Model**, delle *classi Java* che hanno compito descrivere tutte le entità in gioco definendo anche quelle che sono le loro caratteristiche e relazioni che intercorrono tra di esse. Il loro compito architetturale nel nostro sistema è:
 - Consentire di incapsulare lo stato dell'applicazione
 - Rispondere alle richieste sullo stato
 - Mostrare le funzionalità dell'applicazione
 - Fare in modo che la View abbia sempre dei dati aggiornati

Se l'*architettura prescrittiva* scelta è stata quella dell'MVC, l'*architettura descrittiva* è caratterizzata da alcune scelte di progetto che si distaccano dal classico e standard pattern MCV. Tuttavia, trattandosi di lievi modifiche, architettura prescrittiva e descrittiva risultano *consistenti* tra loro.

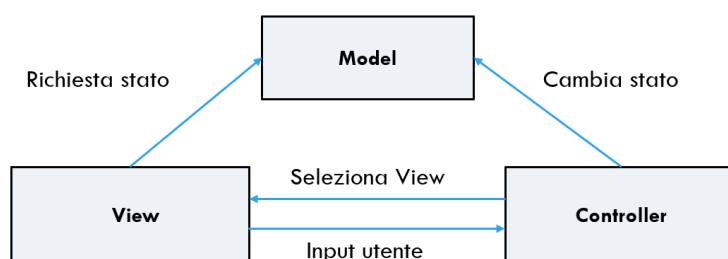
Una **scelta progettuale** riguarda il *passaggio dei parametri* tra Model e View che avviene tramite il Controller il quale, una volta acquisiti i dati dal Model, può consentire alla View di riceverli. In particolare, ad un certo punto viene stabilita una *sessione*, in questo caso di tipo *http*, che può coinvolgere più di un messaggio in ciascuna direzione. Il Controller invia dei dati attraverso la funzione `session.setAttribute(object, "object_name")` mentre la View riceve i dati attraverso il metodo `session.getAttribute("object_name")`.

Un'altra **scelta progettuale** intrapresa durante il processo di realizzazione del software rende l'architettura leggermente atipica rispetto al classico modello MVC; essa riguarda il fatto che il Model *non* ha la responsabilità di notificare ai componenti della View eventuali aggiornamenti verificatisi in seguito a richieste del Controller. La View si limita a richiedere quelli che sono i dati del Model tramite il Controller.

Ho attuato questa scelta perché dal momento che il sistema software è un Web Site, le View sono rappresentate da pagine Web dinamiche che vengono automaticamente aggiornate ad ogni processo di selezione di pagina da parte dell'utente, ovvero durante la *navigazione*.

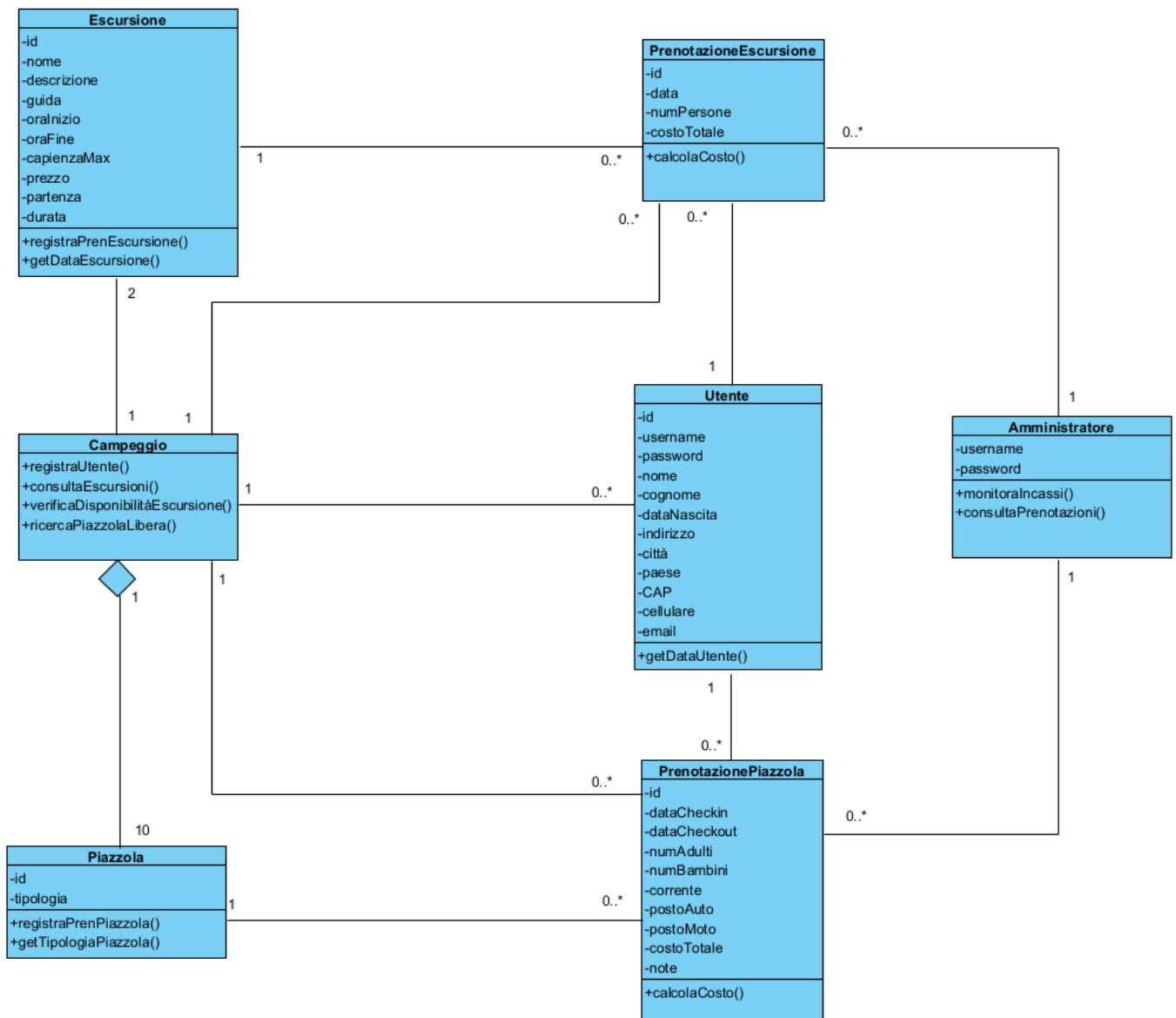
In caso di inattività, ovvero se l'utente si collega al Web Site ma non naviga tra le pagine, le View per essere aggiornate possono essere banalmente ricaricate dall'utente tramite gli appositi pulsanti messi a disposizione da qualsiasi Web Browser.

Per cui il modello generale della nostra architettura sarà il seguente:



5.1 System Domain Model

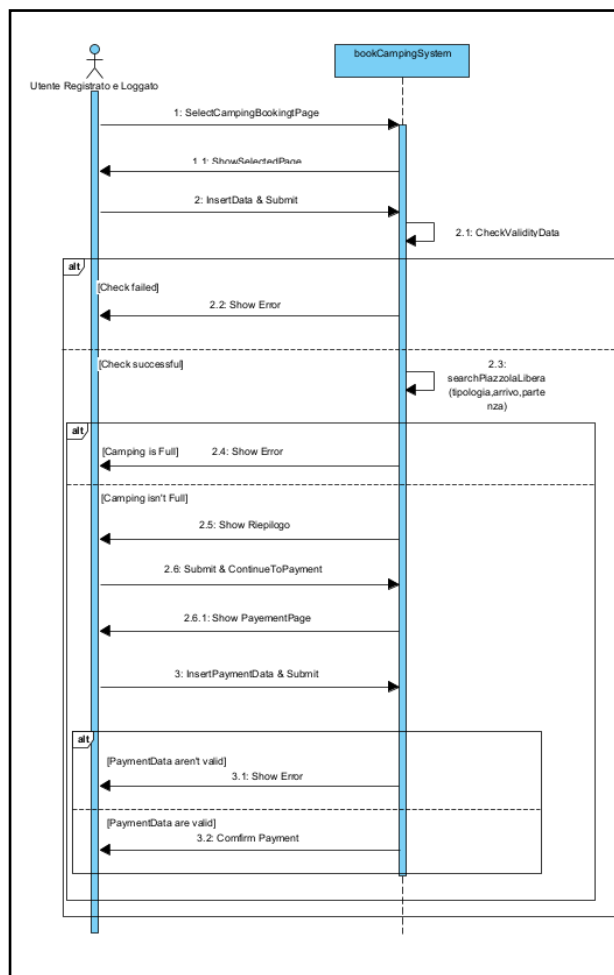
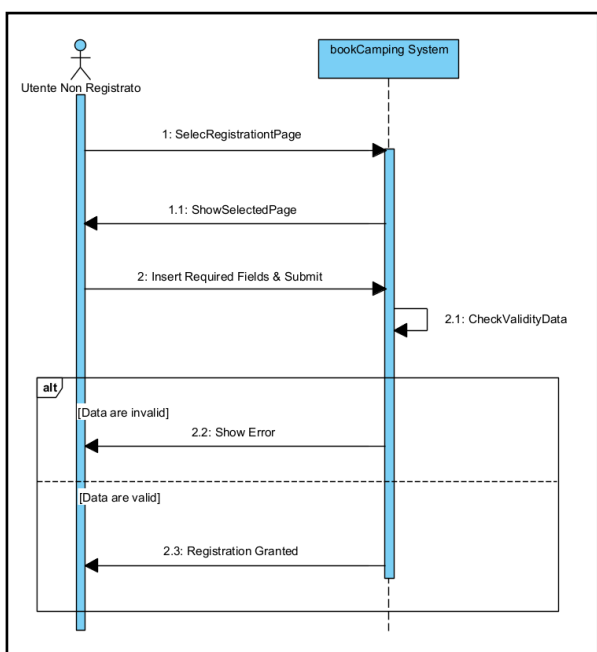
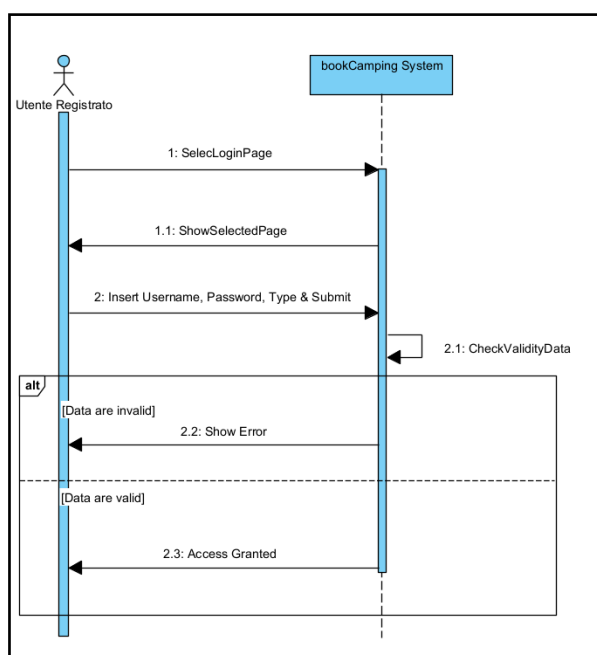
Una visione statica delle classi del Modello di Dominio e delle relazioni che intercorrono tra esse è data dal *System Domain Model*.



5.2 Sequence Diagram d'Analisi

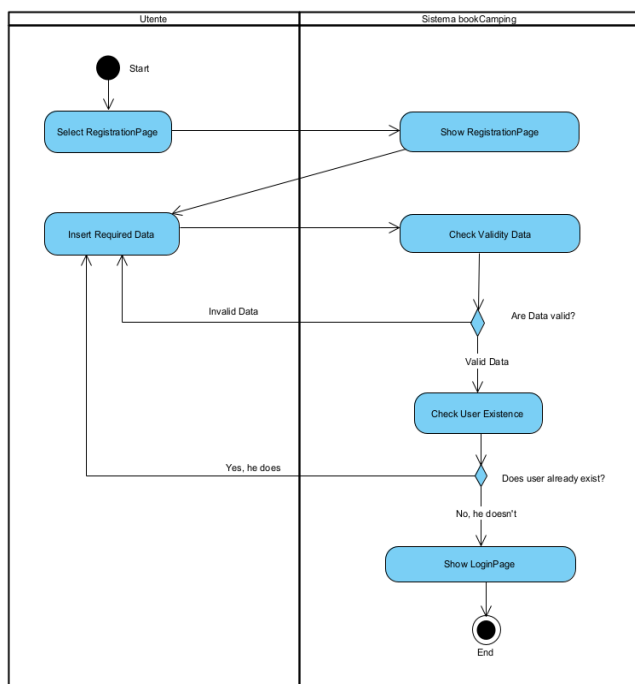
Un Sequence Diagram di analisi appartiene ai Behavioral Diagram e consente di riconoscere meglio lo scambio di battute nel tempo che intercorrono tra un attore ed una entità che rappresenta l'intero sistema.

In questo caso, sulla sinistra ho rappresentato il caso del *login* ed il caso della *registrazione*, mentre a destra il caso della *prenotazione di una piazzola + pagamento* leggermente più complesso e del tutto simile alla prenotazione di una escursione.



5.3 Activity Diagram con *Swimlanes*

Un Activity Diagram è un Behavioral Diagram e consente di riconoscere meglio il flusso di controllo da un punto di inizio ad un punto di fine. In ordine, sono stati rappresentati i casi di *registrazione*, *login* e *prenotazione di una escursione + pagamento*.

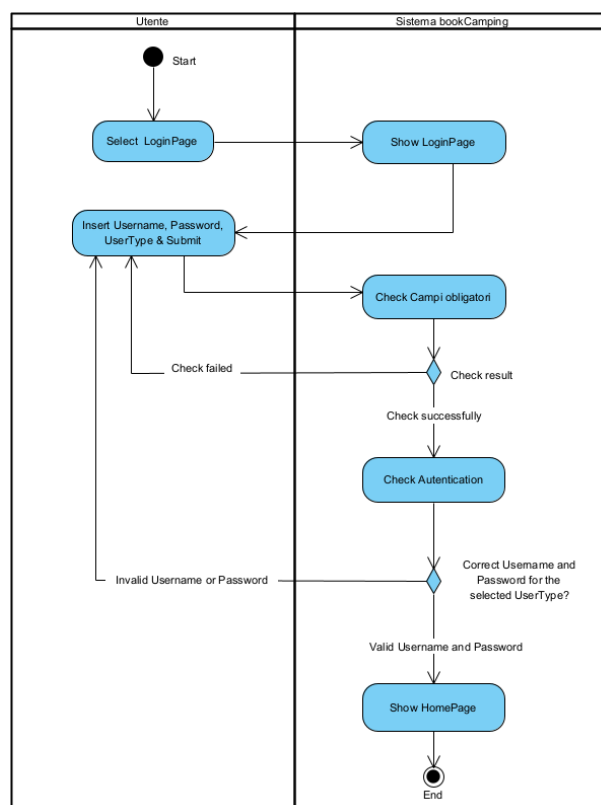


Le entità in gioco in questo caso sono:

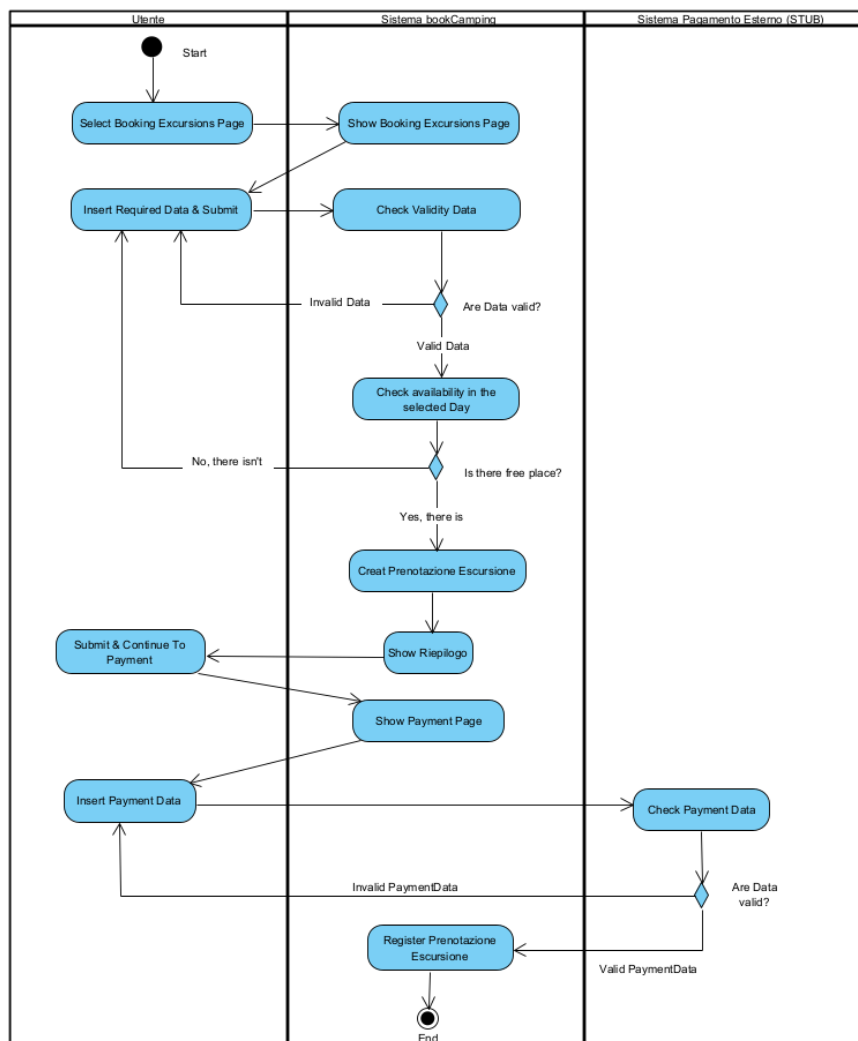
- Utente
- bookCamping

L'AD a destra rappresenta il caso di *login*. I nodi decisionali sono legati prevalentemente a quelli che sono i check di validazione del sistema che fanno in modo tale che il sistema presenti una retroazione.

L'AD a sinistra rappresenta il caso di *registrazione*. In particolare, ho utilizzato la tecnica dello *Swimlanes* secondo cui è possibile capire qual sia l'entità a dover effettuare quell'azione/attività. Ho partizionato lo spazio circostante ed ho inserito le diverse azioni nella sezione associata all'entità che la compie.



L'AD a destra rappresenta il caso di *prenotazione di una escursione + pagamento*. Per quest'ultimo caso sono state necessarie tre partizioni, una aggiuntiva associata al sistema di pagamento esterno che in realtà, viene simulata dal sistema stesso attraverso uno Stub.



5.4 Component Diagram

Il diagramma dei componenti UML è un diagramma che ha lo scopo di rappresentare la struttura interna del sistema software modellato in termini dei suoi **componenti** principali e delle relazioni fra di essi. I componenti sono delle unità software dotate di una precisa identità, responsabilità e interfacce ben definite.



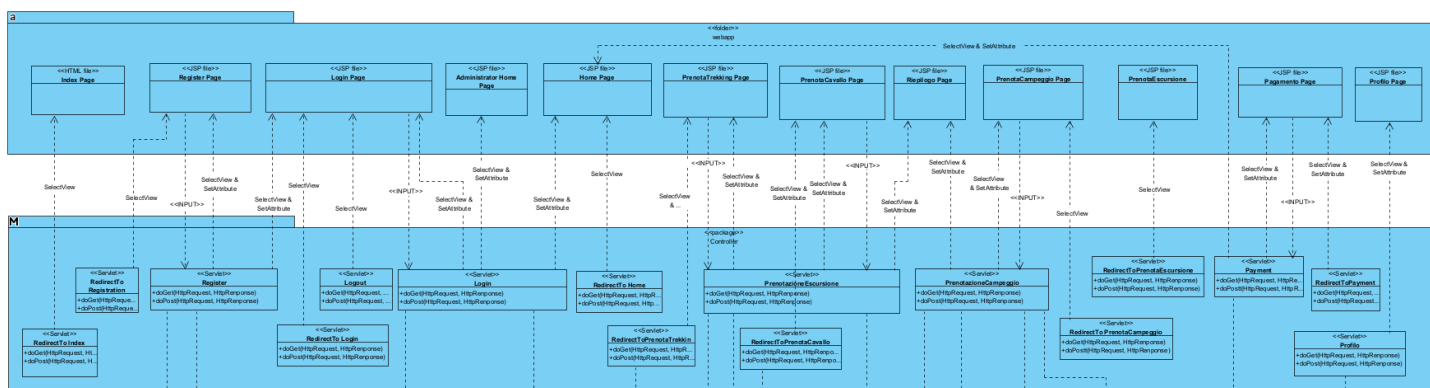
Il componente principale in tal caso è *bookCamping* il quale:

- offre servizio tramite interfaccia al Web Browser che avanza richieste *http*
- usufruisce dei servizi offerti dal Database

5.5 Package Diagram

Il Package Diagram consente di avere una visualizzazione statica del sistema in maniera molto dettagliata. Esso rappresenta dei contenitori, chiamati *Package*, che contengono classi o altre tipologie di file, ed esprime le relazioni in termini di dipendenze che intercorrono tra le differenti entità. Il diagramma, sviluppato interamente in *Visual Paradigm*, presenta un gran numero di elementi a causa della sua forte modularità e quindi ho deciso, per chiarezza, di rappresentare il grafico suddividendolo in tre sezioni, in ciascuna delle quali è possibile visualizzare le dipendenze tra un package e l'altro.

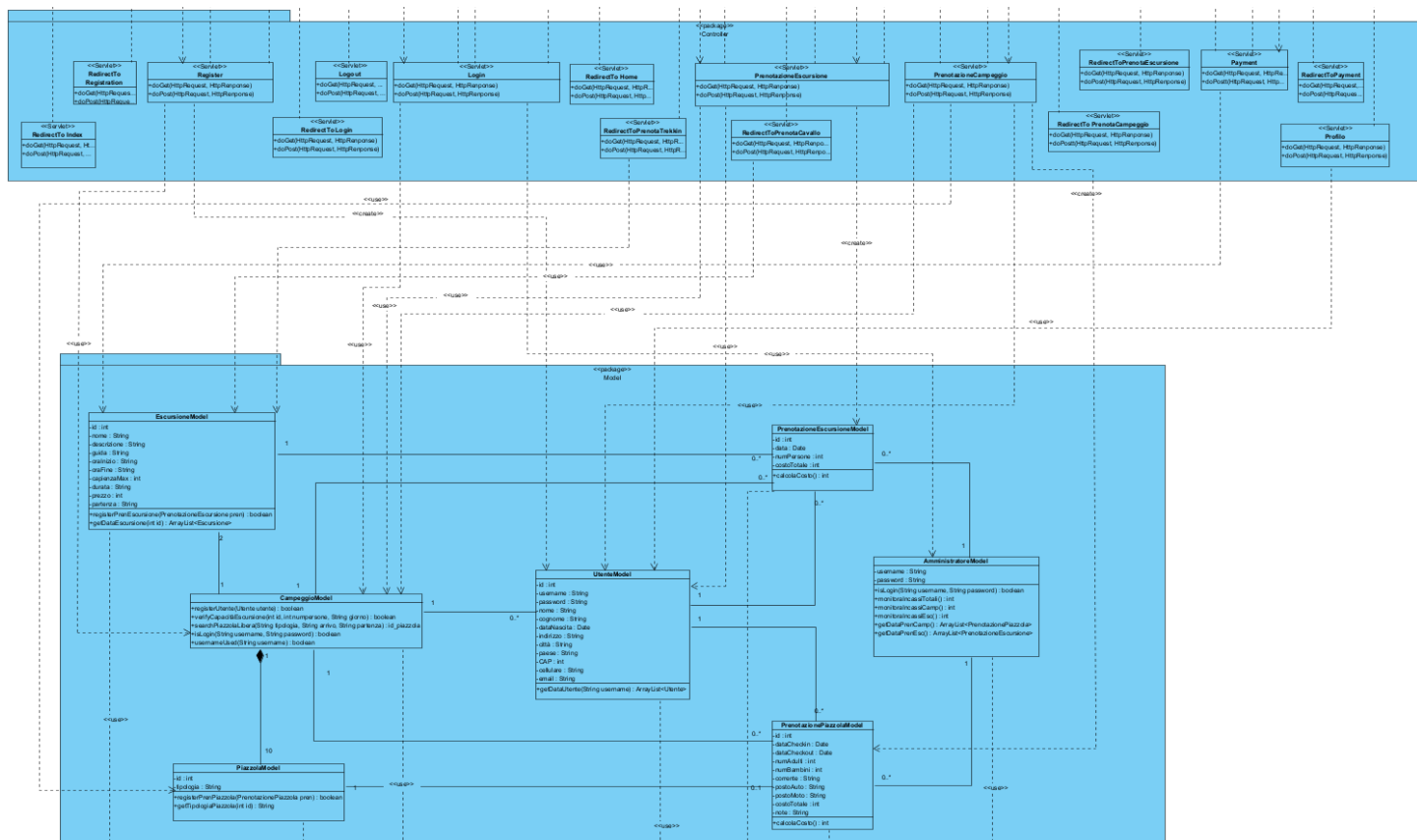
- Rapporto View-Controller



Osservazioni:

- la sezione di diagramma soprastante vuole rappresentare le entità ma anche le dipendenze che intercorrono tra View e Controller in termini di:
 1. `<<SelectView>>`: processi di selezione della View da parte del Controller
 2. `<<setAttribute>>`: passaggi dei parametri che avviene attraverso la creazione di una sessione tra View e Controller, Controller “setta” gli attributi e la View effettuando il “`getAttribute`” accede ai dati di cui necessita per mostrarli all’utente
 3. `<<getInput>>`: mappature degli input inseriti dall’utente tramite View al Controller.
- mentre il Controller è un **package**, la View raccoglie tutti i *file JSP o Html* in una *folder* di nome **webapp** creata di default da Eclipse
- il fatto che il Controller presenti un gran numero di classi è per rendere la struttura più *modulare* possibile, andando ad associare una sola funzionalità ad ogni singolo modulo

• Rapporto Controller-Model

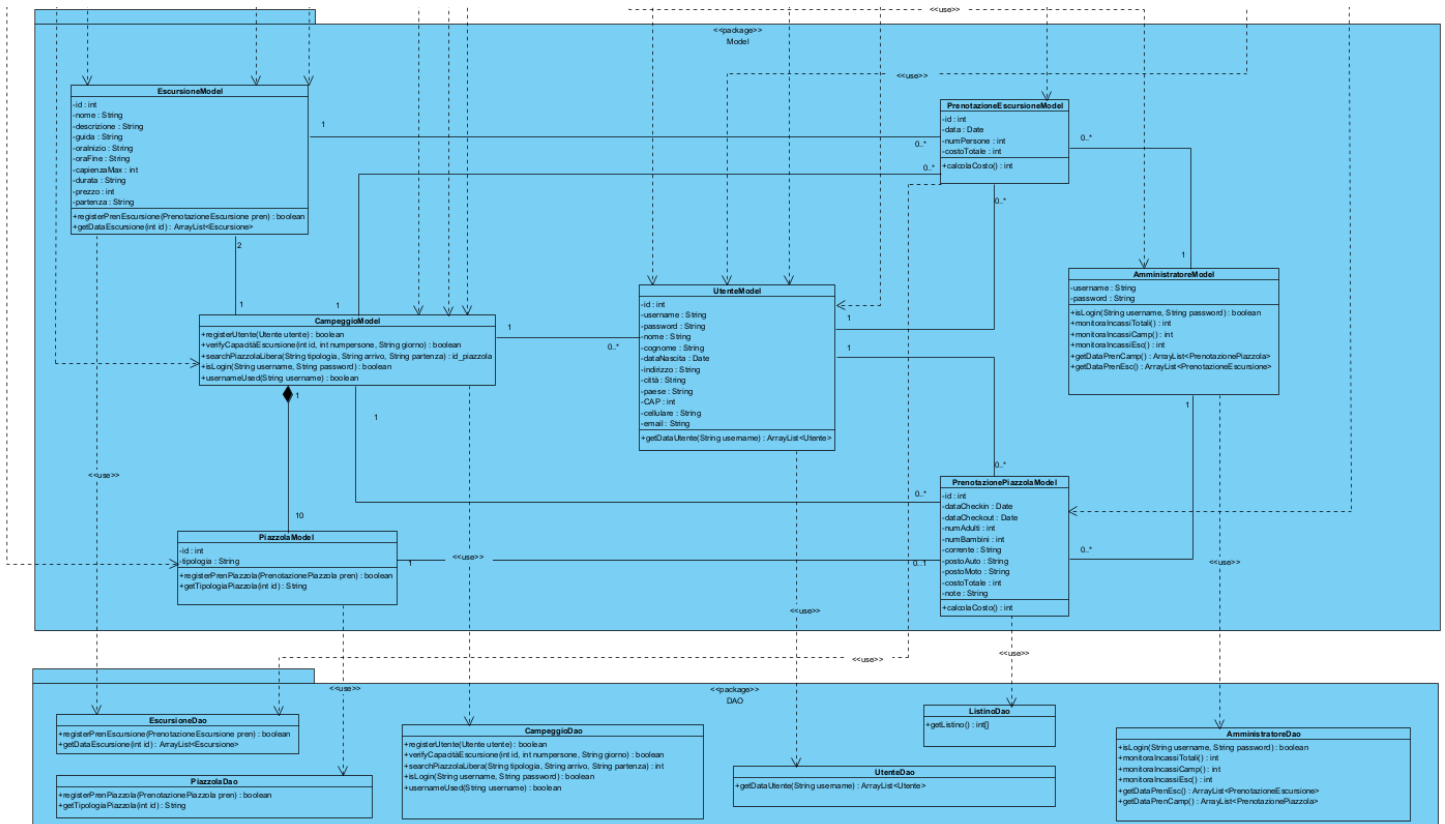


Osservazioni:

- le dipendenze tra le classi del Model e le classi del Controller sono prevalentemente di tipo `<<use>>` e di tipo `<<create>>`
- le classi che detengono il numero più elevato di responsabilità sono:
 1. *model.Campeggio*
 2. *model.Utente*
 3. *model.Ammministratore*
- le classi che invece presentano un alto grado di accoppiamento con le classi Controller sono:
 1. *model.Campeggio*
 2. *model.Utente*

poiché presentano delle funzionalità molto richieste dal sistema come, nel caso del *model.Utente*, la richiesta dei dati dell'utente, utile per il profilo, il riepilogo della prenotazione ed ogni qual volta fosse necessario l'ID o lo username dell'utente, oppure, nel caso del *model.Campeggio*, la richiesta di verificare se un'escursione ha raggiunto capienza massima o esiste una piazzola disponibile sulla base di determinati parametri.

- **Rapporto Model-Dao**



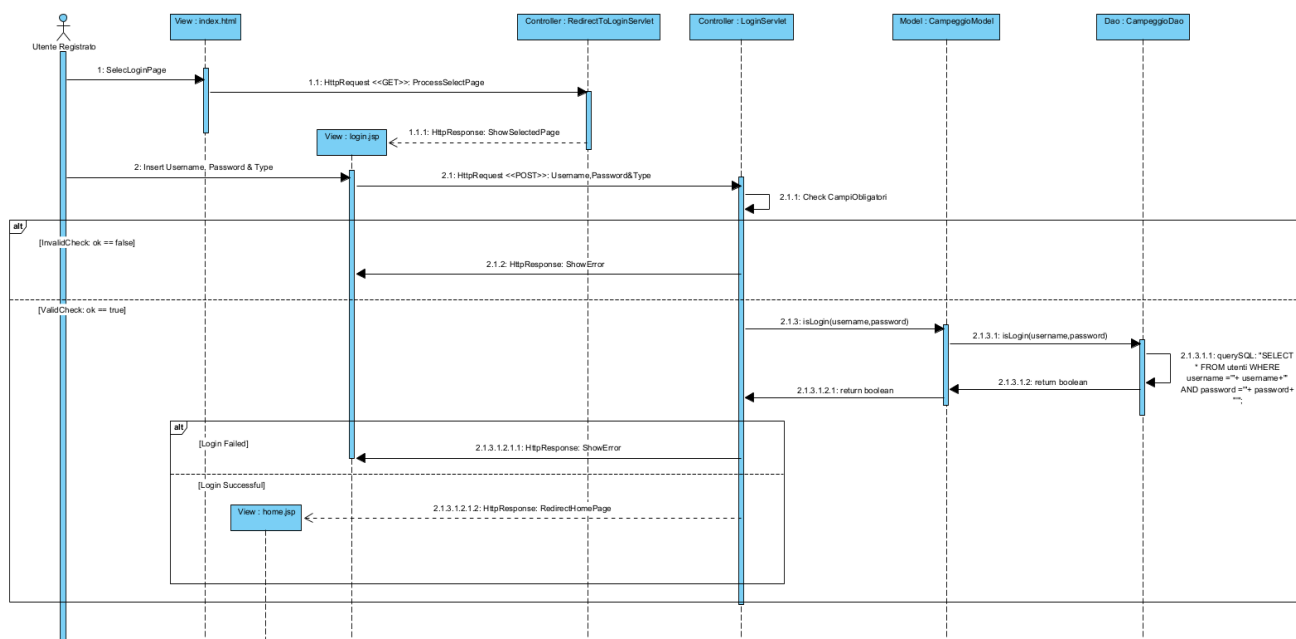
Osservazioni:

- i metodi delle classi del Model, nella maggior parte dei casi, si limitano semplicemente a richiamare metodi di classi appartenenti al package Data Access Object
- le classi del Model in realtà presentano metodi costruttori, *getters* e *setters* che non sono state inserite per rendere il diagramma più leggibile
- usufruisco del package Dao per effettuare operazioni **C.R.U.D.** all'interno del Database in modo tale da rendere indipendente il resto del sistema da eventuali problemi con il Database
- questa indipendenza si traduce in “alta coesione” e “basso accoppiamento”

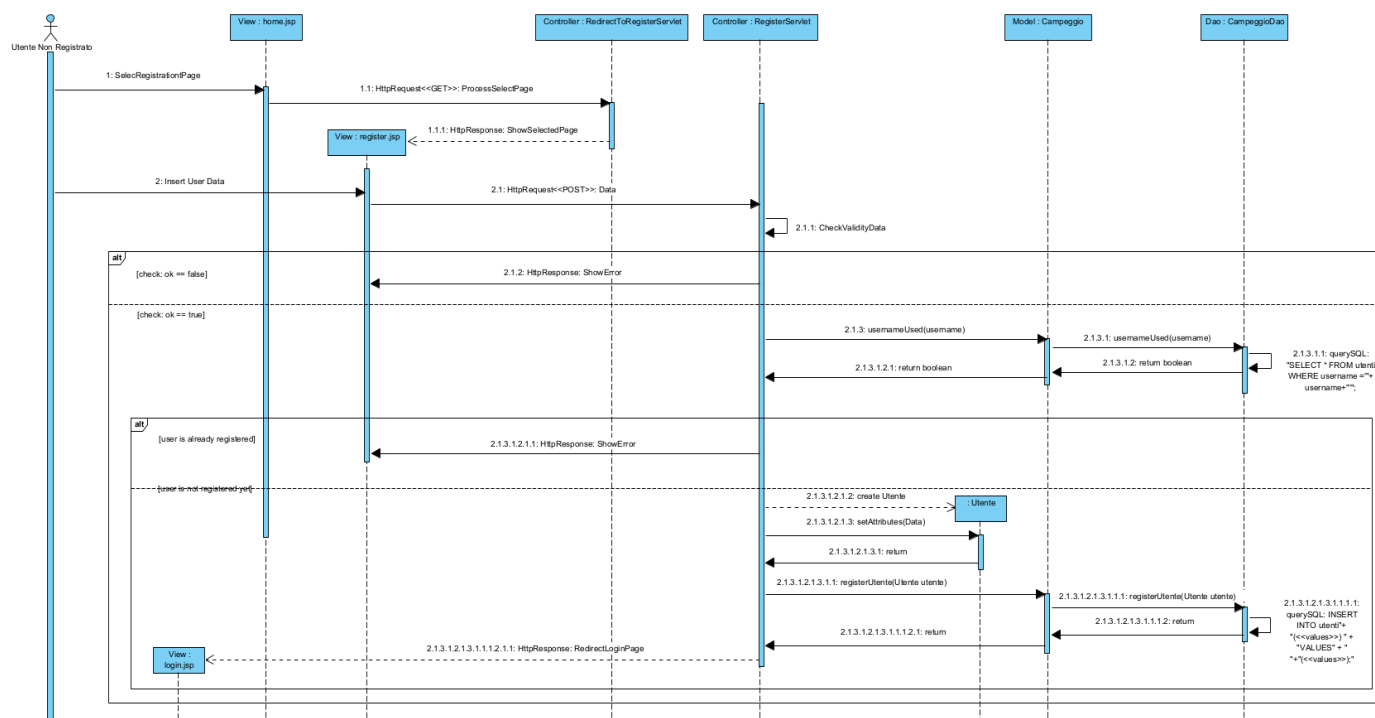
5.6 Sequence Diagram di Dettaglio

Per entrare nel profondo, possiamo analizzare diversi scenari attraverso dei Sequence Diagram di dettaglio, dei Behavioral Diagram che mi consentono di esplicitare tutti i comportamenti del sistema e le azioni dell'Utente.

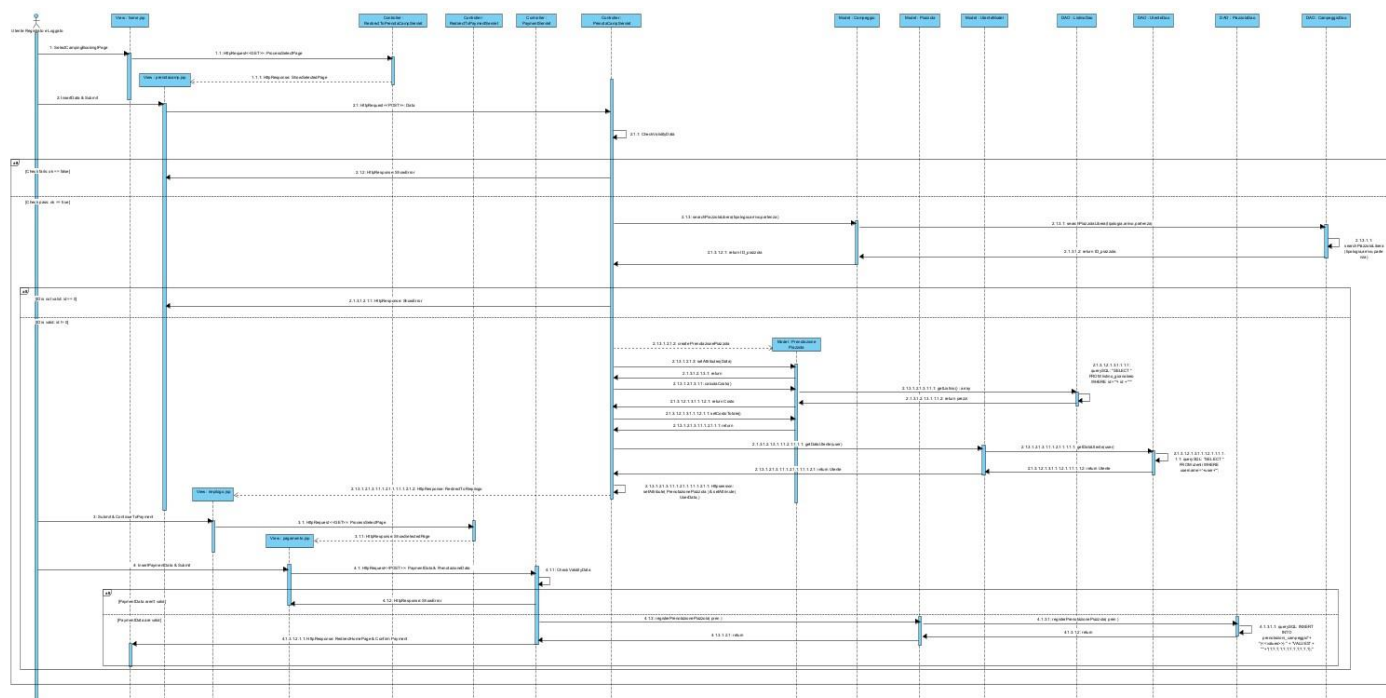
- **Login di un utente** (Analogo è il caso per il login dell'amministratore ma è necessario inviare alla sessione anche i dati relativi alle prenotazioni ed ai monitoraggi dei costi)



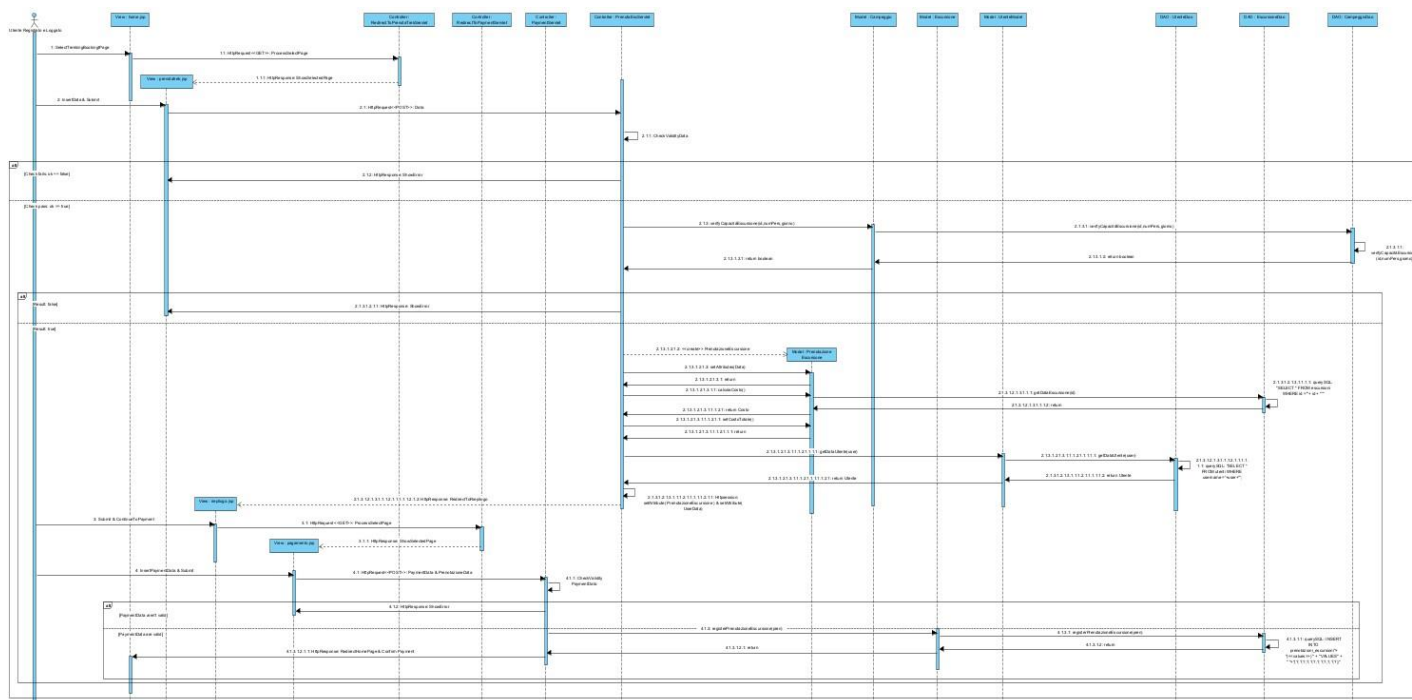
- **Registrazione di un utente**



- **Prenotazione Piazzola**

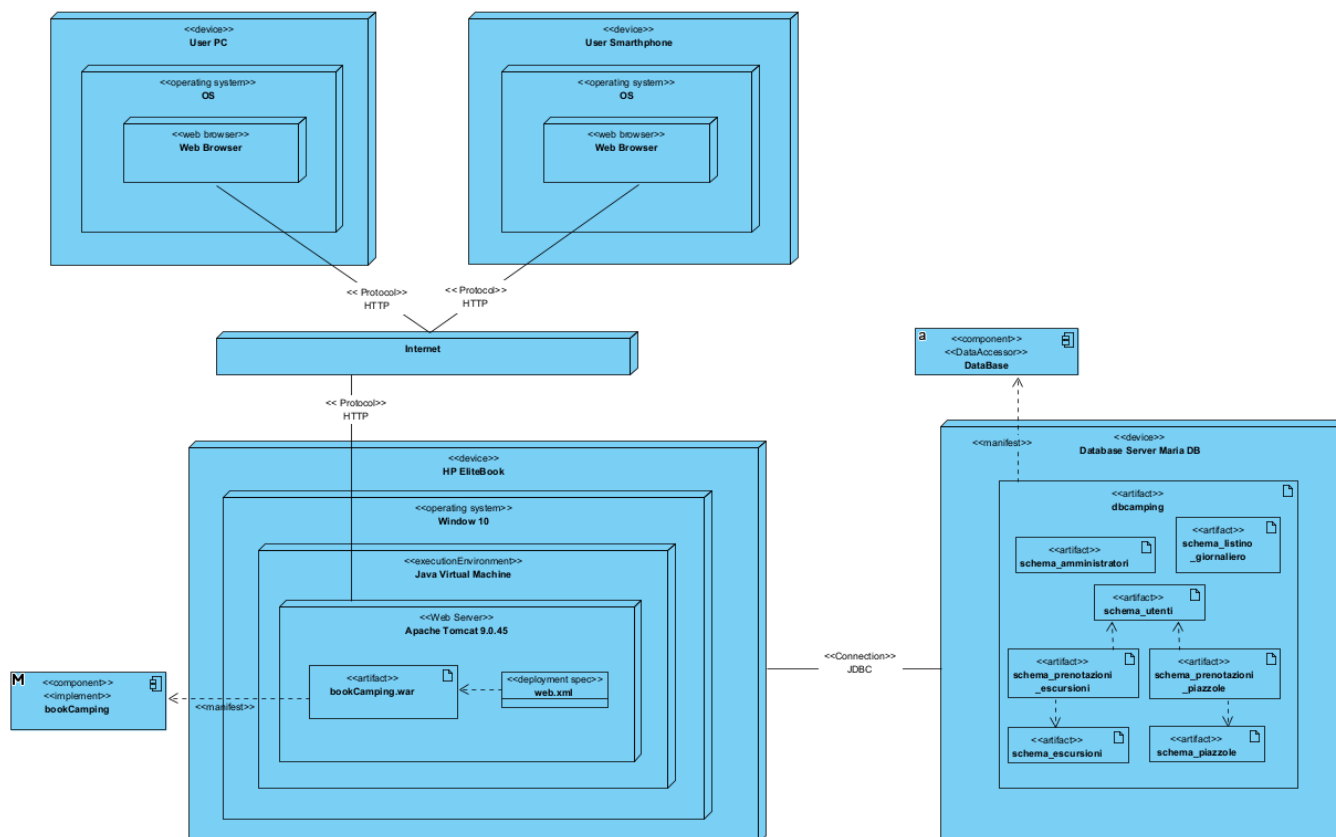


- **Prenotazione Escursione di *Trekking***

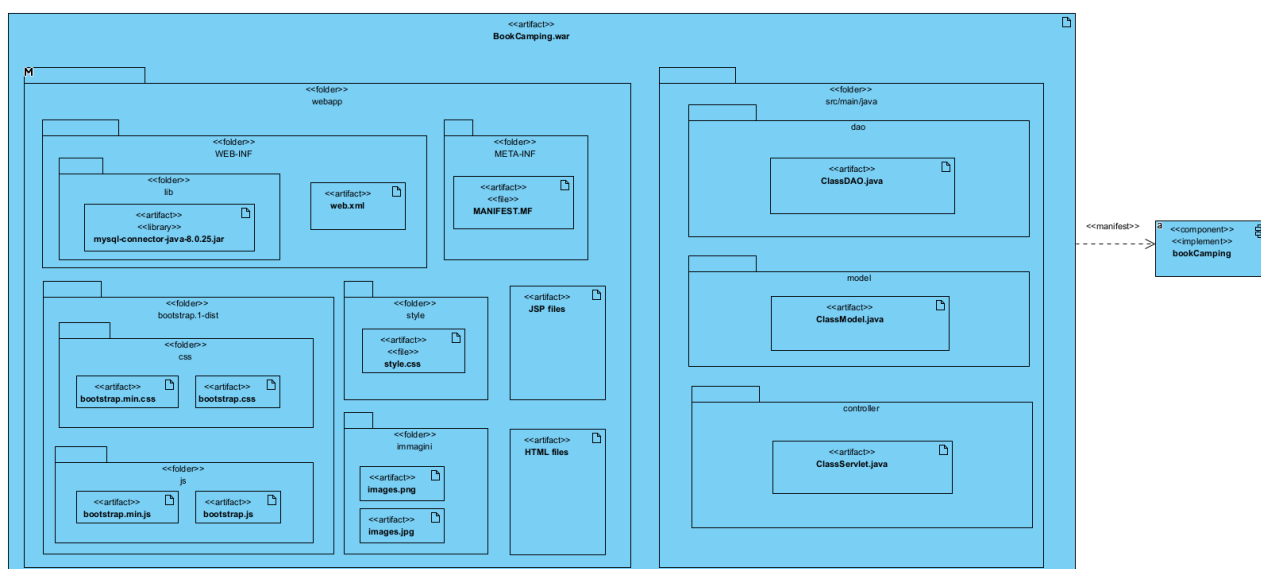


5.7 Deployment Diagram

Il Deployment Diagram, diagramma di tipo statico, mi consente di descrivere il sistema bookCamping in termini di risorse hardware, dette *nod*i, e di relazioni fra di esse.



5.8 Manifestation of Component by Artifact



Capitolo 6: Strumentazione ed implementazione

Per la realizzazione della *Web Application bookCamping* è stato necessario installare e configurare dei sofisticati *tool*: un ambiente di sviluppo in cui poter sviluppare l'applicazione, un database in cui poter immagazzinare le informazioni utili al sistema implementato, un server locale su cui poter far eseguire e testare l'applicazione durante la sua produzione.

6.1 Ambiente di Sviluppo: *Eclipse for Java and Web Developers*

Per l'implementazione del sistema è stato utilizzato un ambiente di sviluppo integrato multi-linguaggio e multiplatforma: *Eclipse for Java and Web Developers*. Quest'ultima è una versione specializzata di Eclipse che presenta dei package aggiuntivi, strumenti per sviluppatori che lavorano con applicazioni Web e Java, un IDE Java e strumenti per *JavaScript*, *TypeScript*, *JavaServer Pages (JSP)* and *Servlet*, *Faces*, *Yaml*, *Markdown*, *Web Services*, ecc... Tra le tante funzionalità di questo ambiente di sviluppo, ho sfruttato a pieno la logica “*JSP – Servlet*” la quale, unita alla connessione ad un database ed alla presenza di un server, ha permesso la realizzazione dell'Applicazione Web.

Durante le iterazioni lavorative Eclipse ha dimostrato di essere un software professionale, al contempo semplice da utilizzare ma soprattutto versatile.

6.1.1 Installazione

L'installazione dell'ambiente di sviluppo *Eclipse for Java and Web Developers* è stata possibile tramite il web browser, in particolare collegandosi al sito ufficiale di Eclipse Foundation: “www.eclipse.org”.



6.1.2 Configurazione

Per scegliere la tipologia del progetto ho selezionato la voce “Progetto Web Dinamico”, in particolare il percorso è il seguente: Click File > New > Other > ***Dynamic Web Project***. Per poter lavorare con Eclipse non è stata necessaria una preconfigurazione.

6.2 Database XAMPP

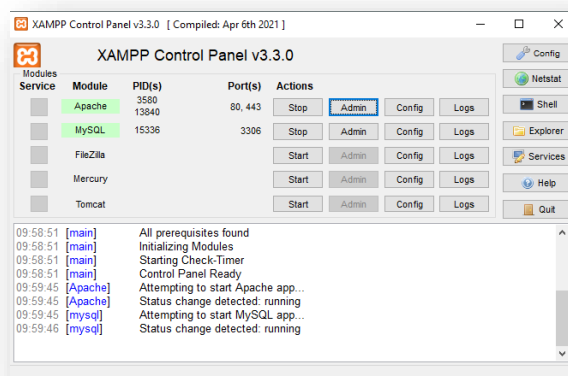
XAMPP è un software libero e *X-platform* o *cross-platform*, che sta per multiplatforma. Costituisce difatti un insieme integrato di pacchetti software che include *Apache*, *PHP*, *MariaDB* (*MySQL*), *phpMyAdmin* ed è orientato allo sviluppo/test di applicazioni web. XAMPP presenta un'interfaccia (*Control Panel*) molto semplice da utilizzare che consente di avviare/stoppare i servizi e configurare porte Web Server. Per l'utilizzo di questo tool è stata utilizzata una configurazione predefinita. Nel progetto ho utilizzato solo i servizi Apache e MySQL in quanto gli unici utili per i miei scopi.



6.2.1 Installazione

L'installazione di XAMPP è stata possibile tramite il web browser, in particolare collegandosi al sito ufficiale di Apache: “www.apachefriends.org”.

Attenzione! Affinché il progetto funzioni, è necessario scaricare una libreria da integrare nel sistema chiamata “*mysql-connector-java-8.0.25.jar*” che consente l'integrazione del DB nell'*Execution-environment*.



6.2.2 Configurazione

Ho utilizzato la configurazione *default* di XAMPP. Quest'ultimo assegna al web server nelle impostazioni standard la porta principale 80 e quella SSL 443. Risulta possibile verificare queste informazioni sulla sezione *Port(s)* del Control Panel. Sulla sezione *Actions* dello stesso è possibile inoltre accedere, tramite il pulsante Admin in corrispondenza di *MySQL*, alla pagina *phpMyAdmin*, una interfaccia Web che consente di gestire i database dei diversi progetti.

amministratori	★	Mostra	Struttura	Cerca	Inserisci	Svuota	Elimina	2	InnoDB	utf8mb4_general_ci	16.0 KiB	-
escursioni	★	Mostra	Struttura	Cerca	Inserisci	Svuota	Elimina	2	InnoDB	utf8mb4_general_ci	16.0 KiB	-
piazzole	★	Mostra	Struttura	Cerca	Inserisci	Svuota	Elimina	10	InnoDB	utf8mb4_general_ci	16.0 KiB	-
prenotazioni_escursioni	★	Mostra	Struttura	Cerca	Inserisci	Svuota	Elimina	1	InnoDB	utf8mb4_general_ci	48.0 KiB	-
prenotazioni_piazzole	★	Mostra	Struttura	Cerca	Inserisci	Svuota	Elimina	1	InnoDB	utf8mb4_general_ci	48.0 KiB	-
tariffa_giornaliera	★	Mostra	Struttura	Cerca	Inserisci	Svuota	Elimina	1	InnoDB	utf8mb4_general_ci	16.0 KiB	-
utenti	★	Mostra	Struttura	Cerca	Inserisci	Svuota	Elimina	10	InnoDB	utf8mb4_general_ci	16.0 KiB	-
7 tabelle		Totali						27	InnoDB	utf8mb4_general_ci	176.0 KiB	0 B

6.3 Server Tomcat

Il server *Tomcat* è un server web *open-source* prodotto dalla Apache e fornisce una piattaforma software per l'esecuzione di applicazioni web sviluppate in linguaggio Java, implementando anche specifiche per tecnologie di sviluppo improntate su *JavaServer Pages* e *Servlet*. La versione di Tomcat che viene utilizzata per la realizzazione della Web Application è la 9.0.45.



6.3.1 Installazione

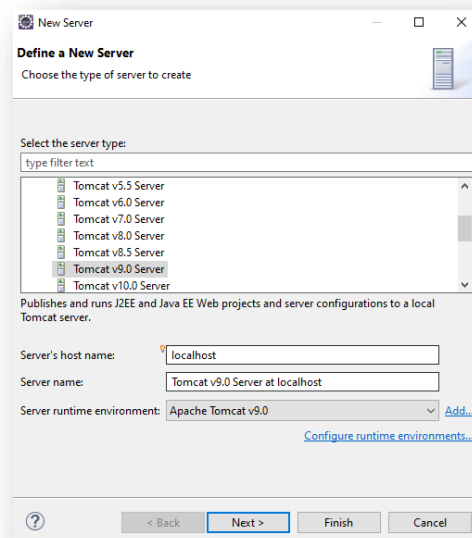
Come requisito, non solo per il server, risulta necessario scaricare tramite prompt dei comandi o tramite altri metodi la *Java Development Kit (JDK)*. Successivamente, come nell'installazione dei tool precedenti, per l'installazione del Server Tomcat basta accedere al sito ufficiale di Apache Tomcat e scaricare la versione 9.0.45.

Attenzione! Affinché il progetto funzioni, è necessario scaricare una libreria da inserire in *Java Build Path > Classpath* che si chiama “*Apache Tomcat v9.0*”.

6.3.2 Configurazione

La configurazione del server, una volta installato, viene effettuata su Eclipse, in particolare dalle opzioni del Dynamic Web Project è possibile creare un nuovo server scegliendo tra i differenti tipi e le differenti versioni sia il nome del server installato (nel nostro caso abbiamo selezionato Tomcat 9.0.45) sia la cartella sorgente in cui risiede il server.

Infine, nel caso il server non riesca ad inicializzarsi ed a stabilire una connessione è necessario o cambiare numero della porta richiesta dal Server sul *localhost* oppure terminare il processo che utilizza la porta richiesta dal Server.



Port Name	Port Number
Tomcat admin port	8006
HTTP/1.1	8080

6.4 Implementazione

Per l'implementazione del codice sono state necessarie competenze e conoscenze che ho acquisito attraverso tutorial, forum, libri in formato elettronico ed informazioni presenti abbondantemente in rete.

Il codice implementato in Java risiede essenzialmente nei package Controller, Model e Dao. La **View** è stata implementata maggiormente in HTML ed in CSS seguendo il tipico schema implementativo di un file HTML caratterizzato da *head* e *body*. La View risiede interamente nella folder *webapp* la quale contiene un insieme di file JSP ed HTML, una cartella *style* con dei file CSS per migliorare lo stile delle web pages, una libreria per l'utilizzo del framework *Bootstrap* ed una cartella *immagini*.

```
<html>

  <head>

  <body>

</html>
```

Tuttavia, trattandosi di un insieme di *file JSP* (escluso *l'index.html*), è stato possibile, attraverso degli *script*, utilizzare il linguaggio Java. Quest'ultimo non è stato utilizzato per implementare la logica applicativa, contenuta interamente nel Controller, bensì è stato utile per poter accedere ai dati provenienti dal Model, attraverso le primitive “*session.getAttribute(“parameter_name”)*” o in taluni casi in cui era necessario l'utilizzo di costrutti condizionali ed iterativi, ad esempio, per poter mostrare il giusto messaggio di errore, in seguito alla ricezione di un input non valido da parte dell'utente, come di seguito viene mostrato:

```
<%
String alert = (String) session.getAttribute("alert");

if(alert == "1"){ %>
  <div class="alert alert-danger" role="alert" style="top: 60px">
    Errore! Campi obbligatori non inseriti!
    <button type="button" class="btn-close" data-bs-dismiss="alert" aria-label="Close"></button>
  </div>
<% }else if(alert == "2"){ %>
  <div class="alert alert-danger" role="alert" style="top: 60px">
    Errore! Username o Password non esistenti!
    <button type="button" class="btn-close" data-bs-dismiss="alert" aria-label="Close"></button>
  </div>
<% }else if(alert == "11"){ %>
  <div class="alert alert-success" role="alert" style="top: 60px" >
    Registrazione effettuata con successo, ora effettua il login!
    <button type="button" class="btn-close" data-bs-dismiss="alert" aria-label="Close"></button>
  </div>
<%
}
%>
```

Il **Controller** è costituito esclusivamente da speciali classi Java chiamate *Servlet* che operano all'interno di un web server e che rappresentano estensioni di classi di tipo *HttpServlet*. Come è possibile notare visionando il codice, le Servlet presentano esclusivamente due funzioni: *doGet(HttpServletRequest request, HttpServletResponse response)* e *doPost(HttpServletRequest request, HttpServletResponse response)*. La *doGet* viene utilizzata, per lo più, quando è necessario visualizzare delle pagine Web, mentre la *doPost* viene utilizzata in seguito alla compilazione di form e quindi quando bisogna trasferire o immagazzinare dei dati sensibili.

```
package controller;

import java.io.IOException;

@WebServlet("/login")
public class LoginServlet extends HttpServlet {

    private static final long serialVersionUID = 1L;

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {}

    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {}

}
```

Ci sono infatti alcune Servlet che hanno solo il compito di reindirizzare l'utente da una pagina all'altra (le quali vengono chiamate implicitamente tramite il metodo di tipo *doGet*), mentre ci sono altre Servlet che non solo hanno solo il compito di reindirizzare l'utente da una pagina all'altra, ma devono anche implementare la logica applicativa servendosi anche di funzioni relative a classi del Model.

Diamo un'occhiata alle principali Servlet implementate.

- La **RegisterServlet** consente di verificare la correttezza dei dati inseriti dall'utente in fase di registrazione. L'implementazione utilizza una semplice variabile **alert** che viene assegnata ad un numero diverso in base all'errore che viene effettuato. I possibili errori sono dati da: campi obbligatori non inseriti, le due password di verifica che non combaciano, lo username che è già utilizzato da un altro utente, la password che non ha almeno nove caratteri, la password che non ha soli caratteri o cifre, la password che non ha almeno due cifre, l'e-mail che non è valida (non è nel formato ad es. *bookcamping@example.com*), il numero di cellulare che non è valido (non è nel formato ad es. *+39 34567890123*), la data di nascita successiva alla data odierna e l'età che non supera i 18 anni.
- Se i dati sono validi, la RegisterServlet prosegue con la creazione di un nuovo Utente e con

la sua registrazione definitiva. Essa richiama un metodo di una classe del Model, la quale a sua volta assegna il compito ad una classe Dao per l'inserimento in Database. La RegisterServlet infine reindirizza l'utente alla *Home*.

Se i dati non sono validi, la RegisterServlet reindirizza l'utente alla stessa pagina di registrazione.

- La **PrenotaCampServlet** consente di verificare la correttezza dei dati inseriti dall'utente in fase di prenotazione di una piazzola. I possibili errori sono dati da: campi obbligatori non inseriti, la data di arrivo che deve essere successiva alla data odierna, la partenza che deve essere successiva all'arrivo (oggi < arrivo < partenza) e la mancanza di piazzole libere. Questa viene verificata richiama un metodo, *searchPiazzolaLibera(tipologia, arrivo, partenza)*, di una classe del Model, il quale a sua volta, attraverso una funzione del package Dao, effettua un check su tutte le prenotazioni associate alla stessa tipologia di piazzola scelta nei giorni selezionati. Inoltre, per poter prenotare è necessario includere almeno una notte.

Se i dati sono validi, la PrenotaCampServlet reindirizza l'utente sulla View di riepilogo per avere un'ulteriore conferma dell'ordine altrimenti reindirizza l'utente alla stessa pagina di prenotazione.

- La **PrenotaEscServlet** consente di verificare la correttezza dei dati inseriti dall'utente in fase di prenotazione di un'escursione. I possibili errori sono dati da: campi obbligatori non inseriti, la data dell'escursione che deve essere successiva alla data odierna e il limite massimo di persone raggiunto. Quest'ultima condizione viene verificata dalla PrenotaEscServlet richiama un metodo, *verifyCapacitàEscursione(id, numpersone, giorno)*, di una classe del Model, il quale a sua volta, attraverso una funzione del package Dao, effettua un check su tutte le prenotazioni con stessa tipologia di escursione scelta nel giorno selezionato.

Se i dati sono validi, la PrenotaEscServlet reindirizza l'utente sulla View di riepilogo per avere un'ulteriore conferma dell'ordine altrimenti reindirizza l'utente alla stessa pagina di prenotazione.

- La **PaymentServlet** è un'entità di tipo **Stub** e consente di verificare la correttezza dei dati inseriti dall'utente in fase di pagamento. I possibili errori sono dati da: campi obbligatori non inseriti, CVV e Card Number che non hanno un formato valido o che non sono costituiti da sole cifre, carta di credito scaduta e circuiti di credito non riconosciuti. A tal proposito, vengono accettate le carte di credito VISA che iniziano con il numero 4, MasterCard che iniziano con il numero 5 oppure il 2, Diners Club che iniziano con il numero 3, American

Express che iniziano con il numero 3 e Discovery Card che iniziano con il numero 6.

Se i dati sono validi, la `PrenotaEscServlet` registra la prenotazione sul DB attraverso il metodo di una classe del Model e reindirizza l'utente sulla View di Home con un messaggio di avvenuto pagamento altrimenti reindirizza l'utente alla stessa pagina di pagamento.

Semplificazione: supponiamo che l'utente abbia sempre disponibilità economica.

- La ***LoginServlet*** è la Servlet necessaria a validare username e password inseriti dall'utente nel form della *login page*. Richiama il metodo `isLogin(username, password)` associato ad una classe del Model che consente di verificare se l'utente si è registrato o meno. Esiste una funzione analoga per l'amministratore. Nel caso il login venga effettuato dall'amministratore e le credenziali siano valide, la `LoginServlet` passa alla *Home_Administrator* tutti i dati associati alle prenotazioni effettuate ed agli incassi individuati richiesti al Model attraverso funzioni del tipo `monitoraIncassiTotali()`, `monitoraIncassiCamp()`, `monitoraIncassiEsc()`. Se l'autenticazione è corretta, la `LoginServlet` invalida la sessione precedente, se esiste, e ne crea una nuova reindirizzando l'utente alla *Home/ Home_Administrator*. Altrimenti, reindirizza l'utente alla stessa pagina di login definendo l'errore che è stato commesso.
- La ***LogoutServlet*** invalida la sessione precedente e reindirizza l'utente alla pagina di *login*.
- Le altre *Servlet* si limitano a reindirizzare l'utente da una View all'altra, a prelevare dati dal Model ed a passare quest'ultimi alle diverse pagine Web che costituiscono la View.

Il **Model** rappresenta il “modello di dominio” ed ha il compito descrivere tutte le entità in gioco definendo anche quelle che sono le relazioni che intercorrono tra di esse. Le entità implementate nel Model sono le seguenti:

- `Amministratore.java`
- `Utente.java`
- `Campeggio.java`
- `Escursione.java`
- `Piazzola.java`
- `PrenotazionePiazzola.java`
- `PrenotazioneEscursione.java`

Le entità vengono definite da classi scritte interamente in Java alle quali corrispondono degli attributi che definiscono le qualità associate agli oggetti istanziati di quella determinata classe. A ciascuna entità sono assegnate delle responsabilità, ovvero dei metodi che sono utilizzati dai moduli del Controller per poter attuare la logica applicativa associata al sistema che si vuole implementare. Il Model ha un ruolo molto importante: offre l'utilizzo di rilevanti funzioni al Controller, consente

alla View l'accesso ai dati tramite il Controller ed è l'unico package ad interfacciarsi con le classi Dao.

Il **Data Access Object**, unito all'utilizzo di un database, mi ha consentito di rendere i dati *persistenti*. Dao è il package in cui sono presenti tutte le entità che definiscono funzioni che hanno accesso diretto al Database. Infatti, ognuno di questi metodi verifica la presenza di un driver indispensabile per il proprio funzionamento e garantisce una connessione al server del Database, attraverso la primitiva *DriverManager.getConnection()*.

In tal è stato possibile effettuare delle *query* ed ottenere dei risultati contenuti nel *ResultSet*.

Presento un esempio dell'implementazione del metodo relativo alla classe CampeggioDao in fase di login per verificare la correttezza di username e password inseriti dall'utente.

```
public static boolean islogin(String username,String password) throws ClassNotFoundException, SQLException {
    try {
        Class.forName("com.mysql.cj.jdbc.Driver"); //libreria che permette la connessione
    } catch (ClassNotFoundException e) {
        System.out.println("ClassNotFoundException: ");
        System.err.println(e.getMessage());
    }

    String SQLquery="SELECT * FROM utenti WHERE username ='"+
        username+"' AND password ='"+
        password+
        "'";

    try (Connection connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/dbcamping","root","");
        PreparedStatement preparedStatement = connection.prepareStatement(SQLquery));) {

        ResultSet resultSet= preparedStatement.executeQuery();
        while(resultSet.next()) {
            if ( username.equals(resultSet.getString("username")) &&
                password.equals(resultSet.getString("password"))) {
                return true;
            }
        }

    } catch (SQLException e) {
        e.printStackTrace();
    }

    return false;
}
```

Le operazioni che effettuo tramite *query* sono maggiormente comandi di tipo *SELECT*, per prelevare dei dati dal database, e di tipo *INSERT*, per aggiungere dei dati al database.

Per quanto riguarda il database vero e proprio, è stato creato in MySQL collegandomi alla pagina phpMyAdmin messa a disposizione da XAMPP su <http://localhost/phpmyadmin/>.

La prima operazione che ho eseguito è stata creare il database:

```
CREATE DATABASE dbcamping;
```

Le successive operazioni che ho eseguito sono state creare le varie tabelle. Le tabelle create sono sette: *utenti*, *amministratori*, *escursioni*, *piazzole*, *prenotazioni_piazzole*, *prenotazioni_escursioni* e *listino_giornaliero*.

Di seguito riporto un esempio della creazione della tabella *utenti*:

```
CREATE TABLE `utenti` (
  `id` int(45) UNSIGNED NOT NULL AUTO_INCREMENT,
  `username` varchar(45) NOT NULL,
  `password` varchar(255) NOT NULL,
  `nome` varchar(255) NOT NULL,
  `cognome` varchar(255) NOT NULL,
  `nascita` DATE NOT NULL,
  `indirizzo` varchar(255) NOT NULL,
  `città` varchar(255) NOT NULL,
  `paese` varchar(255) NOT NULL,
  `cap` varchar(45) NOT NULL,
  `cellulare` varchar(255) NOT NULL,
  `email` varchar(255) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `username` (`username`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci
AUTO_INCREMENT= 1;
```

Oltre alla creazione degli schemi associati al Database, talvolta, è stato necessario aggiungere dei **vincoli** sulle *foreign key* che consentono di mettere in relazione due o più tabelle tra di loro.

Ad esempio, ogni prenotazione di una piazzola presenta un attributo *id_utente* ed un *id_piazzola*, per cui necessita di due vincoli:

id	arrivo	partenza	numAdulti	numBambini	corrente	postoAuto	postoMoto	note	costoTotale	id_utente	id_piazzola
151	2021-09-27	2021-09-30	2	1	no	no	no	/	162	1	4
152	2021-09-27	2021-09-30	2	0	no	si	si	/	156	1	5

Nell'esempio in figura, il primo vincolo lega le tabelle *piazzole-prenotazioni piazzole*, il secondo vincolo lega tra loro *utenti-prenotazioni piazzole*:

ON DELETE	RESTRICT	ON UPDATE	RESTRICT	id_piazzola	dbcamping	piazzole	id
+ Aggiungi campo							
ON DELETE	RESTRICT	ON UPDATE	RESTRICT	id_utente	dbcamping	utenti	id
+ Aggiungi campo							

6.4.1 Linguaggi di programmazione

Attraverso una stima approssimativa eseguita da *GitHub*, possiamo affermare che oltre il 91% del codice è scritto in **Java**, circa l'8,2% del codice è scritto in **HTML** mentre la restante minima parte è scritta in **CSS** (sigla di *Cascading Style Sheets*) ed in linguaggio **SQL** per le query effettuate al DB.

6.4.2 Design Framework: Bootstrap

Bootstrap è un *framework* per applicazioni web multiplatforma che consente di arricchire le *User Interface* di componenti come pulsanti, moduli, barre di navigazione e molto altro.

Alcuni degli elementi di cui ho usufruito, successivamente personalizzati, sono:

- **Carousel**



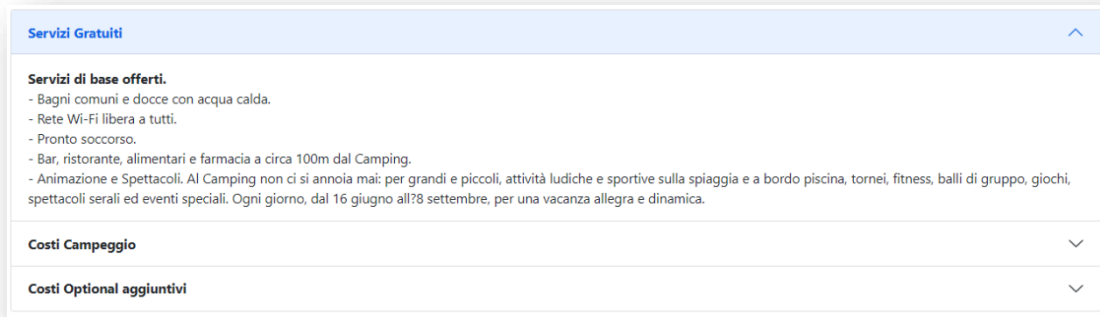
- **Cards**



- **Navbar**



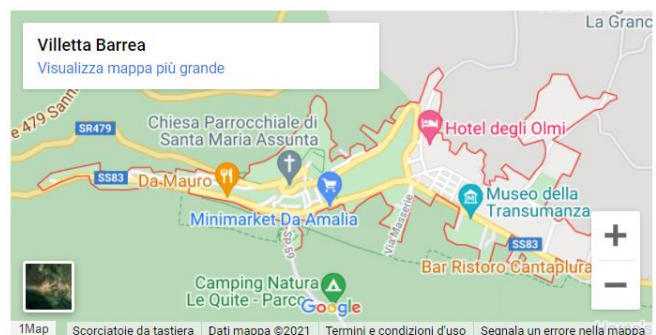
- **Accordion**



Per poter usufruire di Bootstrap, è stato necessario scaricare delle librerie nelle quali sono contenuti file in *CSS* ed in *JS*.

Oltre al *framework* Bootstrap, ho creato una prima cartella *style* in cui ho modificato dei parametri grafici delle interfacce ed una seconda cartella *immagini* nella quale ho inserito tutte le immagini utilizzate per migliorare l'estetica delle interfacce.

Un altro elemento integrato dall'esterno (dal sito <https://1map.com/it/map-embed>) è la mappa del paese in cui è collocato il campeggio e consente di accedere direttamente a *Google Maps*.



6.4.3 Esecuzione del Software prodotto

Per eseguire il software, una volta configurate tutte le componenti su Eclipse è necessario cliccare su **Run > Run on server**. In tal modo, verrà aperto automaticamente un Web Browser che è possibile scegliere nelle opzioni del progetto. L'indirizzo a cui verrà creato un collegamento diretto sarà `"http://localhost:8080/BookCamping/"`. Il file visualizzato di default sul browser sarà `l'index.html`.

Muoversi da una pagina Web all'altra risulta molto semplice ed intuitivo.

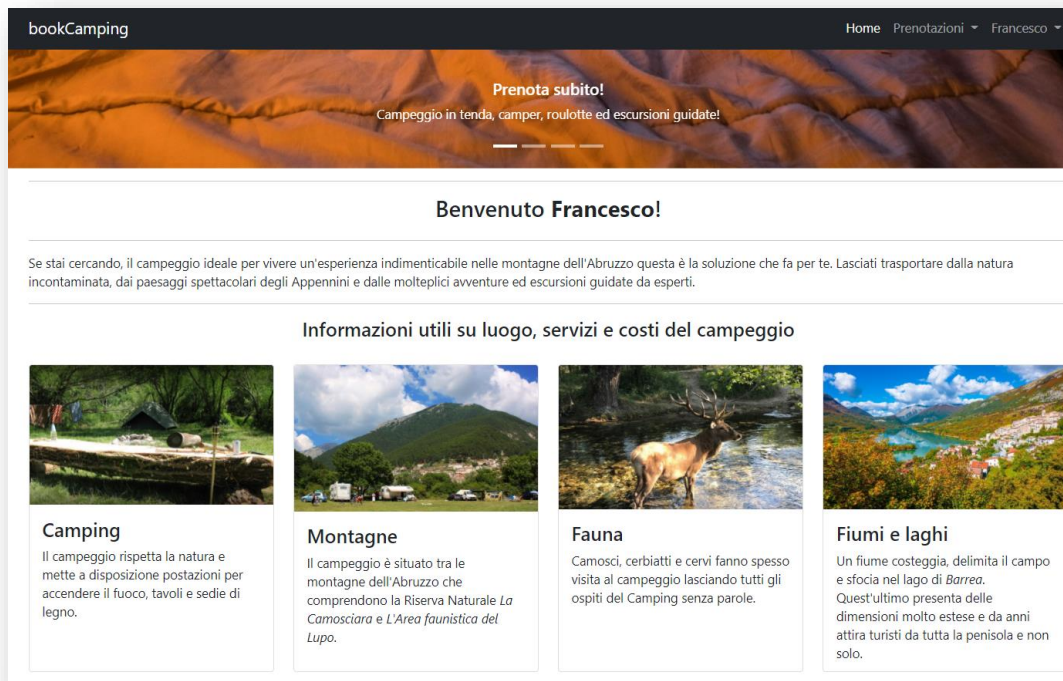
A scopo dimostrativo presento alcune delle pagine web e dei form necessari per l'implementazione della nostra *Web Application*:

- **Login:** da questa sezione è possibile accedere sia come utente ordinario sia come amministratore. In ognuno dei due casi è necessario inserire le proprie credenziali per accedere.

- **Registrazione:** se non si è in possesso di credenziali è necessario registrarsi attraverso un modulo da compilare seguendo il percorso sulla Navbar:

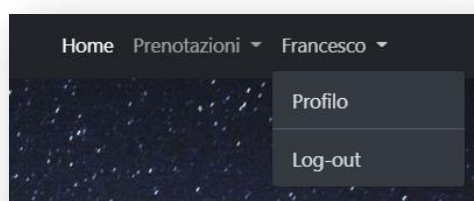
Home > Account > Sign-up

- **Home (in seguito al login):** esiste una home per gli utenti non registrati che possono solo consultare servizi ed escursioni ed una home, quella di seguito mostrata, per utenti loggati dalla quale è possibile accedere alla sezione delle prenotazioni, del profilo e dei servizi offerti.



- **Profilo:** è possibile accedervi seguendo il percorso sulla Navbar:

Home > Nome Utente > Profilo

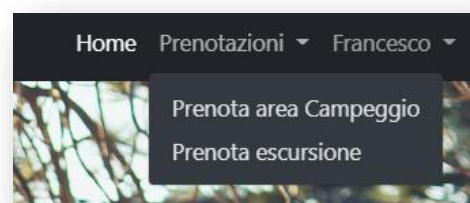


- **Administrator Home:** tutte le funzionalità di monitoraggio degli incassi e delle prenotazioni sono ben visibili sulla home, una volta effettuato il *login*.

The screenshot shows the Administrator Home page of the bookCamping system. The page is titled "Benvenuto Amministratore!" and includes a navigation bar with "Home" and "admin123". The main content area is divided into several sections:

- Monitoraggio incassi:** A section showing the total income from campsite reservations (513 euro) and excursion reservations (290 euro), with a total income of 803 euro.
- Raccolta prenotazioni:** A section containing two tables:
 - Prenotazioni piazzole:** A table with columns for ID_prenotazione, Data_Arrivo, Data_Partenza, n°adulti, n°bambini, Corrente, Posto_Auto, Posto_Moto, Note, Costo_Totale, Id_utente, and Id_piazzola. It lists three reservations with IDs 151, 152, and 153.
 - Prenotazioni escursioni:** A table with columns for ID_escursione, Data_escursione, Numero_Persone, Costo_Totale, Id_escursione, and Id_utente. It lists two excursion reservations with IDs 39 and 40.

- **Richiesta di prenotazione di un'escursione:** seguendo il percorso sulla Navbar: *Home > Prenotazioni > Prenota escursione*



The screenshot shows the "Richiesta di prenotazione Escursione a Cavallo" form. The form is titled "Richiesta di prenotazione Escursione a Cavallo" and includes a note: "* i campi contrassegnati dall'asterisco sono obbligatori".

Escursioni a cavallo

L'escursioni a cavallo si tengono ogni giorno: partono alle **15:00** di pomeriggio e terminano alle **17:00**. L'escursione ti consentirà di visitare tutto ciò che c'è di bello nella natura incontaminata delle montagne dell'Abruzzo. È una giusta opzione anche per i bambini che vorranno intraprendere la stessa esperienza su degli adorabili pony. Guida: *Pasquale Girardi*
Partenza: *Riserva Naturale "La Camosciara"*
Durata: **2h**
Prezzo: **15 euro a persona**

Dati prenotazione

*Giorno:

*n° persone:

[Clicca qui per richiedere la disponibilità](#)

Ogni richiesta è subordinata alla conferma del Camping.

- **Richiesta di prenotazione di una piazzola del campeggio:** seguendo il percorso sulla Navbar:

Home > Prenotazioni > Prenota area Campeggio

bookCamping

Home Prenotazioni Francesco

Richiesta di prenotazione per un'area del campeggio

* i campi contrassegnati dall'asterisco sono obbligatori

Dati prenotazione

*Arrivo:

gg/mm/aaaa

*Partenza:

gg/mm/aaaa

*n° adulti:

1

*n° bambini(4-10):

0

Persona	Costo al giorno
Adulto	13 €
Bambino (4-10)	10 €
Bambino (0-3)	Gratuito

*Tipologia piazzola:

Tenda Small

Tipologia Piazzola	Costo al giorno
Per Tenda Small	11 €
Per Tenda Large	18 €
Per Roulotte	22 €
Per Camper	24 €

Corrente

Posto moto

Posto auto

Optional	Costo al giorno
Corrente	3 €
Posto Auto	5 €
Posto Moto	3 €

note:

es. Arriveremo con un po' di ritardo

Clicca qui per richiedere la disponibilità


Ecco cosa si intende per le diverse tipologie di piazzola:

Tenda Small

Tenda Large

Roulotte

Camper



Camper

Area: 72 m²

6.4.4 GitHub



Ho usufruito della piattaforma GitHub che offre un servizio di hosting per progetti software.

Infatti, caricando il codice sorgente dei programmi su GitHub è possibile accedere al progetto su GitHub collegandosi al profilo *francisfat* e selezionando la repository pubblica *bookCamping*.

Capitolo 7: test e prove eseguite

Black Box Test del 27/10/2021

I test effettuati per verificare il comportamento del sistema sono stati di tipo **Black-Box** e sono basati su requisiti e su scenari d'uso.

Caso d'uso	Pre-condizione	Input	Output	Post-condizione	Successo
Registrazione	L'utente non inserisce tutti i campi Obbligatori.	Nome: "" Cognome: "Grasso" Nascita: "05/08/1998" Paese: "italia" CAP: "82030" Città: "Campoli (BN)" Indirizzo: "Via Roma 25" Cellulare: "+393434567891" E-mail: "fra@gmail.com" Username: "Francesco" Password1: "password123" Password2: "password123"	Errore: Campi Obbligatori non inseriti	Registrazione non effettuata.	✓
	L'utente inserisce password non uguali.	Nome: "" Cognome: "Grasso" Nascita: "05/08/1998" Paese: "italia" CAP: "82030" Città: "Campoli (BN)" Indirizzo: "Via Roma 25" Cellulare: "+393434567891" E-mail: "fra@gmail.com" Username: "Francesco" Password1: "password12" Password2: "password12"	Errore: Le password non combaciano	Registrazione non effettuata.	✓
	L'utente inserisce una password con meno di nove caratteri.	Nome: "" Cognome: "Grasso" Nascita: "05/08/1998" Paese: "italia" CAP: "82030" Città: "Campoli (BN)" Indirizzo: "Via Roma 25" Cellulare: "+393434567891" E-mail: "fra@gmail.com" Username: "Francesco" Password1: "pass123" Password2: "pass123"	Errore: Password con meno di nove caratteri	Registrazione non effettuata.	✓
	L'utente inserisce una password con meno di due cifre.	Nome: "" Cognome: "Grasso" Nascita: "05/08/1998" Paese: "italia" CAP: "82030" Città: "Campoli (BN)" Indirizzo: "Via Roma 25" Cellulare: "+393434567891" E-mail: "fra@gmail.com" Username: "Francesco" Password1: "password1" Password2: "password1"	Errore: Password con meno di due cifre	Registrazione non effettuata.	✓

L'utente inserisce una password con simboli.	Nome: "Francesco" Cognome: "Grasso" Nascita: "05/08/1998" Paese: "italia" CAP: "82030" Città: "Campoli (BN)" Indirizzo: "Via Roma 25" Cellulare: "+393434567891" E-mail: "fra@gmail.com" Username: "Francesco" Password1: "password.12" Password2: "password.12"	Errore: Password con simboli	Registrazione non effettuata.	✓
L'utente "Francesco" è già registrato perché il suo Username è riconosciuto come già esistente sul Database. Lo Username è unico.	Nome: "Francesco" Cognome: "Grasso" Nascita: "05/08/1998" Paese: "italia" CAP: "82030" Città: "Campoli (BN)" Indirizzo: "Via Roma 25" Cellulare: "+393434567891" E-mail: "fra@gmail.com" Username: "Francesco" Password1: "password.12" Password2: "password.12"	Errore: Utente già esistente	Registrazione non effettuata.	✓
L'utente inserisce come data di nascita una data nel futuro.	Nome: "Francesco" Cognome: "Grasso" Nascita: "05/08/2054" Paese: "italia" CAP: "82030" Città: "Campoli (BN)" Indirizzo: "Via Roma 25" Cellulare: "+393434567891" E-mail: "fra@gmail.com" Username: "Francesco" Password1: "password12" Password2: "password12"	Errore: Data di nascita futura	Registrazione non effettuata.	✓
L'utente non risulta maggiorenne.	Nome: "Francesco" Cognome: "Grasso" Nascita: "05/08/2010" Paese: "italia" CAP: "82030" Città: "Campoli (BN)" Indirizzo: "Via Roma 25" Cellulare: "+393434567891" E-mail: "fra@gmail.com" Username: "Francesco" Password1: "password12" Password2: "password12"	Errore: Utente non maggiorenne	Registrazione non effettuata.	✓
L'utente inserisce un cellulare non nel formato corretto (ovvero ad es. +39 345 678 9011).	Nome: "Francesco" Cognome: "Grasso" Nascita: "05/08/1998" Paese: "italia" CAP: "82030" Città: "Campoli (BN)" Indirizzo: "Via Roma 25" Cellulare: "+39d434567891" E-mail: "fra@gmail.com" Username: "Francesco" Password1: "password12" Password2: "password12"	Errore: Cellulare non nel formato corretto	Registrazione non effettuata.	✓
L'utente non inserisce un'e-mail nel formato corretto (ovvero ad es. email@bookcamping.it).	Nome: "Francesco" Cognome: "Grasso" Nascita: "05/08/1998" Paese: "italia" CAP: "82030" Città: "Campoli (BN)" Indirizzo: "Via Roma 25" Cellulare: "+393434567891" E-mail: "fragmail.com"	Errore: E-mail non nel formato corretto	Registrazione non effettuata.	✓

		Username: "Francesco" Password1: "password12" Password2: "password12"			
	L'utente inserisce tutti i dati nel formato corretto e non era ancora registrato.	Nome: "Francesco" Cognome: "Grasso" Nascita: "05/08/1998" Paese: "italia" CAP: "82030" Città: "Campoli (BN)" Indirizzo: "Via Roma 25" Cellulare: "+393434567891" E-mail: "fra@gmail.com" Username: "Francesco" Password1: "password123" Password2: "password123"	Message: Registrazione effettuata con successo, ora effettua il login!	Registrazione effettuata	✓

Caso d'uso	Pre-condizione	Input	Output	Post-condizione	Successo
Login	L'utente non inserisce tutti i campi obbligatori.	Username: "Lorenzo" Password: ""	Errore: Campi Obbligatori non inseriti	Login non effettuato.	✓
	L'utente inserisce password e username non validi o perché sono sbagliati o perché l'utente non ha effettuato ancora la registrazione	Username: "Lorenzo" Password: "passfitti98"	Errore: Password o Username non esistenti	Login non effettuato.	✓
	L'utente inserisce password e username corretti ed è registrato.	Username: "Francesco" Password: "password12"	Login effettuato con successo.	L'utente può accedere e prenotare.	✓

Caso d'uso	Pre-condizione	Input	Output	Post-condizione	Successo
Prenotazione Piazzola	L'utente non inserisce tutti i campi obbligatori.	Check-in: "" Check-out: " 25/10/2021 " Tipologia: " Tenda Small " Num_Adulti: "2" Num_Bambini: "0" Corrente: "no" Posto Auto: "no" Posto Moto: "no" Note: "Faremo ritardo"	Errore: Campi Obbligatori non inseriti	Prenotazione non effettuata.	✓
	L'utente inserisce Check-in successivo al check-out.	Check-in: "27/10/2021 " Check-out: " 25/10/2021 " Tipologia: " Tenda Small " Num_Adulti: "2" Num_Bambini: "0" Corrente: "no" Posto Auto: "no" Posto Moto: "no" Note: "Faremo ritardo"	Errore: Check-in successivo al check-out	Prenotazione non effettuata.	✓
	L'utente inserisce Check-in pari al check-out, bisogna inserire un periodo temporale che comprende almeno una notte.	Check-in: "27/10/2021 " Check-out: " 27/10/2021 " Tipologia: " Tenda Small " Num_Adulti: "2" Num_Bambini: "0" Corrente: "no" Posto Auto: "no" Posto Moto: "no" Note: "Faremo ritardo"	Errore: Bisogna prenotare almeno per una notte	Prenotazione non effettuata.	✓
	L'utente inserisce il Check-in pari ad una data passata.	Check-in: "27/10/2020 " Check-out: " 25/10/2021 " Tipologia: " Tenda Small " Num_Adulti: "2" Num_Bambini: "0" Corrente: "no" Posto Auto: "no" Posto Moto: "no" Note: "Faremo ritardo"	Errore: Check-in in data passata	Prenotazione non effettuata.	✓

	L'utente inserisce una tipologia di piazzola non libera nel periodo indicato.	Check-in: "27/10/2021" Check-out: "30/10/2021" Tipologia: "Tenda Small" Num_Adulti: "2" Num_Bambini: "0" Corrente: "no" Posto Auto: "no" Posto Moto: "no" Note: "Faremo ritardo"	Errore: Tipologia di piazzola non libera nel periodo indicato	Prenotazione non effettuata.	✓
	L'utente inserisce tutti i campi obbligatori in modo corretto. L'utente inserisce una tipologia di piazzola libera nel periodo indicato	Check-in: "23/10/2021" Check-out: "25/10/2021" Tipologia: "Tenda Small" Num_Adulti: "2" Num_Bambini: "0" Corrente: "no" Posto Auto: "no" Posto Moto: "no" Note: "Faremo ritardo"	Message: Prenotazione registrata.	Prenotazione effettuata.	✓

Caso d'uso	Pre-condizione	Input	Output	Post-condizione	Successo
Prenotazione Escursione	L'utente non inserisce tutti i campi obbligatori.	Data: "" NumPersone: "1"	Errore: Campi Obbligatori non inseriti	Prenotazione non effettuata.	✓
	L'utente inserisce una data ormai passata.	Data: "07/07/2020" NumPersone: "1"	Errore: Data scelta passata	Prenotazione non effettuata.	✓
	L'escursione nel giorno selezionato non può ospitare il numero di persone scelto dall'utente.	Data: "07/10/2021" NumPersone: "1"	Errore: Limite di persone massimo raggiunto	Prenotazione non effettuata.	✓
	L'utente inserisce tutti i campi obbligatori correttamente. L'escursione nel giorno selezionato può ospitare il numero di persone scelto dall'utente.	Data: "08/10/2021" NumPersone: "1"	Message: Prenotazione registrata.	Prenotazione effettuata.	✓

Caso d'uso	Pre-condizione	Input	Output	Post-condizione	Successo
Pagamento	L'utente non inserisce tutti i campi obbligatori.	Card Number: "5233 3232 4323 5455" Card Holder: "Francesco Grasso" Expiration Date: "01/01/2022" CVV: ""	Errore: Campi Obbligatori non inseriti	Pagamento non effettuato.	✓
	L'utente non inserisce CVV o Card Number in formato corretto.	Card Number: "523 3232 4323 5455" Card Holder: "Francesco Grasso" Expiration Date: "01/01/2022" CVV: "878"	Errore: CVV o Card Number non in formato corretto	Pagamento non effettuato.	✓
	L'utente inserisce CVV o Card Number non costituiti da sole cifre.	Card Number: "5233 3232 4323 5455" Card Holder: "Francesco Grasso" Expiration Date: "01/01/2022" CVV: "8c8"	Errore: CVV o Card Number non costituiti da sole cifre	Pagamento non effettuato.	✓
	L'utente inserisce un circuito di credito non riconosciuto.	Card Number: "1723 3232 4323 5455" Card Holder: "Francesco Grasso" Expiration Date: "01/01/2022" CVV: "878"	Errore: Circuito di credito non riconosciuto	Pagamento non effettuato.	✓
	L'utente inserisce una carta di credito scaduta.	Card Number: "5233 3232 4323 5455" Card Holder: "Francesco Grasso" Expiration Date: "01/01/2021" CVV: "878"	Errore: Carta di credito scaduta	Pagamento non effettuato.	✓
	L'utente inserisce tutti i campi obbligatori in maniera corretta.	Card Number: "5233 3232 4323 5455" Card Holder: "Francesco Grasso" Expiration Date: "01/01/2022" CVV: "878"	Message: Pagamento effettuato con successo.	Pagamento effettuato.	✓

Glossario

- *Piazzola*: in un campeggio, area riservata a una tenda o a una roulotte.
- *Camping*: soggiorno all'aperto, a scopo ricreativo o turistico, in attendamenti, roulotte o simili / area dotata di attrezzature igieniche e servizi vari, ove si può soggiornare in tende o roulotte.
- *Escursione*: breve viaggio a scopo scientifico, turistico o igienico; part., gita in montagna.
- *Caso d'uso*: il caso d'uso in informatica è una tecnica usata nei processi di ingegneria del software per effettuare in maniera esaustiva e non ambigua, la raccolta dei requisiti al fine di produrre software di qualità.
- *Software*: in informatica, l'insieme delle procedure e delle istruzioni in un sistema di elaborazione dati; si identifica con un insieme di programmi (in contrapposizione a hardware)
- *Architettura software*: l'organizzazione di base di un sistema, espressa dalle sue componenti, dalle relazioni tra di loro e con l'ambiente, e i principi che ne guidano il progetto e l'evoluzione.
- *Modulo software*: generalmente un file o porzione di codice sorgente che contiene istruzioni scritte per essere riutilizzate più volte nello stesso programma o in più programmi diversi: la modularizzazione di un programma permette al programmatore di avere una visione completa del programma stesso.
- *Funzione*: porzione di codice che può essere invocata da qualsiasi punto di un programma
- *Interfaccia*: punto di incontro o collegamento, spesso standardizzato, tra sistemi diversi e/o le modalità per permettere la loro interazione sotto forma di scambio di informazioni; nel paradigma di programmazione orientata agli oggetti, gli elementi pubblici di una classe.
- *Workflow*: (tradotto letteralmente "flusso di lavoro") la creazione di modelli, la gestione informatica dell'insieme dei compiti e dei diversi attori coinvolti nella realizzazione di un processo lavorativo.

Conclusioni

Lo sviluppo di questa applicazione Web e la realizzazione della sua relativa documentazione mi hanno consentito di conoscere: nuove tecniche di sviluppo, abilità di progettazione reale di grandi sistemi e di programmazione in HTML, ma soprattutto ha permesso la mia crescita personale e professionale.

Ho riconosciuto quanto l'adozione di processi organizzativi in ambito lavorativo sia un aspetto rilevante del lavoro stesso. Il processo di sviluppo UP si è rivelato efficace e molto semplice da mettere in pratica.

Questo progetto è stata la mia prima esperienza di progettazione e di sviluppo di Applicazioni Web. Ho avvertito una grande passione nel realizzare quello che mi ero prefissato di creare in fase di ideazione e le tante difficoltà incontrate non hanno fatto altro che alimentare la soddisfazione al termine del progetto.