UNIVERSITÀ DEGLI STUDI DELL'AQUILA

Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica

Progetto d'esame per il corso di Programmazione per il web

ParKing

Candidati

Francesco Buscaino
265469

Matteo Capricci
265906

Matteo Colazilli
264964

Anno Accademico 2020-2021

Indice

Capitolo 1

Introduzione

La seguente documentazione è relativa al progetto d'esame per il corso di "Programmazione per il web" dell'anno accademico 2021/2022. Verranno trattati i temi di analisi, design e progettazione dell' applicazione web che prende il nome di **ParKing**. Il progetto è stato sviluppato da Buscaino Francesco, Capricci Matteo e Colazilli Matteo nell'arco dell' intero corso e portato all'esame nel secondo appello della sessione successiva. L'applicazione web è raggiungibile su `http://par-king.it:8081`

Capitolo 2

Proposta di progetto

Di seguito la proposta di progetto che è stata sottoposta all'approvazione del professore ed è stata giudicata valida:

Applicazione: ParKing

Descrizione: Piattaforma web che consente di mettere a disposizione posti auto e di prenotarli. I posti auto possono essere messi a disposizione da privati, società o enti pubblici, includendo diversi servizi. Gli utenti che scelgono di occupare un posto in un determinato parcheggio specificano la durata della sosta e pagano la rispettiva tariffa. Eventuali servizi offerti possono essere:

- Posto auto coperto
- Videosorveglianza
- Lavaggio auto
- Ricarica elettrica
- Altro

Un utente può lasciare una recensione del servizio offerto esprimendo una valutazione e/o scrivendo delle note. La ricerca del parcheggio può essere fatta inserendo la città nella quale si desidera trovare il posto. Tipologie di utenti e loro obiettivi:

Amministratore:

- Modera i commenti

Utenti:

- Gestore parcheggio:
  - Mette a disposizione uno o più parcheggi
  - Aggiunge/Rimuove i servizi che offre il parcheggio
  - Stabilisce le tariffe dei servizi e dei posti offerti
  - Gestisce le informazioni relative ai parcheggi posseduti
  - Risponde alle recensioni dei clienti
- Cliente non registrato

  - Può registrarsi
  - Può effettuare una ricerca per trovare posti disponibili
  - Può vedere il dettaglio di un parcheggio

- Cliente registrato
  - Può fare tutto ciò che fa il cliente non registrato
  - Può effettuare prenotazione
  - Può disdire prenotazione
  - Può lasciare recensioni
  - Può accedere alla propria area personale

Per l'inserimento sulla piattaforma di nuovi parcheggi o per cambi strutturali a parcheggi esistenti, l'utente gestore dovrà contattare con metodi esterni alla piattaforma l'amministrazione dell' applicazione ed accordarsi sull'operazione che desidera fare.

Capitolo 3

Analisi

Dopo che la proposta è stata accettata, si è passati all'analisi del dominio applicativo e della situazione da modellare.

3.1 Modello logico

Il metodo utilizzato per analizzare la realtà d'interesse è stato utilizzare lo strumento del Modello Concettuale o Logico, ovvero un diagramma nel quale si vanno a rappresentare i concetti significativi del dominio applicativo e si stabiliscono le relazioni che ci sono tra questi.

Nella modellazione logica sono state seguite le linee guida che raccomandano di usare nomi esistenti nel dominio ed usare le associazioni per avere maggiore comprensione della situazione. Il risultato della modellazione è quello rappresentato nella figura 3.1.

Come si può osservare sono stati creati concetti per modellare qualsiasi tipo di dato che, nella realtà d'interesse, rappresenta un concetto che non può essere pensato come un testo od un numero semplici, bensì ha bisogno di un' interpretazione propria. Un caso che portiamo in evidenza è quello della tariffa: rappresenta il costo orario che si deve pagare per sostare in un parcheggio. Questa non può essere espressa come un numero, bensì va interpretata utilizzando un'unità di misura. Un altro

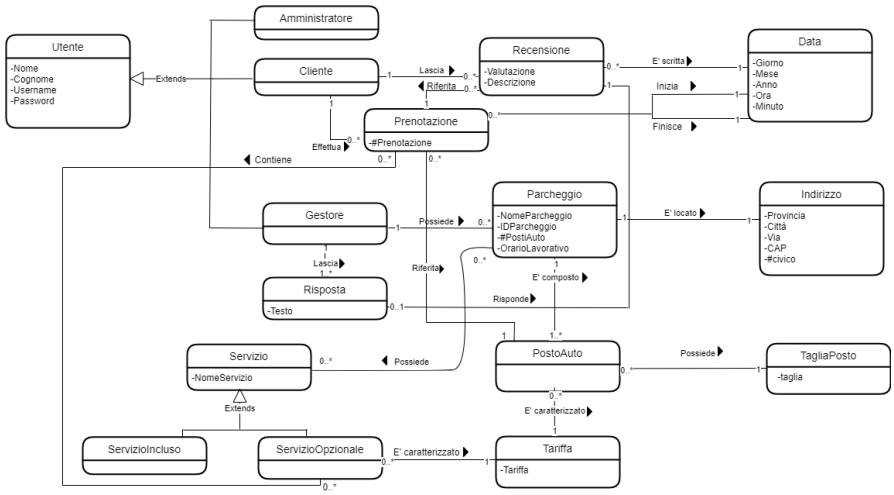

Figura 3.1: Modello logico

caso interessante è quello della taglia: modella la dimensione del posto che il gestore può mettere a disposizione e può scegliere tra quelle che propone la piattaforma. Anche in questo caso questo rappresenta un vero e proprio concetto della realtà del dominio, infatti non può essere espresso semplicemente come un testo.

Nella modellazione logica è stato usato il concetto di *generalizzazione* sia nel caso degli utenti, che possono essere di tre tipologie ongnuna delle quali specifica il ruolo che l'utente ha all'interno della piattaforma, sia nel caso dei servizi, che possono avere o meno una tariffa in base alla loro natura (opzionale o meno). In entrambe le situazioni la generalizzazione è stata utilizzata per estendere un concetto e ricavarne molteplici casi che si differenziano l'uno dall'altro.

3.2 Casi d'uso

Dopo aver analizzato il dominio applicativo con la modellazione logica, per avere una comprensione ulteriore del dominio stesso si è passati all'analisi dei casi d'uso.

Abbiamo individuato i seguenti casi d'uso:

Ricerca

Precondizioni : non sono presenti, anche un cliente non registrato può farlo.

Scenario principale di successo:

1. Il cliente richiede al sistema di cercare un parcheggio.
2. Il cliente specifica i parametri per la ricerca: luogo, durata della sosta (data/ora arrivo, data/ora partenza), taglia del posto.
3. Il sistema ricerca e mostra tutti i possibili parcheggi disponibili. Per ogni parcheggio vengono fornite informazioni parziali per un'anteprima del parcheggio.
4. Il cliente sceglie il parcheggio.
5. Il sistema fornisce i dettagli del parcheggio.

Prenotazione

Precondizioni : Il cliente deve essere loggato.

Scenario principale di successo:

1. Il cliente richiede al sistema di cercare un parcheggio.
2. Il cliente specifica i parametri per la ricerca: luogo, durata della sosta (data/ora arrivo, data/ora partenza), taglia del posto.
3. Il sistema ricerca e mostra tutti i possibili parcheggi disponibili. Per ogni parcheggio vengono fornite informazioni parziali per un'anteprima del parcheggio.
4. Il cliente sceglie il parcheggio.
5. Il sistema fornisce i dettagli del parcheggio.
6. Il cliente decide di prenotare scegliendo i servizi che gradisce.

7. Il cliente conferma i dati.

8. Il sistema effettua la prenotazione.

Elimina prenotazione

Precondizioni : Il cliente deve essere loggato.

Scenario principale di successo:

1. Il cliente si trova nella pagina delle prenotazioni future.

2. Il cliente sceglie una prenotazione.

3. Il sistema fornisce un riepilogo con bottone elimina.

4. Il cliente preme il tasto elimina.

5. Il sistema elimina la prenotazione e mostra un messaggio di conferma.

Inserimento recensione

Precondizioni : Il cliente deve essere loggato ed almeno una prenotazione deve già essersi conclusa.

Scenario principale di successo:

1. Il cliente si trova nella pagina delle prenotazioni passate.

2. Il cliente sceglie una prenotazione.

3. Il sistema fornisce un riepilogo con bottone recensisci.

4. Il cliente clicca sul bottone per recensire il parcheggio.

5. Il sistema apre una finestra da compilare con la recensione e la valutazione.

6. Il cliente compila e conferma la recensione.

7. Il sistema salva la recensione.

Amministratore Moderazione recensioni

Precondizioni : L'amministratore deve essere loggato.

Scenario principale di successo:

1. L'amministratore apre la lista delle recensioni.

2. L'amministratore seleziona la recensione e decide se eliminarla o meno.

3. Il sistema elimina la recensione.

Gestore gestisce i parcheggi

Precondizioni : Il gestore deve essere loggato.

Scenario principale di successo:

1. Il gestore apre la lista dei suoi parcheggi.

2. Il gestore seleziona il parcheggio che desidera.

3. Il sistema apre una pagina per la gestione del parcheggio.

4. Il gestore può decidere di effettuare le operazioni messe a disposizione.

5. Il sistema effettuerà tali operazioni.

Utente Area Personale
Precondizioni : Il cliente deve essere loggato.
Scenario principale di successo:

1. Il cliente apre l'area personale.

2. Il sistema mostra l'area personale con tutte le operazioni che il cliente può effettuare, sia sulle sue prenotazioni che sulle informazioni personali.

3. Il cliente può decidere di effettuare le operazioni messe a disposizione.

4. Il sistema effettuerà tali operazioni.

Dopo aver individuato i casi d'uso più generali, abbiamo disegnato su carta, per ognuno di essi, uno sketch di interfacce che si sarebbero susseguite ad una sequenza di click. Mostriamo con le figure 3.2 e 3.3 lo sketch relativo al caso d'uso della **prenotazione**.

3.3 Diagrammi di sequenza

Arrivati a questo punto per sviscerare completamente la realtà d'interesse, siamo passati alla rappresentazione dei casi d'uso mediante l'utilizzo di diagrammi di sequenza. Nei diagrammi di sequenza, per avere informazioni sul funzionamento dell'applicazione, abbiamo esploso alcuni casi d'uso in più diagrammi di sequenza. Il risultato di questa modellazione è riportato nelle figure 3.4 e 3.5.

3.4 Conclusioni Analisi

Arrivati alla fine della fase di analisi, abbiamo in mano un quadro completo della realtà d'interesse che ci permette di sviluppare e progettare al meglio l'applicazione web. Inoltre alcuni di questi strumenti che abbiamo utilizzato ci hanno fornito dei risultati che, più tardi nello sviluppo del progetto, ci saranno utili.

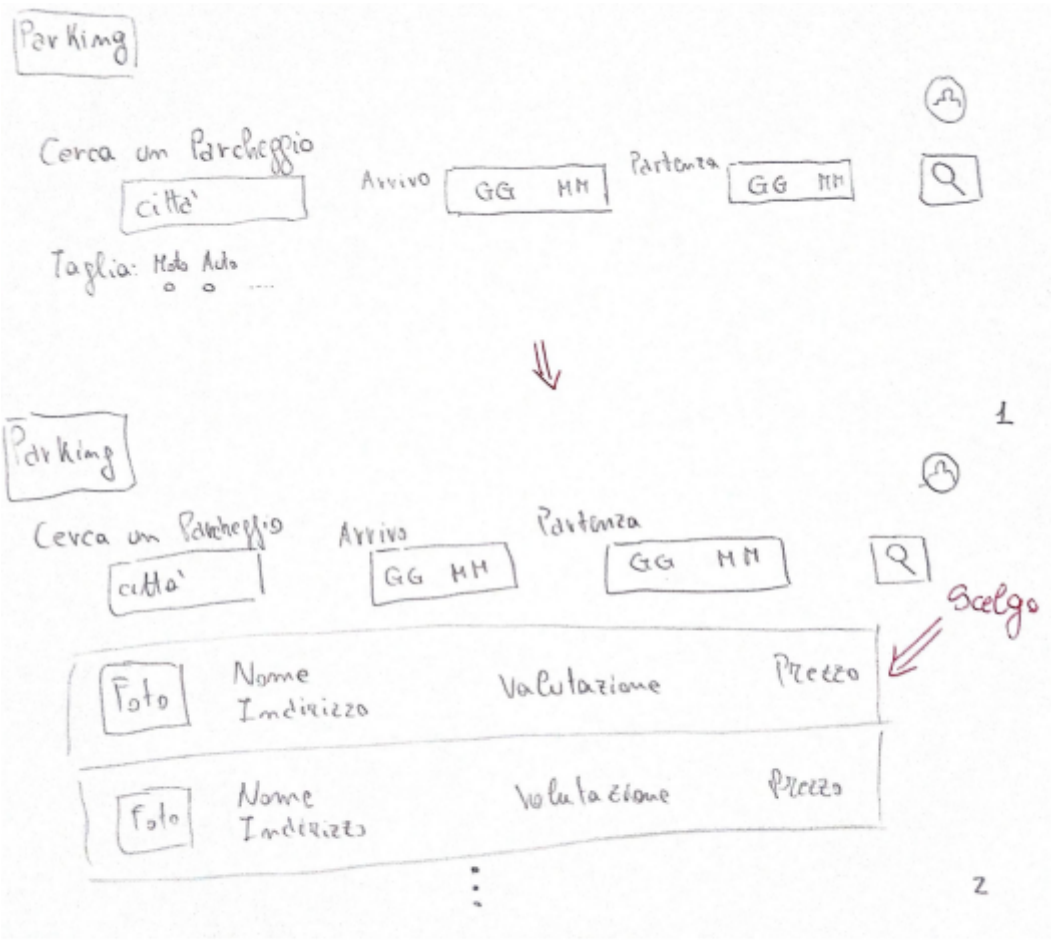

Figura 3.2: Caso d'uso, parte1

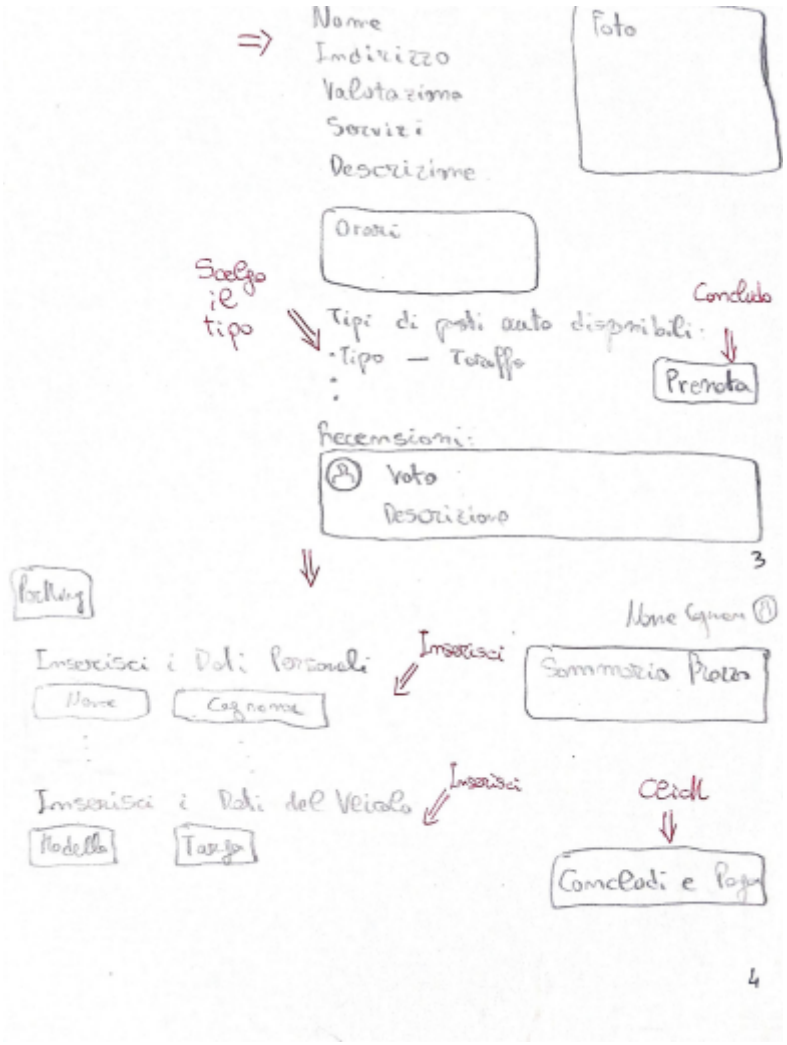

Figura 3.3: Caso d'uso, parte 2

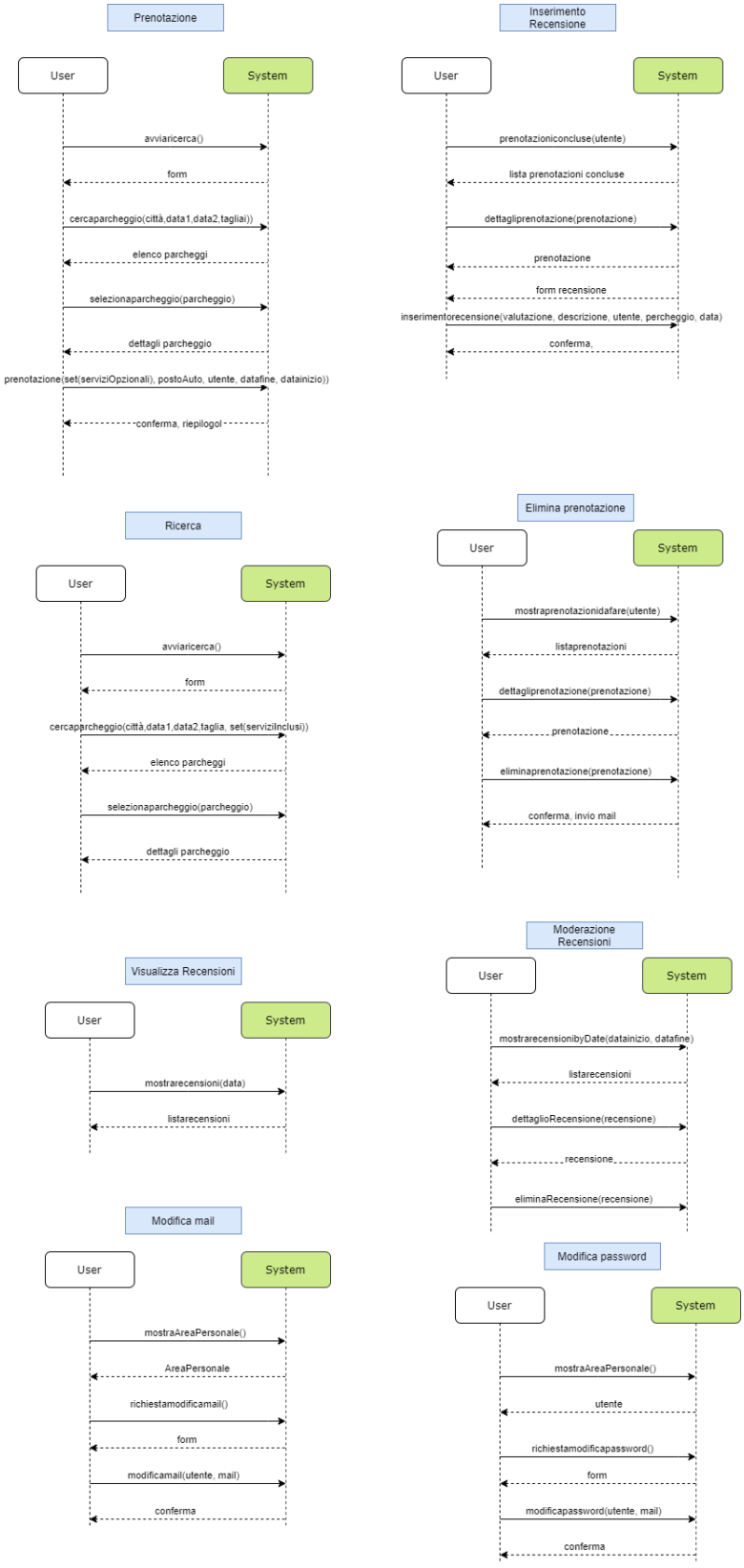

Figura 3.4: Diagrammi di sequenza 1

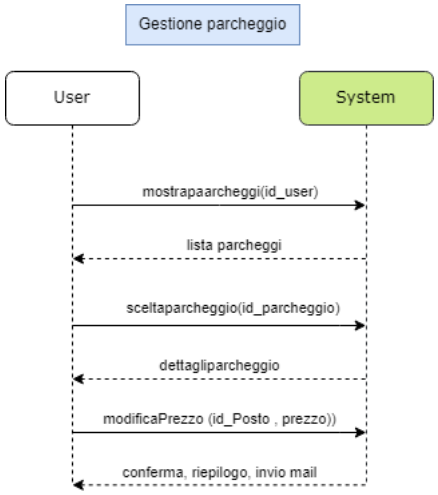

Figura 3.5: Diagrammi di sequenza 2

Capitolo 4

Design del sistema

Nel seguente capitolo si tratterà di come, una volta effettuata la fase di analisi ed ottenuti i risultati, da questi si derivino due fondamentali elementi della progettazione software che sono il **diagramma delle classi** e lo **schema del Database**.

4.1 Diagramma delle classi

Il diagramma delle classi rappresenta la linea guida da seguire per la modellazione, attraverso il software, del domino applicativo. Siamo partiti dal modello concettuale ottenuto in fase di analisi e ne abbiamo ricavato le classi mostrate nella figura 4.1.

4.2 Schema del Database

Per la progettazione di un'applicazione software si ha bisogno di progettare sia strutture dati per mantenere le informazioni in RAM durante l'elaborazione, sia strutture dati per il mantenimento della persistenza delle informazioni. Dopo aver progettato le classi che modellano la realtà d'interesse, abbiamo trovato uno schema per il database che ci permettesse di salvare lo stato degli oggetti. Lo schema del database è riportato nella figura 4.2. Abbiamo eseguito un mapping da classi a modello relazionale seguendo i principi di tali trasformazioni, in particolare, nel caso delle gerarchie di classi, abbiamo riassunto l'intera gerarchia di classi in un'unica tabella, lasciando le tabelle specifiche delle classi che sono coinvolte nella gerarchia, come si può osservare nel caso della tabella **utente** e **servizio**.

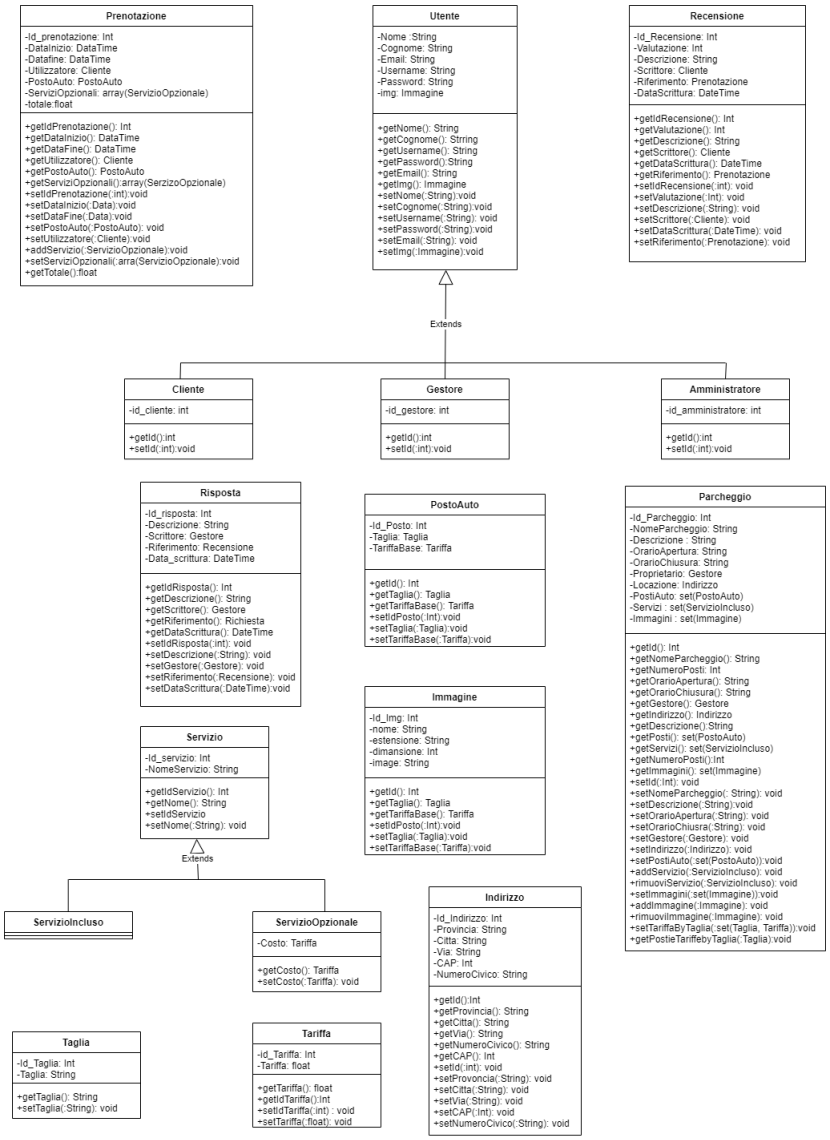

Figura 4.1: Classi

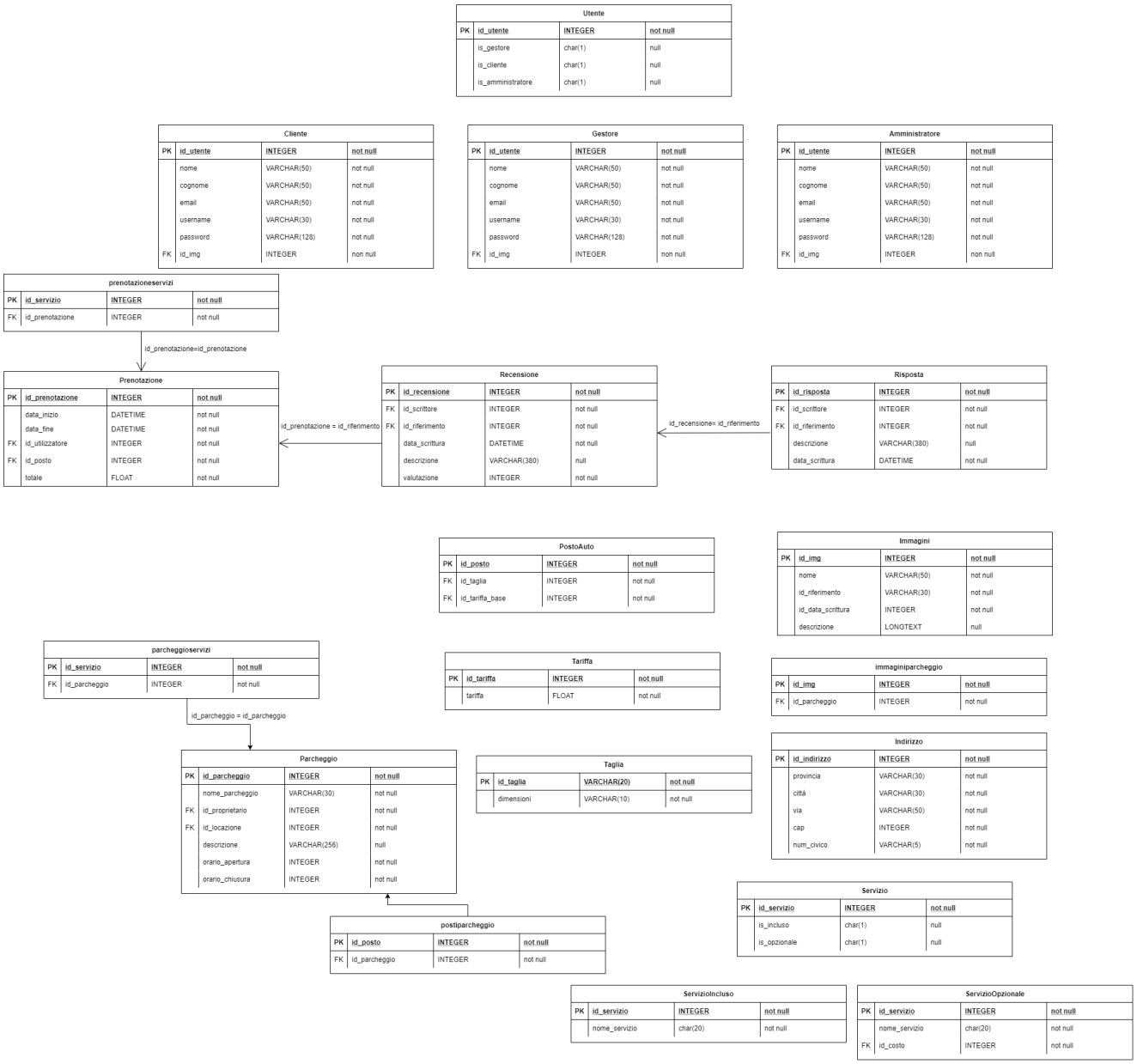

Figura 4.2: Schema del Data Base

Capitolo 5

Progettazione PCEF

Per la progettazione del codice, abbiamo fatto riferimento ai principi di **basso accoppiamento ed alta coesione**, nei quali si specifica di creare classi che abbiano un obiettivo ed un compito ben preciso e che siano connesse il meno possibile tra di loro. Seguendo questi principi si avranno grandi vantaggi in termini di riusabilità del codice e di manutenzione.

Un altro principio che è stato rispettato è quello del **SoC** (Separation of concerns), nel quale un programma viene separato in sezioni distinte, ognuna delle quali responsabile di un ben preciso compito. Nel nostro caso abbiamo utilizzato un **Layered Design**, conosciuto come **PCEF**. Con questo pattern si distinguono quattro diversi **strati** o **sottosistemi**, che sono:

- **Foundation** responsabile della persistenza dei dati, conosce l'implementazione a livello fisico dei dati e si interfaccia con il DB.
- **Entity** definisce gli oggetti specifici del domino applicativo e implementa i requisiti funzionali.
- **Control** definisce classi che disaccoppiano gli strati View ed Entity, conoscono quali classi implementano un determinato caso d'uso.
- **Presentation / View** contiene le classi che realizzano la GUI, permettendo le *human-computer interactions*.

Nelle seguenti sezioni andremo ad esporre le scelte fatte per ognuno di questi strati.

5.1 Foundation

In questo layer abbiamo utilizzato l' approccio base che consiste nel progettare una classe foundation per ogni classe del dominio, con la responsabilità di inviare al DB query relative agli oggetti della classe Entity relativa.

Una prima scelta progettuale da sottolineare è che, essendo che non servono ogetti diversi dalle classi stesse di Foundation, c'era la possibilità di perseguire due strade; quella dell'approccio tutto statico, cioè lavora con metodi statici di classe e non si creano oggetti della classe stessa, o quella dell'approccio Singleton. Noi abbiamo scelto di utilizzare la seconda strategia.

L'approccio **Singleton** prevede che la classe abbia un unico costruttore privato, in modo da impedire l'istanziazione diretta della classe. La classe fornisce inoltre un metodo "getter" statico

che restituisce l'istanza della classe, che è sempre la stessa, creata alla prima chiamata del metodo, e memorizzandone il riferimento in un attributo privato anch'esso statico.

Per quanto riguarda l'accesso a MySQL via PHP, abbiamo scelto di utilizzare PDO (Portable Data Objects). Questo rappresenta uno strato di astrazione, specifico per PHP, che fornisce API riguardo qualsiasi operazione verso il DB al quale ci si connette. In PDO si crea un oggetto connection, al quale si specificano i parametri della connessione, si usa quello per l'interazione con la base dati. Un vantaggio significativo di quest'approccio è quello che se un giorno si volesse cambiare la tecnologia DBMS della base dati, sarà sufficiente cambiare solamente cambiare un parametro nella connessione, al contrario di quanto accadrebbe con tecnologie diverse da PDO.

Nel nostro caso abbiamo specificato le variabili di connessione in un file separato che prende il nome di 'config_db.php', che verrà richiamato quando si crea l'oggetto connessione. Facendo in questo modo abbiamo un unico punto nel quale possiamo settare i parametri.

Le classi Foundation condividono i metodi relativi alle cosiddette operazioni CRUD, oltre ad avere metodi speifici per ottimizzare il funzionamento dell'applicazione. Per la progettazione dello strato 'F' abbiamo creato una superclasse FDb dalla quale tutte le classi di Foundation ereditano i metodi. In questa sono definiti gli attributi: una variabile che contiene l'oggeto connection verso il db, un attributo che rappresenta un array di istanze (singole per ogni classe) che viene utilizzato per l'approccio singleton, ed altri attributi che nelle singole classi che estendono FDb verranno impostati e saranno utili nell'interazione con il DB. Questi ultimi manterranno informazioni inerenti la tabella da utilizzare, la chiave primaria e se la tabella implementa l'autoincrement o meno.

I metodi di questa superclasse corrispondo alle operazioni basilari e generiche che si possono fare verso un db, come store, load, update, delete, e search. Tutti questi restituiscono degli oggetti di tipo statement, che poi verranno processati da altri metodi che sono previsti all'interno della classe, in base alle necessità che ci si pongono di fronte.

Tra questi ultimi metodi abbiamo deciso di inserire il metodo 'getObject' che utilizza il fetch di PDO chiamato 'PDO::FETCH_CLASS'; questo ricostruisce un oggetto della classe che si specifica a partire dalla tabella sul db. Un requisito che si deve soddisfare per usare questo tipo di fetch è quello di avere costruttori vuoti ed attributi pubblici, oltre a definire come attributi le colonne della tabella specifica per ogni classe. Per sfruttare questa funzionalità nelle classi Entity abbiamo rispettato i suddetti requisiti.

Per quanto riguarda l'esecuzione di query sul db è stato progettato un apposito metodo che, una volta passata una stringa che rappresenta una query, la esegue prima facendo un **prepare** dello statement, operazione che serve per ottimizzare l'esecuzione della query stessa, e successivamente, con il metodo **execute** di PDO, si verifica l'esecuzione vera e propria.

Tutto ciò viene inserito all'interno di un costrutto **TRY-CATCH** per verificare il sollevarsi di eccezioni relative alla query. Nell'esecuzione delle operazioni in Foundation è stato implementato il meccanismo delle **Transaction** in PHP, così da rispettare le proprietà ACID.

Per quanto riguarda la problematica del' 'SQL Injection', abbiamo individuato l'unico punto nel quale sarebbe stato possibile inserire testo per poter avere un'iniezione di codice volta al trafugare dati dal DB; questo è il form di ricerca. A tal proposito abbiamo utilizzato in questo contesto le potenzialità della funzione PDO::prepare(), nella quale si vanno ad utilizzare dei placeholder che nell'esecuzione verranno sostituiti con le opportune variabili.

La classe FDb successivamente viene estesta da tutte le altre classi Foundation, nelle quali è personalizzato il funzionamento dei metodi principali e, in base alle necessità, sono stati aggiunti metodi opportuni.

5.2 Entity

Nel layer Entity abbiamo implementato le classi che modellano il dominio d'interesse. Inoltre ogni volta che una classe ha come attributo un oggetto di un'altra classe, abbiamo aggiunto all'inizio del suo nome un '_' che ci permette a livello Foundation di eseguire i metodi specifici per quell'oggetto. Come detto in precedenza, il costruttore risulta vuoto e gli attributi pubblici per utilizzare le funzionalità del PDO.

5.3 Controller

I controllori propagano allo strato Entity o direttamente in Foundation lo stimolo ricevuto dalle View da parte dell'utente finale. Essendo che questo stimolo rappresenta un'operazione che l'utente richiede all'applicazione, i controllori vengono definiti per ogni caso d'uso. Il loro compito è quello di coordinare il lavoro delle classi a livello sottostante per soddisfare lo stimolo ricevuto e di rendere il risultato alle classi View specifiche. Per la progettazione dei controller, abbiamo fatto riferimento ai diagrammi di sequenza che avevamo sviluppato in fase di Analisi.

Abbiamo deciso anche qui di utilizzare un' approccio Singleton, visto che non servono oggetti diversi dalle classi stesse di Controller.

Nella progettazione, si è deciso di inserire al loro interno l'interazione con gli oggetti delle classi View, questo perché viene implementato il meccanismo del rewriting delle URL. A fronte di questa implementazione si è dovuto sviluppare un controllore specifico chiamato Front Controller che, richiamato esclusivamente dall'index.php, analizzando l'URL riesce a richiamare il giusto metodo del giusto controllore; per questo motivo si è rivelato necessario avere dei controllori che utilizzassero oggetti delle classi View al loro interno e non il contrario.

Nel layer Controller è stata creata una classe per ogni caso d'uso ed un metodo per ogni evento di interazione dell'utente con il sistema.

5.4 View

Le classi di questo layer implementano la GUI e realizzano l'interfaccia allo scopo di acquisire l'input e visualizzare l'output. Per rispettare questi principi, abbaiamo sviluppato una classe View per ogni caso d'uso ed un metodo per ogni operazione di input ed output che si va ad eseguire.

5.5 Autoloader

Tramite questo pattern di progettazione, saranno presenti molti oggetti di classi differenti che interagiscono l'uno con l'altro e, in alcuni casi, sono uno un attributo dell'altro. A questo scopo è stato progettato un meccanismo di autoloading, inserito in 'autoloader.php' che partendo dalla prima lettera della classe (F, E, C, V) va a rintracciare ed includere il file di cui si necessita.

Capitolo 6

Scelte progettuali

6.1 Package 'www'

Per quanto riguarda le tecnologie web, abbiamo creato una cartella che prende il nome di 'www' dove abbaimo inserito tutto ciò che concerne quest'ambito.

Abbiamo cercato un template bootstrap che fosse il più possibile simile alle interfacce che in fase di analisi avevamo ideato, o che comunque fosse il più possibile funzionale ai nostri scopi.

Una volta trovato il template giusto si è passati alla fase di personalizzazione di quest'ultimo, creando tutte le pagine che sarebbero state necessarie, seguendo uno stile comune tra tutte e cercando di farle nostre con l'aggiunta di HTML e CSS. Inoltre sono state aggiunte diverse funzioni in JavaScript per la validazione dei form, per il controllo sulle date, e per il controllo sull'inserimento dei file (es. nel caricamento dell'immagine del profilo).

Per quanto riguarda la generazione delle pagine in base ai dati che vengono passati dai layer inferiori, abbiamo utilizzato il template-engine Smarty. Un template è uno strumento che disaccoppia i dati prodotti da Entity dalle possibili differenti implementazioni gestite da View. La presentazione in ambito web è costituita da un testo contenente i dati da visualizzare e la presentazione dei dati fatta attraverso HTML. I template sono modelli che vengono creati a partire da linguaggi di descrizione. Smarty rappresenta un template-engine, cioè un pre-processore per la fusione di dati e template.

Abbiamo creato tutti i template, file con estensione '.tpl', che ci sarebbero stati necessari, sfruttando le funzionalità di Smarty, e successivamente abbiamo fatto agire le classi View con degli appositi attributi $smarty; questi, al momento della creazione di un oggetto delle classi View, vengono inizializzati e configurati utilizzando un'apposita classe che prende il nome di 'StartSmarty.php'.

6.2 Sessioni

Il meccanismo delle sessioni va in contro alle esigenze delle applicazioni che vogliono mantenere lo stato di una connessione, anche basandosi su HTTP, un protocollo stateless. Una sessione gestisce l'interazione tra un browser ed un server web. Le informazioni che si decidono di mettere in sessione saranno archiviate nel server e saranno accessibili solamente utilizzando un **session id**, cioè un identificativo univoco associato alla sessione.

Nella progettazione dell'applicazione web abbiamo deciso di utilizzare le sessioni in diverse occasioni. Il primo caso è quello in cui si accede, tramite login, all'area personale, sia per un cliente, un gestore o un amministratore. Una volta che il login è avvenuto con successo vengono salvati sia l'id dell'utente (univoco), sia la tipologia di utente. Quest'ultima viene salvata perché in base al tipo di utente che sta interagendo con l'applicazione, si avranno interfaccia e funzionalità differenti. Quando l'utente effettuerà il logout, ci sarà sia un **unset** della sessione, cioè verranno cancellate le informazioni della sessione, e sia un **destroy** per eliminare la sessione.

Nel secondo caso, abbiamo utilizzato il meccanismo delle sessioni nel box di ricerca. Questa scelta è dovuta alla volontà di aumentare la praticità di utilizzo dell'applicazione, mantenendo i valori nei campi compilati precedentemente dal cliente. Quindi quando si cerca un parcheggio si andranno a salvare in sessione i dati immessi riguardanti la città, le date di arrivo e la taglia del posto e queste verranno ripristinate nella pagina di risultati, nel box di ricerca che si può utilizzare in caso si voglia modificare o effettuare una nuova ricerca.

Infine, utilizziamo il meccanismo delle sessioni per salvare i dati di un prenotazione che non è ancora stata confermata così, nel caso in cui il cliente debba effettuare il login ed una volta completato, abbia il riepilogo della prenotazione che aveva effettuato precedentemente e non debba ricominciare da zero.

Per l'utilizzo delle sessioni abbiamo creato una classe apposita che prende il nome di 'Sessioni.php', con dei metodi che ci permettono di effettuare qualsiasi operazione con una sessione. La ragione dell'implementazione di questa classe sta nel rispettare il principio di basso accoppiamento della progettazione software, in questo caso il meccanismo delle sessioni è separato dal resto del programma e ci si interfaccia solamente attraverso l'apposita classe.

Una decisione progettuale che abbiamo preso è stata quella di modificare il tipo di cookie che contiene il 'PHP_SES_ID', e impostargli una scadenza dopo 20 minuti, così da evitare che ci siano sessioni troppo prolungate nel tempo.

Un ultimo aspetto riguardante le sessioni è il garbage collector attivato da Apache sul server. Il comportamento di questo sistema si può modificare con l'istruzione 'session.gc_maxlifetime', nel caso del nostro progetto abbiamo deciso di mantenere il comportamento standard che dice di richiamare il garbage collector ogni 24 minuti.

6.3 URL rewriting

Il meccanismo di URL rewriting è quello che permette di avere URL descrittive rispetto alla situazione che sta accadendo e ha il compito di nascondere tutto ciò che l'utente finale dell'applicazione non deve sapere, come il nome dei file che vengono utilizzato o addirittura la loro estensione (per esempio '.php' fornisce ad un attaccante esterno l'informazione che il linguaggio utilizzato è PHP).

Nel nostro caso abbiamo deciso di implementare il meccanismo di riscrittura delle URL.

Abbiamo creato un file .htaccess, cioè un file di configurazione distribuita. Questo file contiene le direttive riguardanti questo meccanismo che dicono di attivare il rewriting, successivamente ci sono delle condizioni che verificano se nel browser siano presenti la directory o il file interessati, poi c'è la condizione di base per la quale si inizia il rewriting dall'inizio del URL, e successivamente c'è la regola di rewriting che dice che tutte le URL devono essere indirizzate verso il file con nome 'index.php'.

Quest'ultimo è un file molto semplice, nel quale si richiama il controllore FrontController e si esegue il suo metodo 'run', al quale gli si passa il '$_SERVER['REQUEST_URI']', cioè il path.

Il FrontController con il suo metodo 'run', analizza l'URL e indirizza la richiesta verso il giusto controllore.

Capitolo 7

Conclusioni

7.1 Sviluppi futuri

Le prospettive future di questo progetto potrebbero essere rivolte verso un approccio di tipo Web Service, cioè un sistema software progettato per supportare l'interoperabilità tra diverse applicazioni in un contesto distribuito. A questo scopo si implementerà la notazione JSON così da avere un'architettura estesa aggiungendo client sviluppati ad-hoc per particolari devices. Sarà fondamentale riscrivere le URL in maniera che rispettino i requisiti REST.

7.2 Conclusione

La realizzazione di questo progetto si è dimostrata fondamentale per comprendere quanto incidano i processi di analisi, design e progettazione nello scenario della programmazione web. Il lavoro di gruppo unito alla puntuale documentazione e analisi dei problemi si sono rivelati la chiave vincente per affrontare lo sviluppo della nostra web app.