Università degli Studi di Torino

Dipartimento di Informatica

Corso di Laurea in Informatica

Anno Accademico 2019/2020

Sviluppo di un'Applicazione Web Based per la gestione, la configurazione e il test di gru mobili

Relatore: Prof.ssa Susanna Donatelli

Tutor: Dr. Giorgio Ponza

Candidato

Luca Olivelli

Indice

0. Dichiarazione di responsabilità

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

1. *Prefazione*

Il progetto da me sviluppato durante le ore di tirocinio svolto presso “Movimatica” ha avuto come obbiettivo la realizzazione un applicativo web che permettesse al suo utilizzatore la gestione e il test di gru mobili.

La web-app, realizzata per “JMG Cranes” azienda italiana produttrice di gru semoventi 100% elettriche, fornisce agli utenti una visione ed una gestione semplificata del database, in cui sono immagazzinate tutte le gru mobili che l’azienda produce. Con questa app è possibile gestire facilmente il database: aggiungendo, modificando o eliminando a proprio piacimento le macchine, con i relativi supporti tecnici per le gru, che l’azienda produce.

L’applicazione fornisce, inoltre, una sezione dedicata al calcolo dei valori di carico e di pressione a cui sono sottoposte le gru, durante la fase di sollevamento, utile per testare in modo virtuale le prestazioni delle macchine presenti nel database. In questa sezione si sceglie la configurazione di gru mobile che si intende testare, comprensiva di zavorre e accessorio, e si inseriscono i dati del carico che si intende sollevare; dopodiché l’applicazione rimanda ad una pagina in cui sono mostrati tutti i valori relativi al carico esercitato sugli assi e/o sugli stabilizzatori, la superficie di contatto degli pneumatici e la pressione al suolo delle ruote. I valori di pressione e carico del che vengono mostrati nella pagina finale di questa sezione sono il risultato di calcoli effettuati da una classe Java costruita appositamente sulla base delle formule che ci sono state fornite dalla JMG.

La progettazione e la realizzazione dell’applicazione sono state fortemente improntate ad uno sviluppo che fosse basato sul pattern *MVC* (*Model-View-Controller*). Questo al fine di garantire una separazione netta tra le parti che interagiscono tra loro per il corretto funzionamento dell’intero progetto; inoltre tale implementazione favorirà il mantenimento e la modifica dell’applicativo da parte dei progettisti che vorranno mettervi mano in futuro.

Tra le tecnologie utilizzate per lo sviluppo troviamo

a) Le Servlet che fanno da controller dell’applicazione;

b) Le pagine JSP (JavaServer Pages) che si occupano della interazione con l'utente e che quindi svolgono il ruolo della view.
c) Le classi oggetto Java, le tabelle del database e la classe *DAO* (*Data Access Object*) per la rappresentazione e la manipolazione degli oggetti relativi alle gru mobili e ai loro componenti che costituiscono il modello del progetto.

L'intera permanenza in azienda, durata quasi tre mesi, anche se in buona parte svolta in modalità smart working, a causa delle stringenti normative di sicurezza dovute alla pandemia di covid-19, è sempre stata accolta positivamente da colleghi e superiori gentili e disponibili.

Sotto l'aspetto esperienziale, invece, mi è stata data ampia libertà di scelta per quello che riguardasse la realizzazione del progetto, salvo per alcune direttive fornitemi dal mio responsabile nonché tutor aziendale il Dott. Giorgio Ponza. Questo mi ha permesso da un lato di poter esprimere a pieno le mie capacità creative, per quanto riguarda la parte di sviluppo front-end; dall'altro la possibilità di incrementare le mie competenze da programmatore back-end, durante le fasi di soluzione dei problemi.

2. Progettazione dell'applicativo

2.1 Analisi e selezione dei dati

Il primo passo necessario per la realizzazione del progetto è stato quello relativo alla identificazione e selezione dei dati rilevanti per l'azienda destinataria del progetto (ad esempio le specifiche tecniche di gru, accessori e zavorre, le formule per il test virtuale delle gru, ecc…). Dopo l'analisi dei dati necessari per alla realizzazione dell'applicazione si è deciso di costruire i seguenti oggetti:

- Le famiglie di gru mobili
- Le configurazioni delle macchine
- Gli accessori con cui equipaggiare le gru
- Le zavorre da montare come contrappeso

Per definire meglio il concetto di famiglie di gru, si potrebbe fare un paragone con le automobili: dove le tipologie di auto che una casa automobilistica produce (ad esempio per la FIAT abbiamo la 500, la Panda, la Tipo, ecc…) rappresentano i nomi delle famiglie di gru prodotte da JMG. I nomi delle gru mobili però, a differenza di quelli delle automobili, sono meno artistici e ricercati, ma più tecnici e specifici (tra questi troviamo: MC25S, MC300, MC580S, ecc…). In questo modo, come riportato nella figura 1, si vanno a categorizzare le gru sulla base delle dimensioni e della capacità di carico della stessa, oppure sulla base della presenza o della cabina di pilotaggio, identificando quindi quelle gru che vengono pilotate da remoto tramite radiocomando.

Le configurazioni dei mezzi, invece, sono una caratterizzazione più specifica e completa delle gru. Tornado al paragone automobilistico, così come una stessa gamma di autovetture ha diversi modelli che variano a seconda del numero di porte, delle dimensioni dell'abitacolo o dell'assetto dell'auto (ad esempio nella famiglia della Fiat 500 vi fanno parte la 500 "base", la 500L e la 500X). Allo stesso modo, per una stessa famiglia di gru mobili, sono disponibili diverse configurazioni a seconda che la gru si appoggi sugli pneumatici oppure sugli stabilizzatori.

Modello Model	Portata max Max Capacity		Lunghezza Lenght Gruppo Chiuso Short wheelbase		Gruppo Aperto Long wheelbase		Larghezza Width		Altezza Height	
	kg	lbs	mm	inch	mm	inch	mm	inch	mm	inch
MC25S	2.500	5,500	2.190	86			900	35	1.610	63
MC32S	3.200	7,000	2.430	96			980	39	1.696	67
MC45S	4.500	10,000	2.775	109			1.200	47	1.648	65
MC60S	6.000	13,200	3.195	126			1.380	54	1.845	73
MC100S	10.000	22,000	3.800	150			1.500	59	1.949	77
MC130S	13.000	28,600	4.030	159			1.850	73	1.990	78
MC180S	18.000	39,600	4.535	179			1.990	78	2.000	79
MC250S	25.000	55,000	4.770	188			2.150	85	2.200	87
MC300S	30.000	66,000	4.820	190			2.150	85	2.219	87
MC350S	35.000	77,000	5.170	204			2.250	89	2.295	90
MC450S	45.000	99,000	4.900	193	5.900	232	2.200	87	2.415	95
MC580S	58.000	127,800	5.295	208	6.495	255	2.350	93	2.479	98

Figura 1: tabella delle famiglie di gru radiocomandate

Le diverse configurazioni, inoltre, sono dotate di proprie specifiche tecniche quali il numero di ruote, il peso del mezzo e della batteria, il passo tra l'asse anteriore e quello posteriore, e via discorrendo. In aggiunta a questi dati standard, ne sono presenti anche altri un po' meno comuni che sono indispensabili per calcolare le capacità di carico e le condizioni di ribaltamento della gru. Tra questi troviamo ad esempio le distanze tra i baricentri della macchina o della batteria e i vari punti di interesse della gru come: il fulcro del braccio, lo scudo, il centro di rotazione ecc… di cui si può avere una visione più completa nella figura 2.

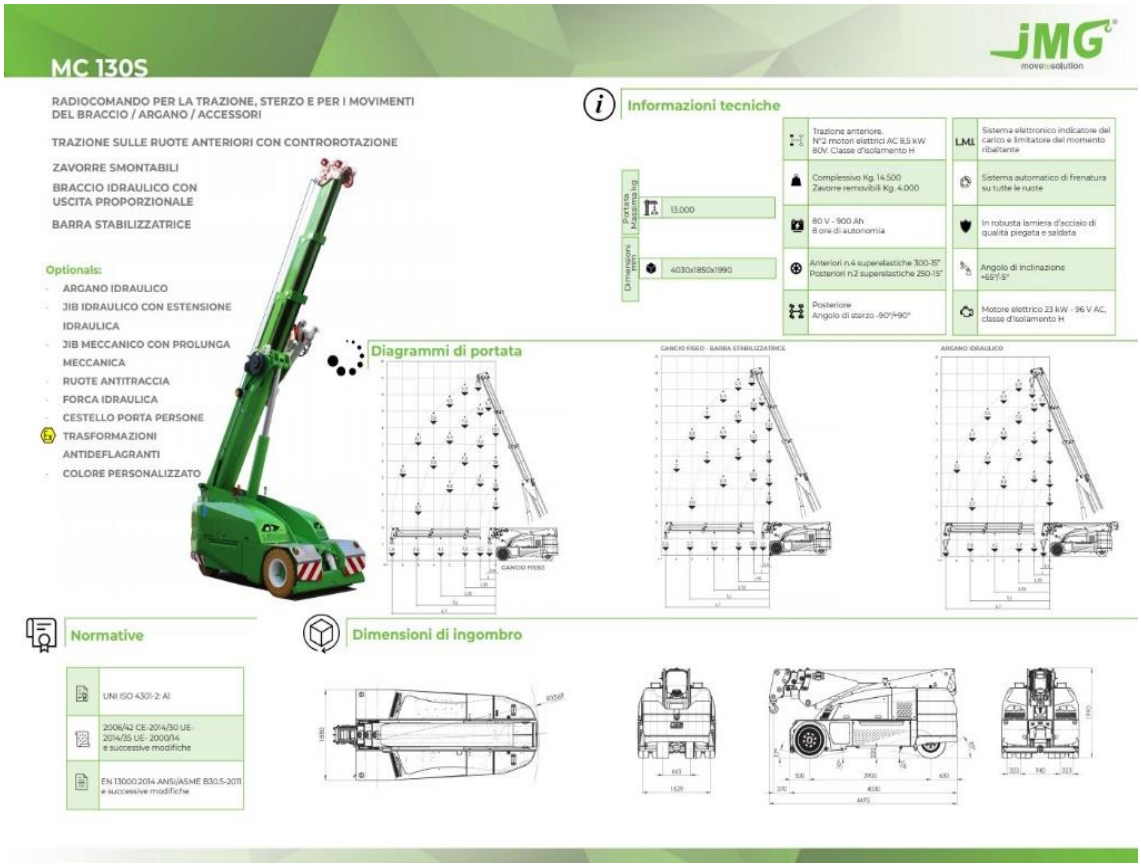

Figura 2: scheda tecnica della gru mobile MC130S

Infine si prendono in considerazione gli accessori e le zavorre che potremmo definire come dei veri e propri optional. Ogni gru mobile, infatti, possiede due ulteriori liste: una per gli accessori ed una per le zavorre, selezionate dei rispettivi elenchi che le contengono tutte presenti nel database/ i cui elementi sono selezionati dalle rispettive tabelle del database, da cui scegliere quali componenti andare ad equipaggiare. In questo modo si può adattare la macchina al meglio per le diverse mansioni o spazi di lavoro.

Le zavorre non sono altro che dei contrappesi da aggiungere, in fase di lavoro, alle macchine allo scopo di stabilizzare la gru durante il sollevamento del carico in modo tale da evitare che quest'ultima si ribalti. Gli unici dati che le caratterizzano sono, quindi, un nome ed un peso che varia in base alla gru su cui viene montata; infatti, le gru più grandi avranno bisogno di un contrappeso maggiore per andare a bilanciare il maggiore carico che sono in grado di sollevare.

Gli accessori, (come forche, jib meccanici o idraulici, prolunghe, …) invece, sono oggetti leggermente più complessi. Questi, infatti, se pur accomunati dal medesimo compito di sollevare oggetti, variano sia nella forma che nella dimensione. Ciò permette alla gru su

cui vengono equipaggiati di adattarsi al lavoro che dovrà andare a svolgere. Come per le zavorre, anche gli accessori, sono dotati sia di un proprio nome identificativo che di un peso, ma in aggiunta, gli accessori, sono stati introdotti altri tre valori: uno per la lunghezza del braccio, uno per la distanza tra l'asse del gancio e il baricentro dell'accessorio ed uno per la posizione della testa. Quest'ultimo valore viene utilizzato solamente per quegli accessori che hanno a disposizione una testa flottante, ovvero che può variare la sua posizione di sollevamento, ed è composto da una serie di posizioni, in cui la testa può essere installata, a cui sono associate le rispettive lunghezze che si andranno poi a sommare alla lunghezza del braccio per ottenerne il valore complessivo. Ovviamente, così come per il valore di peso, anche tutti i dati elencati per gli accessori variano in base alla gru su cui vengono montati.

2.2 Design e strutturazione delle pagine web

Lo step successivo, fatto dopo aver definito tutti gli oggetti necessari al funzionamento dell'applicazione, è stata la scelta del "layout" di come si sarebbero mostrate agli utenti le pagine web.

Per rendere l'utilizzo della applicazione il più semplice, pratico ed efficiente possibile, le pagine web sono state ideate con un design "minimal" che favorisse la semplicità di utilizzo al mero aspetto estetico. Per tale motivo si è optato per la realizzazione di pagine che mostrassero, sotto forma di tabelle o "carte", solo le informazioni essenziali quali: nomi e dati tecnici delle gru, degli accessori e delle zavorre presenti all'interno del database. Inoltre per favorire la navigazione tra le pagine presenti nell'applicazione è stata aggiunta una pagina principale, mostrata nella figura 3, da cui è possibile raggiungere tutte le altre, esattamente come l'indice di un libro.

Un'ultima accortezza a cui ho pensato, volta sempre a migliorare l'esperienza di navigazione delle pagine, è stata quella di inserire una barra di navigazione che mostri sullo schermo quale pagina si stia utilizzando in ogni momento e il percorso a ritroso che si è fatto per arrivarci fino alla pagina principale. In questo modo il suo utilizzatore ha sempre sott'occhio quale operazione sta andando ad eseguire oppure può velocemente tornare sui suoi passi.

Figura 3: pagina principale dell'applicazione

Poiché il fine ultimo dell'applicazione è quello di fornire al cliente un duplice utilizzo, ovvero la gestione del DB e il test delle gru, le pagine web sono state suddivise in due gruppi, in base a ciò che è possibile fare al loro interno:

1. Uno preposto alla gestione e il mantenimento del database (non vincolato da sequenza)
2. L'altro che permetta all'utente di testare in modo virtuale le capacità di carico di una determinata gru attraverso un insieme di pagine, sequenzialmente vincolate.

Il primo gruppo fornisce all'utente una visione completa ed ordinata delle gru, degli accessori e delle zavorre contenute nel database, come mostrato in figura 4. Le pagine web appartenenti a questo gruppo permettono, inoltre, di interagire con i singoli componenti attraverso l'ausilio di bottoni.

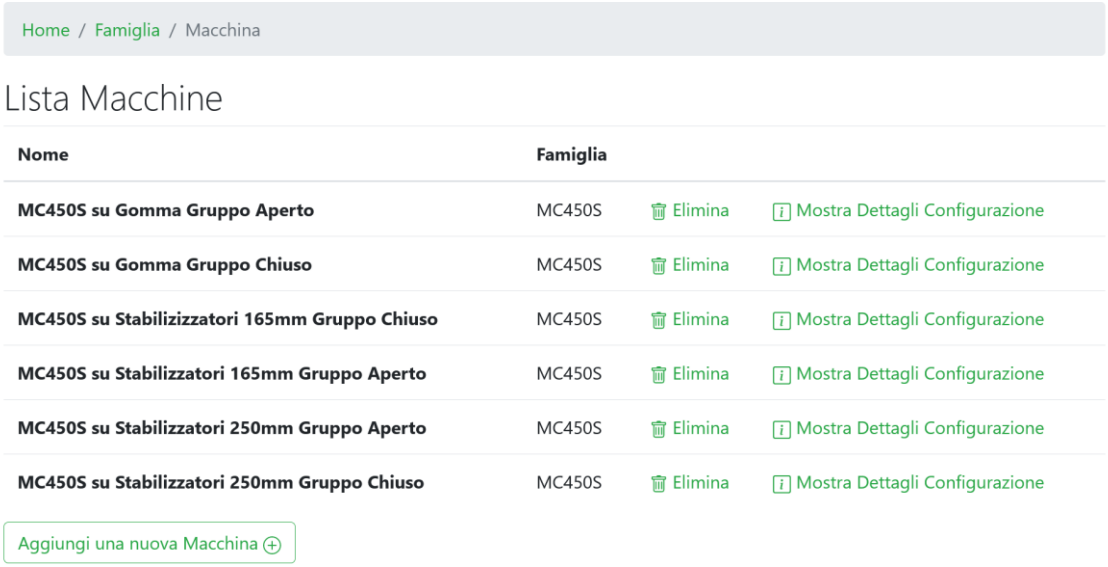

Figura 4: tabella delle gru mobili per la famiglia MC450S

I bottoni, che come visibile sempre in figura 4 possono essere di due tipologie: ovvero link dotati di immagine che riassume il suo utilizzo (le scritte “elimina” e “mostra configurazione”), oppure veri e propri bottoni (il riquadro in basso che riporta “Aggiungi una nuova macchina”), servono per reindirizzare l’utente a pagine ad hoc realizzate appositamente per svolgere le operazioni di inserimento, modifica o eliminazione degli oggetti contenuti nel database. Come esempio si veda la figura 5 che mostra la pagina in cui è possibile modificare i nomi, nelle diverse lingue, e l’immagine per la famiglia delle gru MC130S.

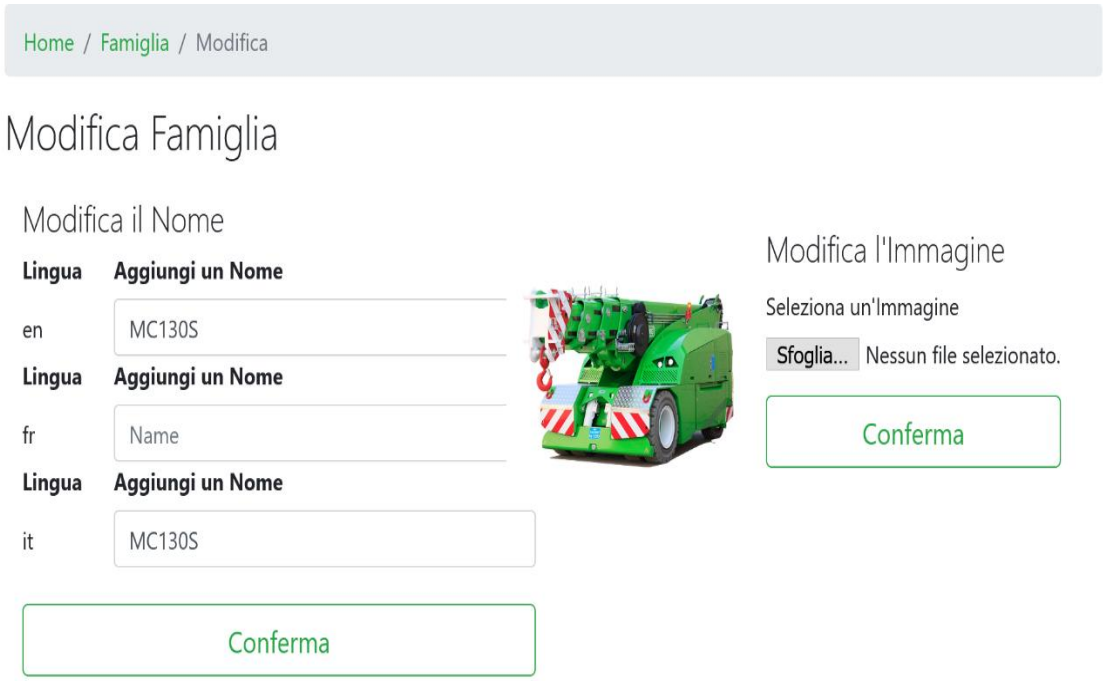

Figura 5: pagina per la modifica del nome e dell'immagine per la famiglia di gru MC130S

In fine allo scopo di rendere più semplice la gestione delle configurazioni, sono state aggiunte delle pagine dedicate proprio alle gru mobili e alle operazioni che è possibile effettuare unicamente nel loro caso (come ad esempio la modifica delle liste di zavorre ed accessori che una gru può equipaggiare). La prima di queste pagine, mostrata in figura 6, riporta al suo interno tutte le specifiche della gru, citate nel capitolo precedente, e le liste degli accessori e delle zavorre, con i relativi dati tecnici, da cui la gru può scegliere quali optional può equipaggiare. Da questa pagina, sempre tramite bottoni, si possono raggiungere tutte le altre pagine in cui è possibile sia modificare i valori della scheda tecnica relativa alla gru mobile presa in considerazione, sia aggiungere, rimuovere o modificare gli elementi all'interno delle liste degli accessori e delle zavorre, come esempio si veda la figura 7 che mostra la pagina da cui è possibile aggiungere una nuova zavorra all'elenco della gru mobile MC400 con stabilizzatori su gomma.

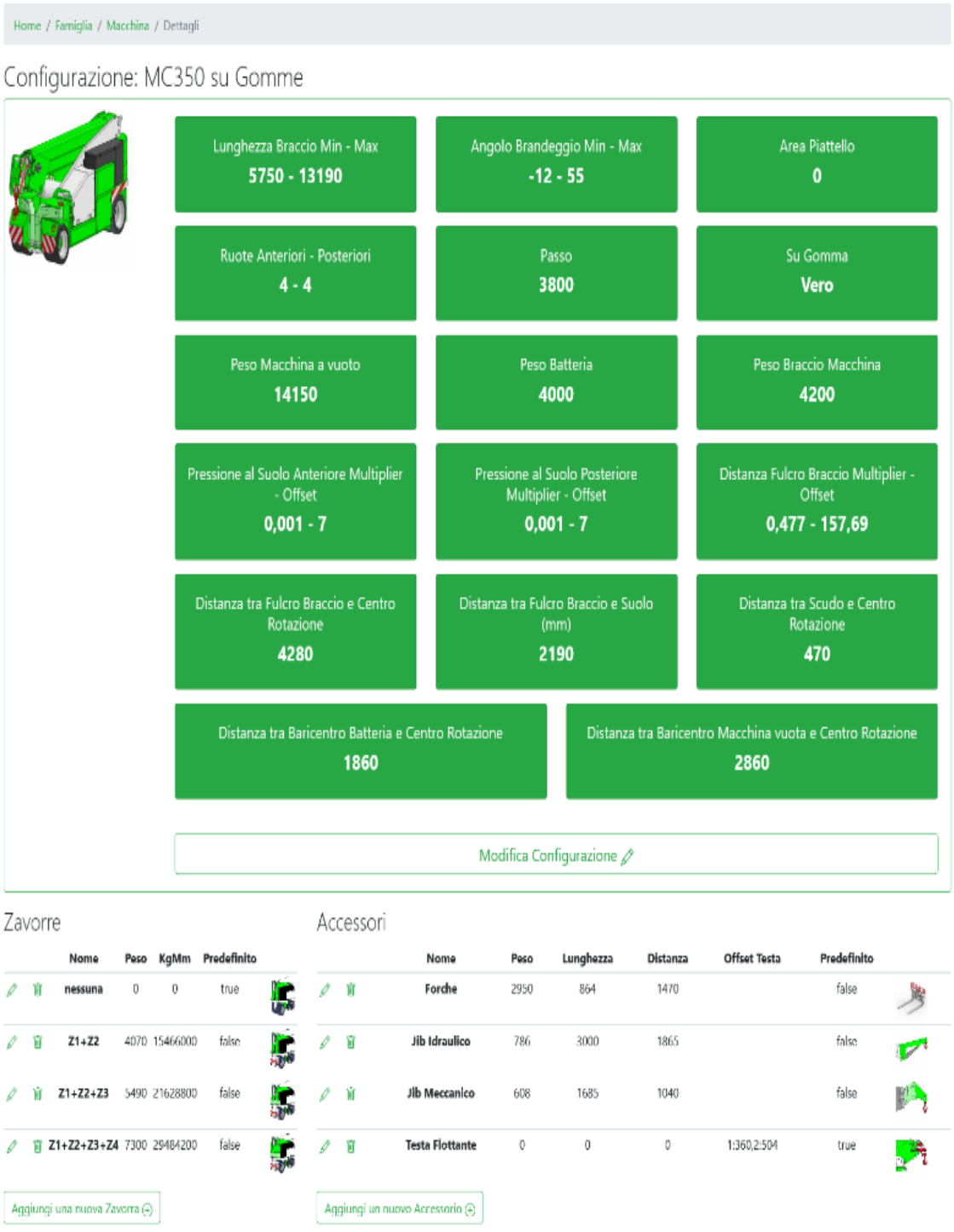

Figura 6: pagina delle specifiche tecniche per la configurazione su gomme della gru MC350S

Figura 7: pagina per l'inserimento di una nuova zavorra alla lista della gru MC400 con stabilizzatori su gomme

Il secondo gruppo, invece, è stato realizzato seguendo un diverso stile di progettazione. In primo luogo l'utente non ha più la facoltà di navigare liberamente tra le pagine, come nel caso del gruppo di pagine per la gestione del database, ma bensì è vincolato ad una sequenza predefinita pagine nella quale l'utente, passo dopo passo, configura la gru mobile che successivamente andrà a testare. L'altra caratteristica che contraddistingue queste pagine dal gruppo precedente è la scelta del design: tutte le pagine della sequenza, infatti, non sono più dotate delle tabelle che mostrano i nomi e i dettagli delle gru e dei suoi componenti che sono state sostituite da delle cards, letteralmente delle carte con tanto di immagine dell'oggetto che sta prendendo in considerazione come si può vedere dalla figura 8. Le cards si comportano esattamente come i bottoni, esse, infatti, permettono di selezionare l'elemento che si sta cliccando e di passare alla pagina successiva per la scelta dell'elemento successivo.

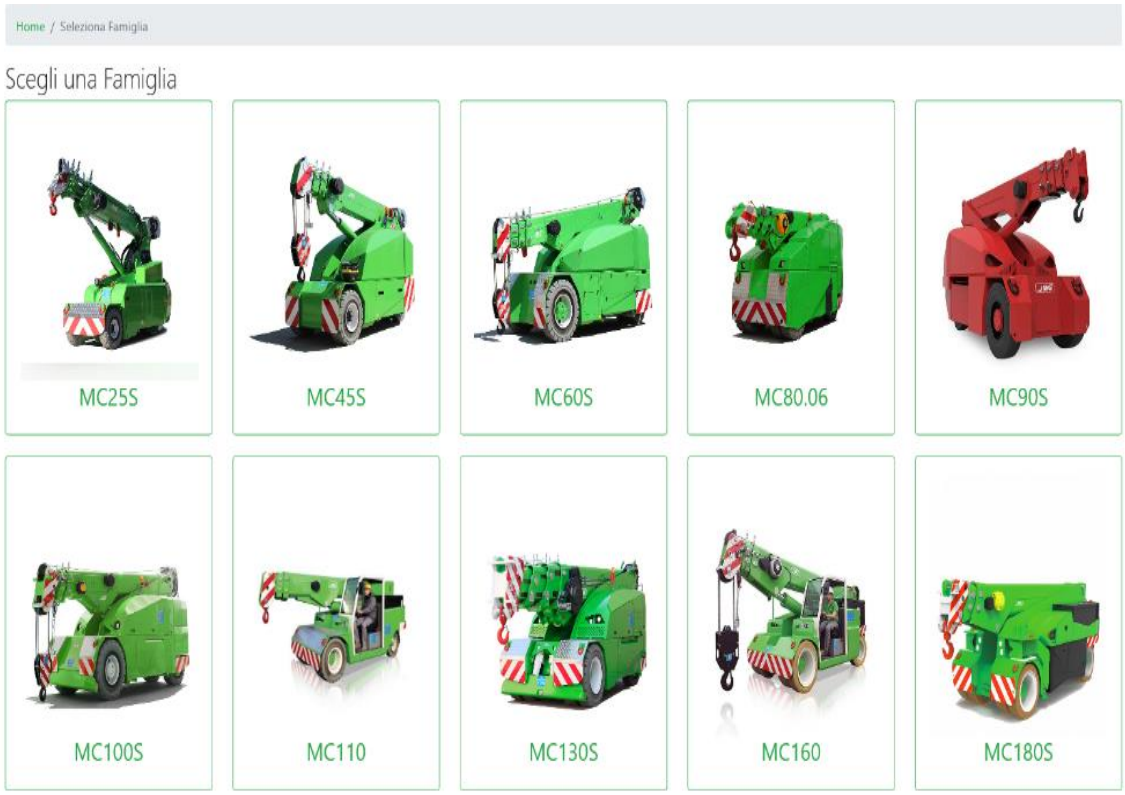

Figura 8: primo step per la configurazione della gru mobile da testare

La costruzione della gru mobile che si ha intenzione di andare a testare parte, anzitutto, con la scelta di una famiglia di gru, la quale darà poi spazio alla conseguente scelta di una delle macchine che vi fanno parte. Successivamente si passa alla selezione di una zavorra e di un accessorio, con cui equipaggiare la gru, tra quelli presenti nelle liste viste in precedenza. Infine si arriva ad una pagina, mostrata nella figura 9, in cui vanno inseriti i valori di peso del carico da sollevare, altezza a cui sollevarlo, area di lavoro e, se necessario, la posizione della testa dell'accessorio.

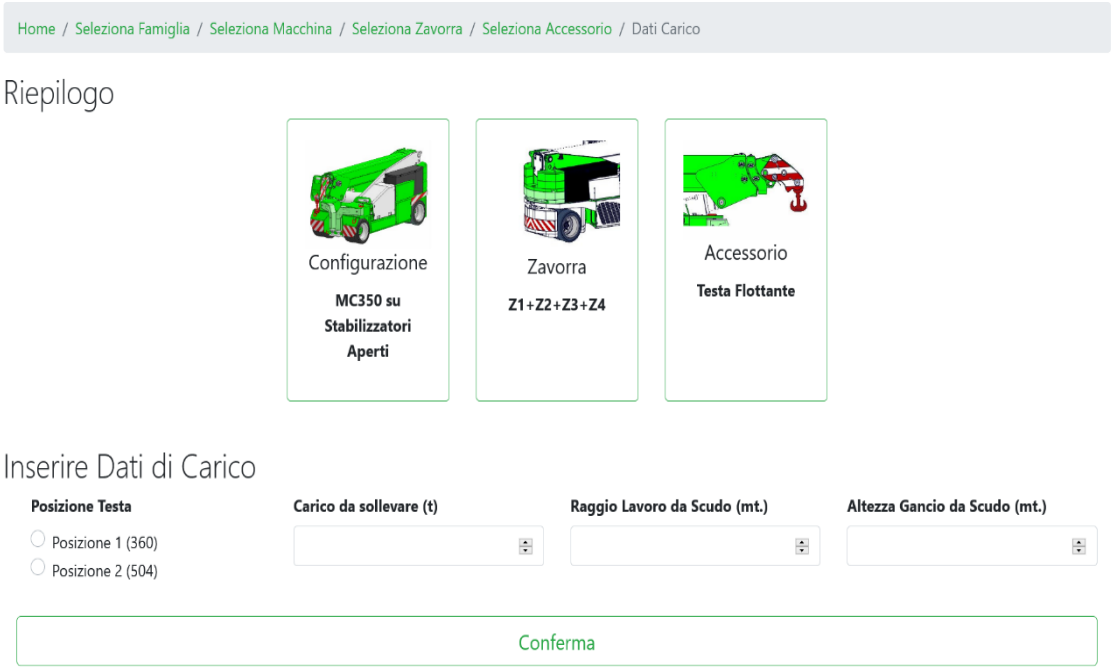

Figura 9: pagina per l'inserimento dei dati per il carico da sollevare

A questo punto, se i dati immessi sono alla portata della gru, l'ultima pagina (figura 10) della sequenza riporta i valori di carico al suolo a cui sono sottoposti gli pneumatici o gli stabilizzatori, la pressione delle gomme, ecc…. In alternativa viene mostrato un errore e si richiede all'utente di inserire nuovamente i valori.

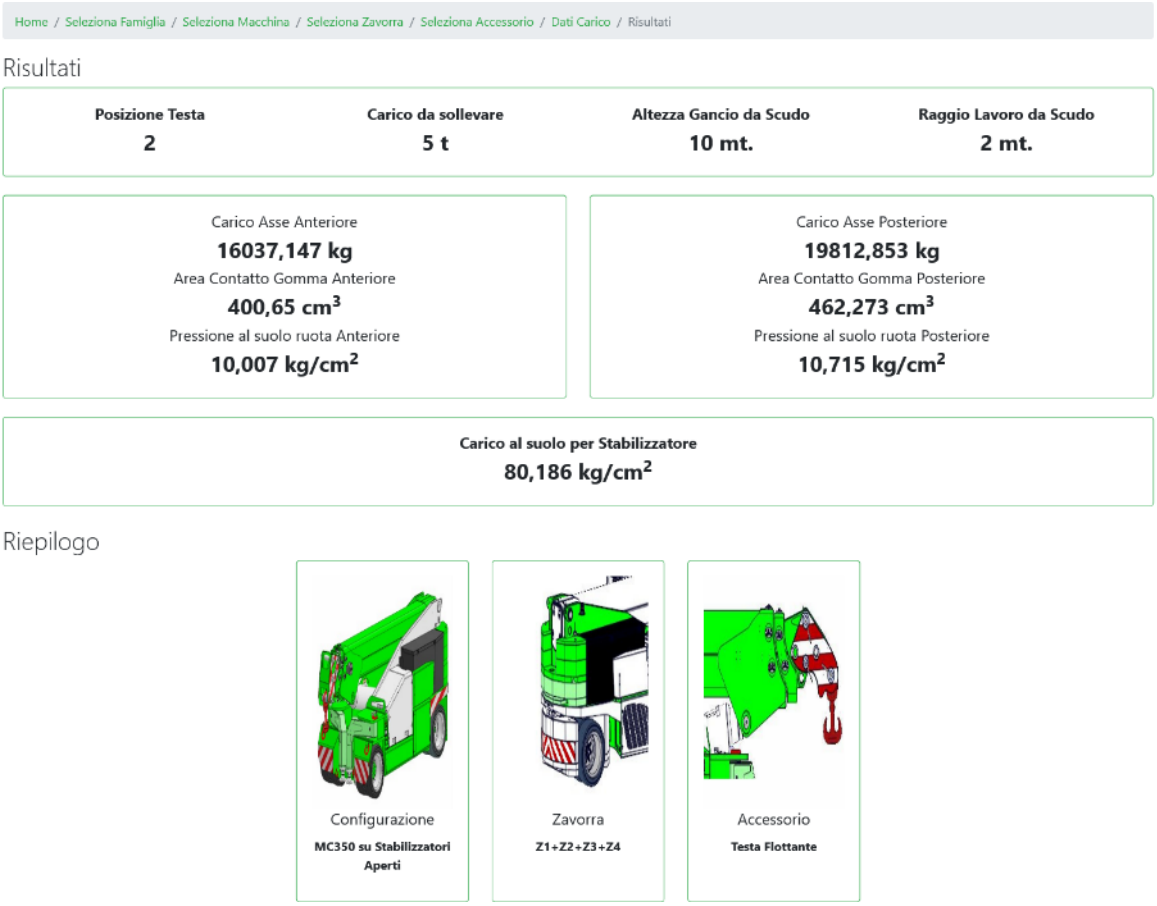

Figura 10: ultima pagina della sezione test che mostra i risultati relativi alla gru MC350S con gli stabilizzatori aperti

3. Sviluppo e realizzazione del progetto

3.1 Creazione delle pagine web con JSP

La parte centrale di ogni web-application è quella che nel pattern MVC viene detta view (o vista), ovvero colei che si occupa di fornire una visione chiara dei dati e dell'interazione tra l'utente e l'applicazione stessa, attività che in questo progetto avviene per mezzo di pagine web realizzate tramite delle JSP (Java Server Pages).

La scelta di utilizzare JSP anziché HTML, è stata fatta per avere delle pagine web rese dinamiche grazie all'uso di JSP Expression Language (EL) e ai JSP Standard Tag Library (JSTL), in questo modo è possibile evitare l'utilizzo di costrutti JavaScript e chiamate Ajax all'interno del codice, per quelle operazioni come l'iterazione di liste o l'implementazione di salti condizionali, attuabili tramite tag specifici che JSTL ed EL mettono a disposizione. Ne consegue un codice nel complesso molto più pulito e leggibile per gli sviluppatori che necessitano modificarlo in un secondo momento.

Infine, un'altra peculiarità di queste librerie è che permettono di creare pagine web multilingua in modo semplice e veloce, traducendo tutta la parte statica delle pagine, ovvero tutti gli elementi che non fanno parte del database come ad esempio i bottoni, i titoli, le descrizioni, ecc…, questo avviene per mezzo di dizionari, uno per ogni lingua che si intende supportare, contenenti tutte quelle parole o frasi nella forma chiave-valore, in questo modo le JSP, dopo aver verificato la lingua con cui si sta visualizzando il browser, andranno a tradurre automaticamente il testo della pagina web; il tutto a condizione che esista un dizionario per quella lingua, altrimenti di default la pagina viene visualizzata in lingua inglese.

3.2 *Controllo dell'applicazione tramite Servlet*

Il controllore, o controller, nel pattern MVC ritengo sia una delle parti più consistenti ed importanti dell'intero progetto, in quanto si occupa di svolgere diversi compiti, tra i principali vi sono: l'intercettazione delle richieste fatte dall'utente attraverso le pagine web, la gestione del modello per la creazione degli oggetti ottenuti dal database ed il coordinamento della vista fornendo la visualizzazione nel giusto ordine delle pagine web.

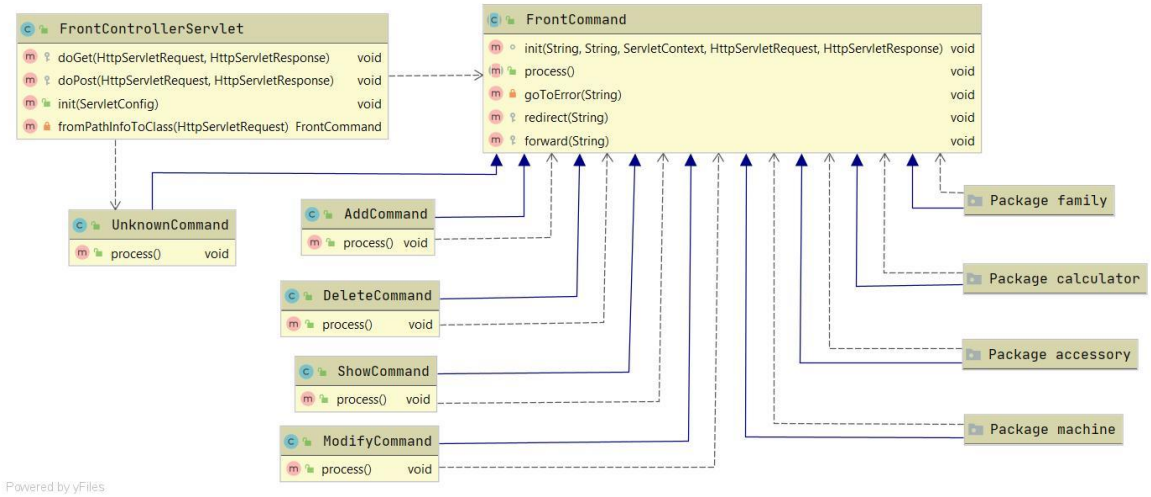

Figura 11: diagramma UML delle classi che costituiscono il controller dell'applicazione

La sua implementazione in questa web-app avviene per mezzo di *servlet* (facendo riferimento alla figura 11 sono rappresentate dalle classi che includono "Command" nel nome, ad eccezione di *FrontCommand*) che numericamente sono tante quante sono le operazioni che è possibile effettuare tramite le JSP, questo per favorire la manutenzione e le eventuali modifiche del codice in futuro (nella figura 11 sono visibili, a titolo esemplificativo, solamente le servlet relative alle operazioni sulle famiglie di gru, le servlet mancanti sono contenute nei package, o cartelle, che fanno riferimento all'oggetto controllato). Tale scelta richiede, dunque, il coordinamento di tutte queste servlet da parte di un'unica *Front Controller Servlet*, il cui compito è quello di intercettare tutte le *form*, ovvero le richieste, che vengono fatte dalle JSP tramite metodi *get* o *post* ed in base all'operazione che si vuole svolgere, ovvero il nome della servlet che si vuole chiamare (Add, Modify, Show, …), tramite il metodo *fromPathInfoToClass* che interpreta la richiesta, si costruisce il *path*, o il percorso, che porta verso la servlet adibita alla gestione della richiesta e gli passa il controllo.

Le servlet a loro volta possono svolgere operazioni più o meno complesse, queste spaziano dalla semplice invocazione della pagina web successiva tramite i metodi *redirect e forward*, ereditati dalla classe *FrontCommand*, con l'eventuale passaggio di dati richiesti al database, fino al dover eseguire frammenti di codice più complesso come, ad esempio, la gestione di una o più form e la creazione di eventuali nuovi oggetti annessi da dover successivamente aggiungere, rimuovere o modificare dal database.

Le richieste che vengono fatte all'interno delle JSP possono essere distinte in due tipologie, sulla base dei parametri che contengono:

1. Quelle che contengono solamente dati "semplici", quali numeri, stringhe, ecc…
2. Quelle in cui sono presenti delle immagini da associare all'oggetto che si vuole andare a costruire.

Di conseguenza si richiede, da parte delle servlet, una gestione ed un'elaborazione differente. Nel caso "più semplice", quello in cui non vi è un immagine, tutti i parametri all'interno della form possono essere recuperati semplicemente facendo uso del metodo *request.getParameter*, al cui interno si specifica il nome del tag contenente il valore che si vuole passare. Nell'altro caso, ovvero quello in cui si vi è anche un'immagine da associare ad un oggetto, la *form* necessita di una codifica dei dati che formano "il corpo della richiesta" (aggiunta al tag della *form* tramite la dicitura *enctype="multipart/form-data"* e che verrà analizzata più nel dettaglio nel capitolo 4.3*)* per essere effettuata. Questa codifica rende impossibile accedere ai parametri attraverso il metodo *request.getParameter*, *come nel caso precedente,* rendendo di conseguenza obbligatorio l'uso di librerie esterne che permettano di recuperare i parametri della richiesta La libreria che ho utilizzato per questa applicazione è l'*Apache file upload* che fa parte delle librerie di *Apache Commons*. Questa libreria permette di suddividere i parametri della *request*, in due gruppi:

a) Quelli contenenti dei valori.
b) Quelli contenenti dei file

I primi vengono messi in un *buffer* che è possibile sfogliare a piacimento per recuperarne i valori in esso contenuti mediante il nome del tag, come per le form classiche. I file, invece, che in questa applicazione sono sempre e solo delle immagini di gru, accessori o zavorre devono essere convertiti per poter essere utilizzati all'interno dell'applicazione.

Il formato che ho scelto di utilizzare per la conversione è il formato standard detto *BASE64*. La conversione in BASE64 trasforma un'immagine in una stringa di testo *ASCII* (*American Standard Code for Information Interchange*) che rappresenta il codice binario dell'immagine convertita. Tale conversione, per quanto risulti poco efficiente in termini di memoria occupata, risulta la più pratica da usare, perchè essendo un formato standard viene riconosciuta sia dalle classi Java sia dalle pagine JSP, rendendo possibile visualizzarne il contenuto direttamente dalle pagine web senza ulteriori riconversioni. Inoltre trattandosi di una stringa può essere tranquillamente salvato nel database sotto forma di testo, rappresentando cosi il codice di lettura da cui si può ricavare l'immagine.

3.3 Creazione del database per la persistenza dei dati

Per realizzare il modello dell'applicativo, ovvero quella parte del pattern MVC che gestisce e fornisce l'accesso ai dati, ho dovuto in primis costruire un database relazionale che contenesse tutti i dati necessari per l'utilizzo dell'applicazione e nel quale JMG può andare ad aggiungere, modificare o eliminare tutte le informazioni relative alle gru che costruisce.

Per offrire al suo utilizzatore una maggiore flessibilità nella gestione dei dati contenuti all'interno del database, ho provveduto alla creazione di 3 diversi gruppi di tabelle suddivisi nel seguente modo:

1. Tre tabelle di "anagrafica" per gli oggetti più semplici: quali famiglie di gru, accessori e zavorre.
2. Una tabella per la rappresentazione delle gru mobili nel dettaglio.
3. Due tabelle aggiuntive: una per gli accessori e una per le zavorre, con i relativi valori di peso, lunghezza, dimensione, … con le quali è possibile definire gli elenchi degli optional da equipaggiare ad una gru.

Le tabelle di anagrafica (nella figura 12 nominate *family*, *accessory* e *ballast*) sono usate principalmente come "inventario", sono quindi formate da un'id (o identificativo), per garantire un riferimento univoco alla "*tupla"* (riga del DB), un nome e solamente per le famiglie un'immagine.

Il nome di tutti gli oggetti presenti nel database, in previsione di un utilizzo internazionale dell'applicazione, è stato pensato in modo tale che potesse contenere diversi valori, a seconda della lingua che si intende usare. Ciò è stato possibile grazie all'utilizzo dei *JSON* (*JavaScript Object Notation*), oggetti particolari, salvati all'interno della tabella nel database sottoforma di stringa, che permettono la creazione di collezioni di coppie chiave-valore dove: la chiave rappresenta il codice della lingua (ad esempio: *en* per l'inglese oppure *it* per l'italiano); mentre il valore, associato ad una chiave, contiene la traduzione del nome nella lingua specificata dal codice. L'uso dei *JSON* permette, oltre che una semplificazione delle tabelle del database, in quanto vengono salvati al loro interno come stringhe mantenendo comunque alterati la struttura e le proprietà una volta estratti ed utilizzati all'interno delle classi Java, anche la possibilità di aggiungere, in qualunque momento, altre traduzioni per le lingue di un nuovo mercato agevolando così l'internazionalizzazione dell'applicativo.

La tabella delle gru mobili è usata per rappresentare nel dettaglio i mezzi prodotti da JMG. Come per le tabelle di anagrafica, anche questa tabella ha le colonne con gli id, i nomi e le immagini delle macchine, ma a differenza delle precedenti, sono state aggiunte altre colonne usate per contenere tutti i valori dei dati tecnici delle gru che si sono stati descritti nel capito 2.1 e mostrati nella figura 12 dalla tabella nominata *machine.*

Le ultime due tabelle, una per gli accessori e una per le zavorre (nominate *machine_accessory* e *machine_ballast* nella figura 12), servono per associare alle gru gli equipaggiamenti che questa può montare. Per farlo si utilizzano gli *id*, in questo modo è possibile attribuire alla singola macchina (ovvero la gru mobile) uno o più accessori e/o zavorre, così da avere una sorta di lista tra cui scegliere i componenti che è possibile equipaggiare. In aggiunta ad ogni associazione, tra gru e accessorio e/o zavorra, vengono riportati anche tutti dati di peso, dimensioni, ecc… di questi ultimi. In questo modo, una volta che sono stati selezionati i componenti si ha una visione completa della gru mobile.

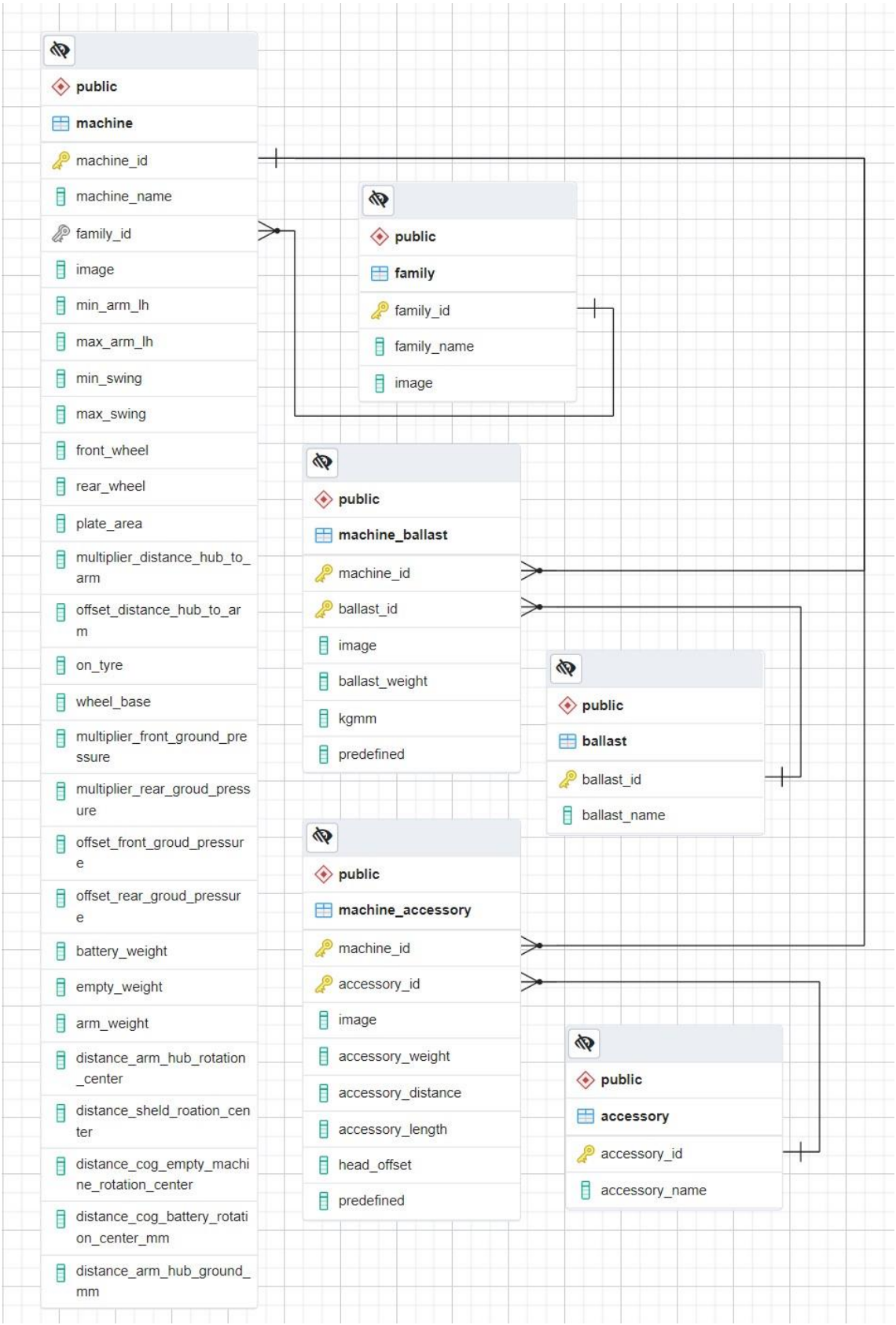

Figura 12: schema ER del database

3.4 Trasposizione del database in oggetti Java

La realizzazione del modello per l'applicazione prosegue poi con la trasposizione in classi oggetto Java delle famiglie, delle gru mobili, degli accessori e delle zavorre presenti all'interno del database. I dati, infatti, definiti come tabelle relazionali in un database per poter essere manipolati dalle servlet e visualizzati nelle pagine web, necessitano di una loro definizione come classe oggetto Java.

Le classi Java per le famiglie, le gru mobili, gli accessori e le zavorre (rispettivamente Family, *Machine, Accessory* e *Ballast* nella figura 13) servono per la creazione di oggetti temporanei che permettono la loro manipolazione da parte delle *servlet* e il loro successivo passaggio alle pagine web al fine di essere visualizzati dall'utente.

La classe *Translator* viene usata dalle altre classi oggetto per convertire i JSON, ottenuti dal database e contenenti le varie traduzioni dei nomi, in *HashMap*, ovvero degli oggetti nativamente supportati da Java e dalle JSP il cui scopo principale è quello di rendere veloci ed efficienti operazioni quali inserimento e ricerca di elementi. La conversione in HashMap viene fatta in quanto le pagine JSP, grazie all'uso combinato delle JSTL e delle EL come si vedrà nel capitolo 4.2, sono in grado trovare e selezionare il giusto valore della lingua che si desidera usare, così da rendere l'applicativo flessibile e multilingua allo stesso momento.

In fine la classe *Calculator* è stata creata per effettuare i calcoli relativi al test delle gru mobili durante la fase di sollevamento, come si era accennato all'inizio di questa tesi. La sua realizzazione è avvenuta scrivendo dei metodi che sulla basa delle formule che ci sono state fornite da JMG calcolano tutti i valori di carico sugli assi, condizioni di ribaltamento, ecc… necessari per stabilire se una gru sia in grado o meno di sollevare il peso selezionato all'altezza desiderata.

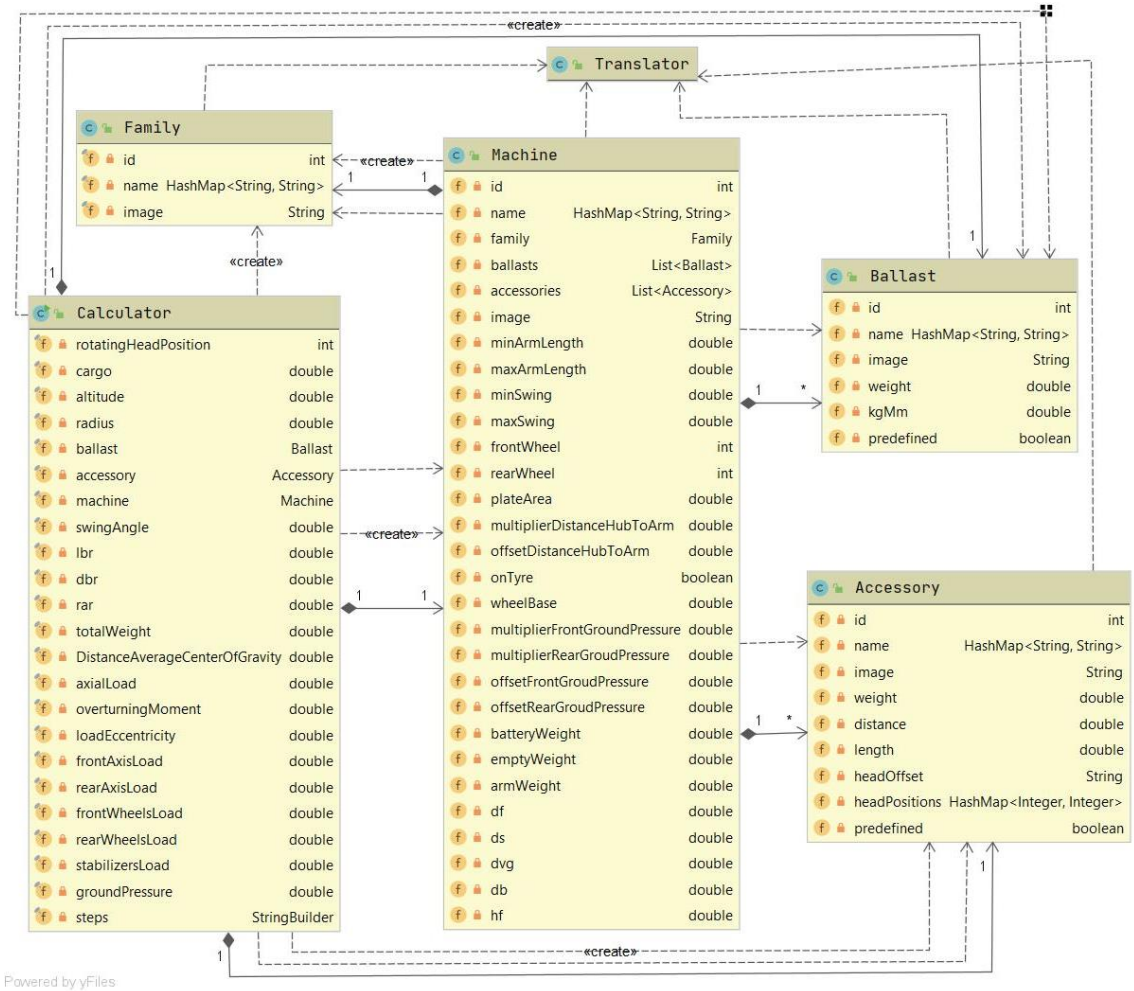

Figura 13: schema UML delle classi oggetto che compongono il modello dell'applicazione

3.5 Comunicazione con il database: DAO

L'ultimo oggetto necessario per la completa realizzazione del modello per il progetto è quello che si occupa dell'interazione con il database e della creazione degli oggetti Java, discussi pocanzi nel capitolo 3.4.

La classe DAO (*Data Access Object*), in questa applicazione non è altro che un enorme contenitore al cui interno sono stati scritti tutti quei metodi utili per fare richieste o apportare modifiche al database. La scelta implementativa adottata per questa classe, seppur poco in linea con le idee di separazione e suddivisione dei compiti viste fino ad

ora, è stata determinata dalla mancanza di tempo, alla fine progetto, per la pulizia ed l'ottimizzazione di alcune classi, tra cui quest'ultima. All'interno di questa classe trovano posto molteplici possibilità di interrogazione (ovvero le query) al database, sottoforma di metodi java, che mi hanno permesso di poter produrre realizzare l'applicazione in oggetto.

I metodi che ho scritto (si riporta qualche esempio nella figura14) non fanno altro che completare ed eseguire le query di selezione, inserimento, cancellazione o aggiornamento, che sono state già in parte scritte grazie all'uso di oggetti chiamati, *PreparedStatement.* Questi ultimi permettono, appunto, di "preparare" le query, scrivendo l'operazione che si vuole eseguire, ma sostituendo i valori dei parametri con dei punti interrogativi (?) che fanno da segnaposto per i valori che si vorranno effettivamente richiedere.

Le query verranno poi completate sostituendo i punti interrogativi con i valori effettivi richiesti dall'utente presi dai parametri in input del metodo. Completata la query i metodi provano ad eseguirla nel database per verificarne l'integrità della transazione, al fine di evitare eventuali situazioni di inconsistenza dei dati nel database come ad esempio nel caso dell'eliminazione di una famiglia di gru con ancora associate delle configurazioni, e se la transazione non presenta errori le modifiche vengono rese persistenti, altrimenti viene generata un'eccezione che rende noto all'utente che sta provando ad effettuare l'operazione il problema.

```java
public static Family getFamilyById(Connection conn, int id){
    try (PreparedStatement ps =
     conn.prepareStatement( sql: "SELECT * FROM family WHERE family_id = ? ORDER BY family_name ASC")){
            ps.setInt( parameterIndex: 1, id);
            ResultSet rs = ps.executeQuery();
            rs.next();
            return new Family(rs);
    }catch(SQLException e){
        System.out.println("********************ERORE DAO.getFamilyById*****************" + e.getMessage());
    }
    return null;
}

public static State insertFamily(Connection conn, Family family){
    State result = State.NOTOK;
    try (PreparedStatement ps =
     conn.prepareStatement( sql: "INSERT INTO family (family_name, image) VALUES (?, ?)")){
                ps.setString( parameterIndex: 1, Translator.toJson(family.getName()).toString());
                ps.setString( parameterIndex: 2, family.getImage());
                result = ps.executeUpdate()==1 ? State.OK : State.ERROR;
    }catch(SQLException e){
        System.out.println("********************ERORE DAO.insertFamily*****************" + e.getMessage());
        result = State.ERRDB;
    }
    return result;
}

public static State deleteFamily(Connection conn, Family family){
    State result = State.NOTOK;
    try (PreparedStatement ps =
     conn.prepareStatement( sql: "DELETE FROM family WHERE family_id = ?")){
                ps.setInt( parameterIndex: 1, family.getId());
                result = ps.executeUpdate()==1 ? State.OK : State.ERROR;
    }catch(SQLException e){
        System.out.println("********************ERORE DAO.deleteFamily*****************" + e.getMessage());
        result = State.ERRDB;
    }
    return result;
}
```

Figura 14: codice relativo alle query di selezione, inserimento e cancellazione delle famiglie di gru

4. Scelte tecnologiche ed implementative

4.1 Il Front controller pattern

Il *Front Controller*, o anche detto *Dispatcher,* rappresenta la parte più importante di un sistema *MVC,* tanto da essere associato ad un design pattern a sé stante.

Il *Front Controller* viene definito come "un *controller* che cattura tutte le richieste di un sito web". Si posiziona di fronte ad una Web-Application e delega le richieste alle risorse successive, provvede inoltre a fornire un'interfaccia per comportamenti comuni come: la sicurezza, l'internazionalizzazione e la presentazione di viste particolari a determinati utenti. L'implementazione del *Front Controller* permette all'applicazione di modificare il proprio comportamento in fase di esecuzione. Inoltre aiuta a leggere e mantenere un'applicazione impedendo la duplicazione del codice.

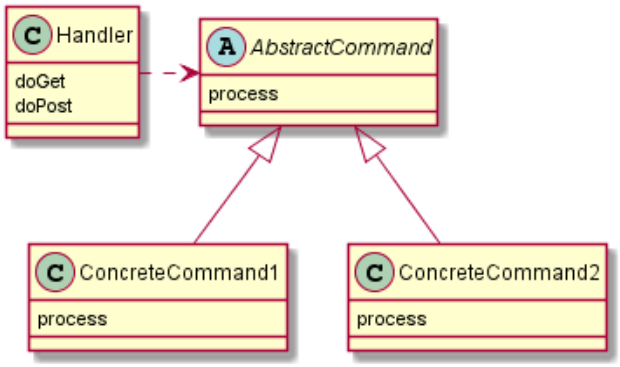

Figura 15: diagramma UML per le classi che compongono il front-controller pattern

Tale pattern è suddiviso in due parti: un singolo *dispatching controller* (nella figura 15 la classe denominata *Handler)* ed una gerarchia di comandi (le classi *AbstracCommand* e *ConcreteCommand* della figura 15). La relazione tra le classi di una generica implementazione del front controller è mostrata nel diagramma UML della figura 14.

Il compito del Front Controller è quello di intercettare, tramite il *dispatching controller* tutte le richieste che vengono fatte al sistema, farle elaborare al giusto *controller* ed infine reindirizzare il flusso sulla visualizzazione corretta, detta anche *view*.

Un'organizzazione del flusso in questa forma presenta, inoltre, diversi vantaggi tra cui:

- componenti che svolgono compiti semplici e di conseguenza sono più facili da realizzare, modificare e testare

- una suddivisione dello sviluppo in strati, i quali possono essere sostituiti in qualunque momento da nuove implementazioni, inoltre garantisce che non vi siano interferenze e che ogni componente svolga solamente il compito che gli spetta
- sistema facilmente estendibile, ovvero permette di realizzare più funzionalità con minore sforzo

4.2 JavaServer Page Standard Tag Library ed Expression Language

La *JSTL* (JavaServer Page Standard Tag Library) è una libreria inclusa come componente della piattaforma software per applicazioni web *Java EE (Enterprise Edition)*. JSTL estende le specifiche delle *JSP* incorporando un insieme di tag alla libreria di JSP per attività comuni quali:

- La manipolazione di documenti XML e processamento di dati XML.
- Esecuzioni condizionali e cicli loop, usate nel progetto per la visualizzazione di un'immagine di repertorio in caso di oggetto senza immagine e l'iterazione delle liste di oggetti da visualizzare nelle pagine web.
- Accesso a database e tag SQL.
- L'internazionalizzazione: con cui l'applicazione è in grado di riconoscere la lingua che si sta utilizzando per tradurre il contenuto delle pagine in quella lingua.
- L'integrazione dei tag personalizzati esistenti con i tag JSTL

Il vantaggio nell'uso delle JSTL è quello di offrire un modo efficiente di incorporare costrutti Java eliminando la necessità di definire degli *scriptlet* (*applet* fatte con un linguaggio di *scripting*) all'interno delle pagine JSP, ottenendo anche l'ulteriore vantaggio di distinguere il compito dello sviluppatore Java da quello del web designer.

Le EL (Expression Language) sono istruzioni aggiunte dalla versione 2.0 alle specifiche per le pagine JSP. Mediante l'uso di queste istruzioni è possibile per gli autori di pagine utilizzare espressioni semplici per accedere dinamicamente ai dati dei componenti JavaBeans (ovvero le classi oggetto Java usate per la rappresentazione delle gru mobili e dei loro componenti), e dunque di arrivare allo stesso risultato che otterrebbe attraverso elementi di *scripting* base, con la differenza che la sintassi delle EL è più semplice.

Riassumendo EL fornisce un modo per utilizzare semplici espressioni con la quale è possibile eseguire le seguenti attività:

- Leggere dinamicamente i dati dell'applicazione memorizzati nei componenti JavaBeans, in varie strutture dati e negli oggetti impliciti.
- Scrivere dinamicamente dati come input dell'utente nei componenti JavaBeans.
- Richiamare arbitrariamente metodi statici e dinamici.
- Eseguire dinamicamente operazioni aritmetiche.

Un altro possibile utilizzo delle EL è quello delle cosiddette *scriptless pages:* una delle direttive di progettazione che è stata richiesta dal mio tutor aziendale. Le *scriptless pages* sono essenzialmente delle pagine JSP in cui non è consentito inserire nessuno degli scripting elements, ovvero frammenti di codice incorporato all'interno delle pagine e strutturato come segue: *<% codice di scripting %>*, al fine di separare la parte di view (l'interfaccia utente) da quella contenente la logica di business (il modello dell'applicazione). In queste pagine, dunque, il comportamento dinamico dovrà essere fornito da altri elementi come i JavaBeans (le classi oggetto Java), e le standard tag libraries; a cui andranno ad aggiungersi anche le qui citate espressioni EL.

Le Expression Languages sono, inoltre, usate per specificare i seguenti tipi di espressioni che un attributo di tag personalizzato accetterà:

- *Espressioni di valutazione immediata* o *differita;* le prime vengono valutate immediatamente dalla tecnologia sottostante (come ad esempio le JavaServer Faces), le altre possono essere valutate successivamente utilizzando l'EL.
- *Espressioni di valore* che fanno riferimento ai dati, o *espressioni di metodo* le quali invocano un metodo.
- *Espressioni rvalue o lvalue.* Una rvalue expression può solamente leggere un valore, mentre le lvalue expression possono sia leggere un valore che scriverlo su un oggetto esterno.

La sintassi delle espressioni EL è la seguente: *${espressione}* dove *espressione* corrisponde, appunto, ad un'espressione valida, ovvero che includa al suo interno uno o più dei seguenti elementi: letterali, operatori, variabili e/o invocazioni a metodi. Come esempio si riporta un frammento di codice di una delle pagine del progetto che mostra l'utilizzo combinato dei JSTL ed EL per la realizzazione della pagina che mostra le

zavorre da selezionare per la configurazione di una gru da testare. Nella figura 16 si può vedere come tramite le JSTL sia possibile iterare, per mezzo del ciclo *forEach,* la lista delle zavorre che la gru può equipaggiare a cui è possibile accedere grazie all'espressione EL *${requestScope.machine.ballasts}.* Inoltre si va a verificare se la zavorra ha un'immagine associata per farla visualizzare, altrimenti ne viene fatta vedere una di default come segnaposto; ed infine è anche possibile accedere al nome nella lingua di visualizzazione della pagina sempre che sia stato definito.

```
<c:forEach items="${requestScope.machine.ballasts}" var="ballast">
    <div class="col mb-4">
        <div class="card h-100 border-success">
            <div class="card-body">
                <a class="text-success" href="<c:url value="/web/calculator/accessory">
                    <c:param name="ballast" value="${ballast.id}"/>
                    <c:param name="machine" value="${requestScope.machine.id}"/></c:url>">
                    <c:choose>
                        <c:when test="${empty ballast.image}">
                            <img src="<c:url value="/media/Image.png"/>" class="card-img"
                                 alt="<c:out value="${ballast.name.get(lang.toString())}"/>"
                                 height="300" width="300">
                        </c:when>
                        <c:otherwise>
                            <img src="<c:url value="${ballast.image}"/>"
                                 class="card-img"
                                 alt="<c:out value="${ballast.name.get(lang.toString())}"/>"
                                 height="300" width="300">
                        </c:otherwise>
                    </c:choose>
                    <h3 class="text-center text-break">
                        <c:choose>
                            <c:when test="${empty ballast.name.get(lang.toString())}">
                                <c:out value="${ballast.name.get('en')}"/>
                            </c:when>
                            <c:otherwise>
                                <c:out value="${ballast.name.get(lang.toString())}"/>
                            </c:otherwise>
                        </c:choose>
                    </h3>
                </a>
            </div>
        </div>
    </div>
</c:forEach>
```

Figura 16: codice della pagina page_select_ballast *presa del progetto realizzato*

4.3 Upload delle immagini tramite Apache file upload

L`Apache File Upload è un package facente parte di *Apache Commons*, un progetto della *Apache Software Foundation* il cui scopo è quello di fornire software Java riutilizzabile e open source. L'utilizzo di *File Upload* permette di integrare facilmente, nelle proprie servlet e web app, la possibilità di caricare un file nell'applicazione, da parte di un utente, che sia consistente, per assicurarne l'utilizzo in modo corretto, e performante.

File Upload può essere utilizzato in diversi modi a seconda dei requisiti dell'applicazione, nel caso più semplice si può richiamare un singolo metodo per analizzare una richiesta fatta alla servlet e quindi elaborare un elenco di elementi per usarli a proprio piacimento, in alternativa si può decidere di personalizzare FileUpload per avere il pieno controllo del modo in cui vengono archiviati i singoli elementi, ad esempio decidendo di trasmettere il contenuto in un database.

Una richiesta di *caricamento file* comprende un elenco ordinato di elementi codificati secondo *RFC 1867*, "*Caricamento di file basato form in HTML*"; cioè, se una richiesta HTTP viene inviata utilizzando il metodo POST e con un tipo di codifica "multipart / form-data", come detto nel capitolo 3.2, FileUpload può analizzare quella richiesta e rendere i risultati disponibili in un modo facilmente utilizzabile dal chiamante. *File Upload* (come sopra) analizza tale richiesta e fornisce all'applicazione un elenco dei singoli elementi caricati, ciascuno dei quali implementa l'interfaccia *FileItem*, indipendentemente dalla sua implementazione sottostante.

Ogni elemento del file ha una serie di proprietà per soddisfare ogni tipo di applicazione: ad esempio, ogni elemento ha un nome e un tipo di contenuto e può fornire un *InputStream* per accedere ai suoi dati. D'altra parte, potrebbe essere necessario elaborare gli elementi in modo diverso, a seconda che l'elemento sia un regolare campo di una *form*, ovvero i cui dati provengono da una normale casella di testo o da un campo HTML o simile; oppure da un file caricato. L'interfaccia *FileItem* fornisce i metodi per effettuare tale determinazione e per accedere ai dati nel modo più appropriato. *FileUpload*, inoltre, crea dei nuovi elementi di tipo file utilizzando *FileItemFactory*. L'implementazione di fabbrica attualmente fornita con FileUpload archivia i dati dell'elemento in memoria o su disco, a seconda delle dimensioni dell'elemento (cioè byte

di dati). Tuttavia, questo comportamento può essere personalizzato per adattarsi all'applicazione.

4.4 La codifica in Base64 delle immagini

Base64 è un insieme di schemi di codifica *binary-to-text* che permette la traduzione di dati binari in stringhe, nello specifico una sequenza di 8-bit in forma di stringa *ASCII* traducendoli in una rappresentazione *radix*-64. Il termine Base64 ha origine da una specifica codifica di trasferimento detta MINE. Ogni cifra Base64 non finale rappresenta esattamente 6-bit; perciò tre byte (per un totale di 24-bit) possono essere rappresentati da 4 cifre Base64 a 6-bit

Base64, comunemente a tutti gli altri schemi di codifica da binario a testo, è progettato per trasportare i dati archiviati in formati binari attraverso canali che supportano in modo affidabile solo contenuto di testo. Tale caratteristica lo rende particolarmente diffuso sul World Wide Web dove i suoi usi includono la possibilità di incorporare file di immagine o altri asset binari all'interno di asset testuali come file HTML e CSS.

Tra le altre cose, Base64 è anche ampiamente utilizzato per l'invio di allegati di posta elettronica, ciò è necessario perché SMTP nella sua forma originale è stato progettato per trasportare solo caratteri ASCII a 7 bit. Questa codifica in Base64 causa un overhead (un aumento di memoria occupata) del 33-36% (33% dalla codifica stessa, e fino al 3% in più dalle interruzioni di riga inserite) dovuta dal fatto che ogni gruppo di 3 byte viene convertito in 4 caratteri. Questo supponendo che per rappresentare un carattere si utilizzi un intero byte.

Il particolare insieme dei caratteri scelto per rappresentare i 64 valori delle cifre varia da implementazione a implementazione. La strategia generale consiste nello scegliere 64 caratteri comuni alla maggior parte delle codifiche e che sono anche stampabili. Questa combinazione rende improbabile che i dati vengano modificati durante il trasferimento attraverso sistemi di informazione, come la posta elettronica, che tradizionalmente non erano a *8 bit puliti*, ovvero non gestivano correttamente le codifiche di tali caratteri (contrariamente allo standard ISO 8859 e UTF-8 di Unicode). Le altre varianti, solitamente derivate dal Base64, condividono queste proprietà ma differiscono nella

scelta degli ultimi due caratteri e per il "*padding char"*: in particolare, molto usate sono le varianti *URL* e *file name safe* (RFC 4648/Base64URL), le quali usano i caratteri "-" e "_" come valori per i caratteri 62 e 63 al posto di "+" e "/" e non utilizzano il padding.

Per fare un esempio, l'implementazione Base64 di MIME utilizza i caratteri A – Z, a – z e le cifre 0 – 9 per i primi 62 valori; altre varianti condividono questa proprietà ma differiscono nei simboli scelti per gli ultimi due valori; un esempio è UTF-7.

Le prime istanze di questo tipo di codifica sono state create per la comunicazione dial-up tra sistemi che eseguono lo stesso sistema operativo - ad esempio, uuencode per UNIX (Unix to Unix encode), BinHex per TRS-80 (successivamente adattato per Macintosh) - e potrebbero quindi fare più ipotesi su quali caratteri sono più sicuri da usare. Ad esempio, uuencode utilizza lettere maiuscole, cifre e molti caratteri di punteggiatura, ma non minuscole.

A titolo esemplificativo si riporta una tabella di traduzione degli indici in Base64:

Index	Binary	Char	Index	Binary	Char	Index	Binary	Char	Index	Binary	Char
0	000000	A	16	010000	Q	32	100000	g	48	110000	w
1	000001	B	17	010001	R	33	100001	h	49	110001	x
2	000010	C	18	010010	S	34	100010	i	50	110010	y
3	000011	D	19	010011	T	35	100011	j	51	110011	z
4	000100	E	20	010100	U	36	100100	k	52	110100	0
5	000101	F	21	010101	V	37	100101	l	53	110101	1
6	000110	G	22	010110	W	38	100110	m	54	110110	2
7	000111	H	23	010111	X	39	100111	n	55	110111	3
8	001000	I	24	011000	Y	40	101000	o	56	111000	4
9	001001	J	25	011001	Z	41	101001	p	57	111001	5
10	001010	K	26	011010	a	42	101010	q	58	111010	6
11	001011	L	27	011011	b	43	101011	r	59	111011	7
12	001100	M	28	011100	c	44	101100	s	60	111100	8
13	001101	N	29	011101	d	45	101101	t	61	111101	9
14	001110	O	30	011110	e	46	101110	u	62	111110	+
15	001111	P	31	011111	f	47	101111	v	63	111111	/

Figura 17:tabella di codifica dei caratteri in binario

4.5 La creazione di nomi multilingua con i JSON

Il JavaScript Object Notation (JSON) è un formato per lo scambio di dati semplice e leggero. La sua peculiarità è che risulta al contempo sia facile da leggere e scrivere per gli umani, sia facile da analizzare e generare per i compilatori. Si basa su un sottoinsieme dello standard del linguaggio di programmazione JavaScript MCMA-262 nella 3ª edizione (dicembre 1999); per quanto JSON sia un formato di testo completamente indipendente, utilizza comunque convenzioni famigliari ai programmatori della famiglia di linguaggi C, Java, Perl, Python e molti altri. Tutte queste proprietà rendono JSON un linguaggio ideale per lo scambio di dati.

La sua semplicità ha favorito l'utilizzo specialmente nella programmazione in AJAX (Asynchronous JavaScript and XML), questo perché, se usato tramite JavaScript, l'interprete è in grado di eseguirne il parsing tramite la funzione JSON.parse()

I JSON si basano su due strutture diverse:

- Una collezione di coppie nome-valore (*JSON Object*), realizzato quindi come un oggetto, una struttura, un record, una tabella di hash, ecc…
- Un elenco di valori (*JSON Array*), ovvero realizzato come un array, un vettore, una sequenza, un elenco, …

Queste sono strutture dati universali, di conseguenza praticamente tutti i linguaggi di programmazione li supportano in una forma o nell'altra; è logico quindi che anche un formato di dati interscambiabile con i linguaggi di programmazione si basi su queste strutture.

Un JSON Object (figura 18) è un insieme non ordinato di coppie chiave-valore, che rispetta la seguente sintassi: inizia con una parentesi graffa aperta ("{") e termina con una chiusa ("}"); ogni chiave è seguita dai due punti (":"); infine ogni coppia chiave-valore e separata dalla successiva per mezzo di una virgola (","). Inoltre, come si può vedere nelle figure 18, 19 e 20, un JSON Object può contenere al suo interno un altro *JSON Object* oppure un *JSON Array* andando a creare così una struttura più complessa. Come esempio si riporta di seguito uno dei JSON utilizzati nel progetto per internazionalizzare i nomi di tutti gli oggetti (famiglie, gru, accessori e zavorre) all'interno dell'applicazione:

{"labels": [{

"code": "it",

"value": "MC350 su Stabilizzatori Aperti"},

{"code": "en",

"value": "MC350 on open Stabilizers"}]}"

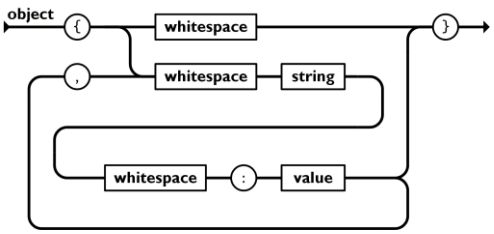

Figura 18: costruzione di un Json object

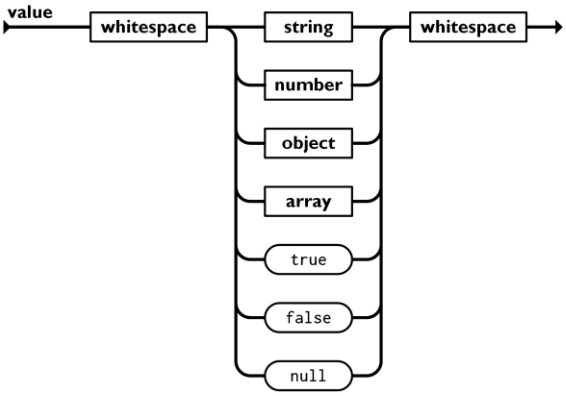

Figura 19: valori che possono assumere gli elementi all'interno di un Json

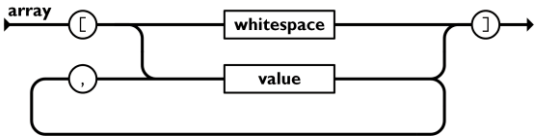

Figura 20: costruzione di un Json array

4.6 Interrogazione del database tramite Prepared Statement

Un *PreparedStatemet* (o istruzione preparata) è un oggetto in cui vengono archiviati gli statement (o istruzione) SQL precompilata. Un oggetto PreparedStatement può essere quindi utilizzato per eseguire in modo efficiente gli statement SQL, in esso contenuti, più volte. L'istruzione preparata assume la forma di un modello in cui vengono sostituiti determinati valori costanti durante ogni esecuzione.

Il flusso di esecuzione tipico dell'utilizzo di un prepared statement è il seguente:

1. Preparazione: viene creato il modello dell'istruzione e lo invia al DBMS (Database management system). Alcuni valori vengono lasciati non specificati, chiamati parametri, segnaposto o variabili di associazione etichettati con "?". Si riporta di seguito un esempio usato nel progetto: ***UPDATE family SET family_name=? WHERE family_id=?***

2 Il DBMS compila (ovvero analizza, ottimizza e traduce) il modello dell'istruzione e memorizza il risultato senza eseguirlo.

3 Esecuzione: in un secondo momento, vengono forniti (o associati) i valori per i parametri del modello di istruzione e il DBMS esegue l'istruzione (possibilmente restituendo un risultato).

L'applicazione può eseguire l'istruzione salvata tutte le volte che vuole con valori diversi (nell'esempio precedente si potrebbe inizialmente fornire come nome "famiglia 1" e come id "123"; successivamente si fare la stessa query ma con i valori "famiglia 2" e id "456")

Paragonata all'esecuzione diretta delle istruzioni, le Prepared Statement offrono due vantaggi principali:

- Il sovraccarico della compilazione dell'istruzione viene sostenuto solo una volta, sebbene l'istruzione venga eseguita più volte.
- Le Prepared Statement sono resilienti contro SQL injection perché i valori che vengono trasmessi in seguito utilizzando un protocollo diverso non vengono compilati come il modello di istruzione

SQL Injection è una tecnica di *code injection* (o iniezione di codice, è lo sfruttamento di un bug del computer causato dall'elaborazione di dati non validi.) utilizzata per attaccare le applicazioni che gestiscono dati attraverso database relazionali sfruttando il linguaggio SQL. Tale tipologia di attacco sfrutta il mancato controllo dell'input dell'utente, oppure la mancata tipizzazione forte delle variabili impiegate, per inserire artificiosamente delle stringhe di codice SQL, che saranno eseguite dall'applicazione server, con la quale e possibile far eseguire comandi SQL anche molto complessi, dall'alterazione dei dati al download completo dei contenuti nel database

5. Sitografia e Riferimenti

5.1 Front controller pattern

- https://www.baeldung.com/java-front-controller-pattern
- https://www.artera.net/it/blog/programmazione/implementare-il-design-pattern-mvc-in-java-con-le-servlet/

5.2 JSTL

- https://en.wikipedia.org/wiki/Jakarta_Standard_Tag_Library
- https://it.wikipedia.org/wiki/JSTL
- https://www.tutorialspoint.com/jsp/jsp_standard_tag_library.htm#:~:text=The%20JavaServer%20Pages%20Standard%20Tag,internationalization%20tags%2C%20and%20SQL%20tags.

5.3 EL

- https://www.html.it/articoli/jsp-expression-language/
- https://docs.oracle.com/javaee/6/tutorial/doc/bnahq.html

5.4 Apache file upload

- https://commons.apache.org/proper/commons-fileupload/
- https://commons.apache.org/proper/commons-fileupload/using.html

5.5 BASE64

- https://it.wikipedia.org/wiki/Base64
- https://en.wikipedia.org/wiki/Base64

5.6 JSON

- https://www.json.org/json-en.html

5.7 Prepared Statement

- https://en.wikipedia.org/wiki/Prepared_statement
- https://docs.oracle.com/javase/7/docs/api/java/sql/PreparedStatement.html