UNIVERSITÀ DI PADOVA **FACOLTÀ DI INGEGNERIA**

TESI DI LAUREA

PROGETTO E SVILUPPO DI UNA PIATTAFORMA PER LA GESTIONE IN REMOTO DELL'HERBARIUM PATAVINUM

Laureando: Carlo Menegazzo

Relatore: Sergio Canazza Targon

Corso di Laurea Triennale in Ingegneria dell'Informazione

Data Laurea 26/09/2013 Anno Accademico 2012/2013

Sommario

Questo elaborato descrive la progettazione e lo sviluppo del software per la gestione dell'Herbarium Patavinum, commissionato dalla dott.ssa Antonella Miola per conto del Dipartimento di Biologia dell'Università degli Studi di Padova. Saranno messi in evidenza gli strumenti utilizzati e l'integrazione dei differenti framework come Spring e Hibernate per sfruttare le potenzialità del design pattern MVC. Il sistema é quindi una piattaforma web che permette la ricerca, l'inserimento, la modifica e l'eliminazione delle componenti che descrivono il singolo foglio di catalogazione dell'erbario.

La piattaforma, presentata il 20 Settembre 2013 presso il 108° Congresso della Società Botanica Italiana, è al momento utilizzata e raggiungibile all'indirizzo web http://erbario.biologia.unipd.it e conta quasi mille fogli inseriti.

Ringraziamenti

Ringrazio i miei genitori,
il relatore dott. Sergio Targon Canazza per la sua disponibilità
e la dott.ssa Elena Bergamo per il supporto morale nelle lunghe giornate afose passate al dei.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

1.1 Committente

Il Dipartimento di Biologia dell'Università degli Studi di Padova ha avviato nel 2010 un progetto di catalogazione digitale del materiale d'erbario depositato presso la sua struttura per renderne possibile l'utilizzo e valorizzarlo. Si chiama erbario [2] una collezione di piante o di parti di piante essiccate e pressate accuratamente, individuate e classificate scientificamente. Un foglio di erbario [15] costituisce la singola componente dell'erbario. Nel Dipartimento di Biologia sono depositati oltre 50.000 fogli, prodotti del lavoro di ricercatori e docenti e utilizzabili come materiali di confronto per il riconoscimento delle specie di piante. In ogni foglio, come mostrato in figura 1.1, l'esemplare essiccato è corredato di informazioni riguardanti la sua identificazione e provenienza. La responsabile del progetto, dott.ssa Antonella Miola, ha posto le basi del lavoro con l'aiuto di oltre una decina di studenti del corso di laurea in Scienze naturali, Biologia e in Scienze e tecnologie per l'Ambiente, che, dal 2010 ad oggi, hanno svolto le loro tesi di laurea e/o tirocini su questo progetto. In particolare con la tesi di Clementi M. [11] è stata delineata una struttura di catalogazione digitale in grado di creare dati compatibili con quelli dei database moderni dei più importanti erbari del mondo.

L'azione di progettazione e sviluppo di una soluzione per la digitalizzazione delle risorse è iniziato nel Gennaio 2013 e si è protratta fino ad Agosto 2013.

Per rendere visibili e utilizzabili i materiali d'erbario dall'esterno era necessario sviluppare una piattaforma per la gestione in remoto dei dati e immagini catalogati. La struttura di catalogazione digitale già esistente è stata quindi adattata allo scopo e integrata nel nuovo progetto sviluppato in questa tesi, grazie ad una attiva collaborazione tra progettista e utilizzatore/committente.

Le caratteristiche del sistema dovevano prevedere una zona privata di gestione del catalogo (inserimento/modifica dati e upload immagini) e una zona pubblica di consultazione. Nella zona privata è stato richiesto di:

- gestire l'accesso alla piattaforma attribuendo ruoli diversi e quindi gestire la registrazione degli utenti, il login e modifica dei dati dei singoli utenti;

- gestire l'inserimento dei dati relativi ai fogli d'erbario e di altri dati tecnici ausiliari che costituiscono l'INHP (Index Nominum Herbarium Patavinum), cioè dati specifici sul nome scientifico, il genere, la famiglia e molto altro di una particolare categoria;
- effettuare l'upload di immagini da associare ad ogni foglio.

Parallelamente alla parte privata con l'inserimento e la modifica dei vari record, doveva essere permessa la ricerca dei dati salvati, ovvero una zona accessibile al pubblico per presentare i risultati della ricerca e successivamente anche fornire maggiori dettagli riguardo ai specifici record selezionati.

1.2 Obiettivi

Dall'analisi preliminare é emersa la necessità di sviluppare i seguenti punti:

- gestione della registrazione, della modifica e della cancellazione degli utenti;
- gestione dell'inserimento, della modifica e della cancellazione degli INHP;
- gestione dell'inserimento, della modifica e della cancellazione dei fogli di erbario;
- gestione dell'upload delle immagini;
- gestione della ricerca nel database.

1.3 Linguaggi e strumenti utilizzati

È stato scelto di sviluppare il software in Java e in particolare di sfruttare il web server Apache Tomcat come server HTTP. Vista la necessità di registrare grosse moli di dati, il database MySQL ci é sembrata la scelta migliore viste le ottime prestazioni che vanta, la gratuità e il fatto che sia multipiattaforma. Per facilitare lo sviluppo si é pensato ai framework Spring e Hibernate. Questa scelta ci permette di separare la business logic dall'ambiente in cui il software viene eseguito. Le pagine JSP sono quasi del tutto prive di codice Java, infatti l'utilizzo delle api JSTL permettono la definizione di tag standard che mascherano i fastidiosi innesti di codice rendendo così i sorgenti più leggibili. Attraverso l'integrazione di set già esistenti di funzioni JavaScript come il framework jQuery é stata aumentata l'interattività con l'utente finale. Tra le altre api utilizzate vi è javax.validator per la validazione lato server dei campi delle form, operazione necessaria sia per evitare l'inserimento di dati strutturalmente errati che di codice malevolo. Nel capitolo 3 è presente nel dettaglio un'analisi più approfondita delle loro caratteristiche.

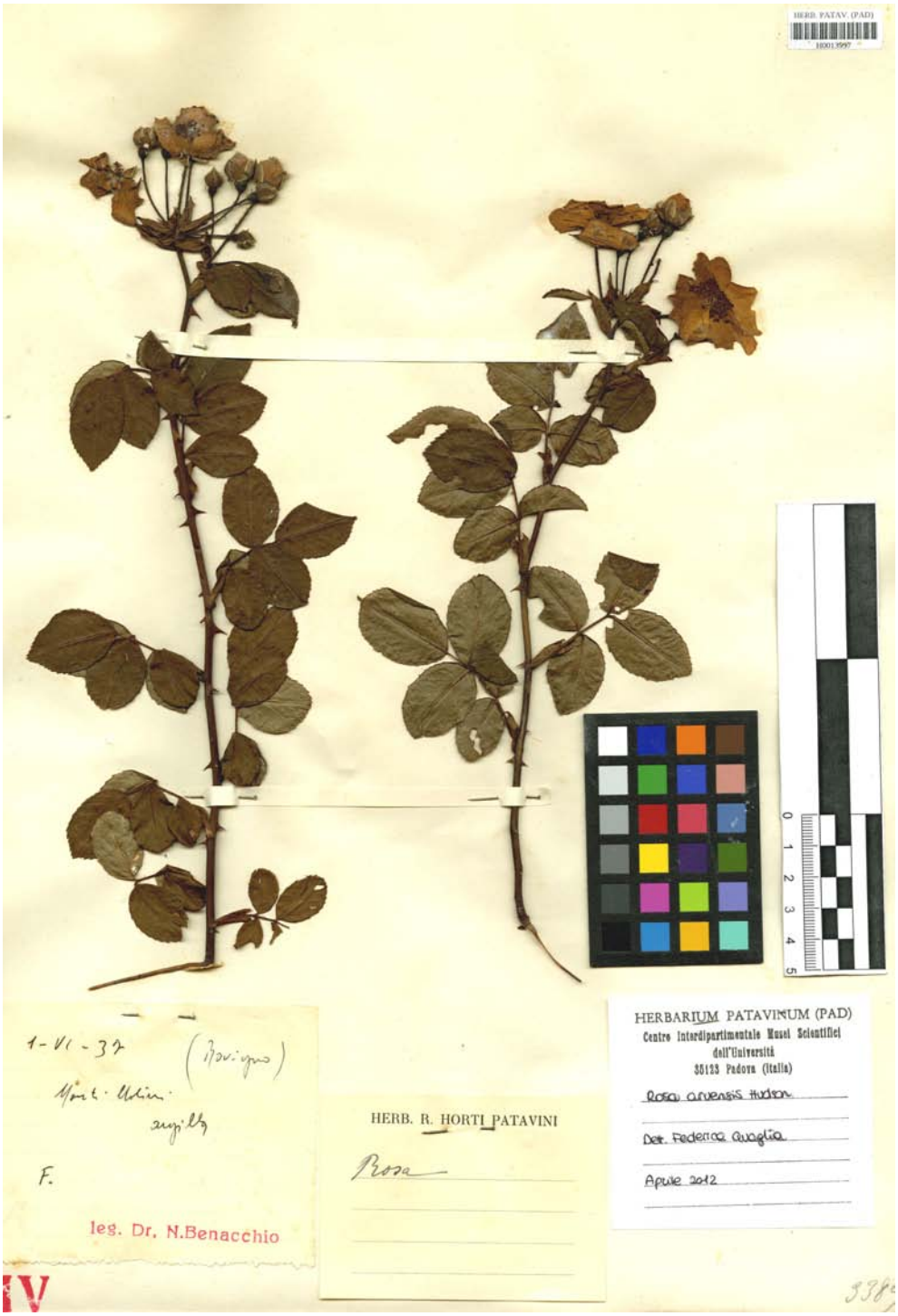

Figura 1.1: *Foglio di erbario*

Capitolo 2

Analisi generale del progetto

2.1 Contesto di utilizzo

Lo scopo dell'applicazione è quello di fornire, tramite il concetto Software as a Service (SaaS), un sistema autonomo e completo per la memorizzazione e la ricerca dei dati dei fogli di erbario dell'Herbarium Patavinum conservati presso il Dipartimento di Biologia. Il sistema potrebbe essere, in un futuro, utilizzabile anche per la gestione dei materiali depositati presso la sede principale dell'Herbarium Patavinum, che è uno dei più antichi in Italia e quindi potrebbe essere interessato a nuove tecnologie di catalogazione, che garantiscano un più facile accesso alle sue collezioni. L'applicazione permetterà infatti di condurre ricerche sui fogli di erbario senza la necessità di andare di persona a studiarli. Inoltre la memorizzazione digitale di tutti i dati offre un alto livello di ridondanza così da assicurare una conservazione migliore dell'intero erbario patavino. Il salvataggio dei dati verrà effettuato da studenti e docenti i quali, dopo la registrazione e la convalida dell'account, potranno trascrivere i dati sul terminale via web, da qualunque postazione. I fogli di erbario sono uno ad uno distinguibili grazie ad un sistema di codici a barre, i quali sono disposti in alto a destra (figura 1.1). Uno scanner ad alta risoluzione provvederà a fornire le immagini dei fogli in formato JPEG. Per le ricerche sarà disponibile, nella zona pubblica, il necessario per effettuare richieste al database, la possibilità di visualizzare e scaricare i risultati e inoltre analizzare in dettaglio i singoli fogli.

2.2 Descrizione del funzionamento

Il progetto si compone di due macro sezioni: una parte privata relativa alla registrazione e alla modifica dei record del database (fogli di erbario, INHP, utenti) e l'altra pubblica dedicata alla ricerca e alla presentazione dei dati.

2.2.1 Parte privata

Dalla pagina di login, inserendo username e password, è possibile effettuare l'accesso alla zona privata. Sempre nella medesima sezione è possibile registrarsi nel sistema fornendo come dati,

oltre a quelli per l'autenticazione, il nome completo. Ogni utente appena registrato risulta essere tuttavia disabilitato. Essendo un sistema universitario, si presume che l'accesso al sistema debba essere garantito solo a docenti e studenti dell'Università di Padova, in tal modo viene data la possibilità di valutare caso per caso l'idoneità delle persone. Solo gli amministratori possono convalidare gli account abilitandoli.

All'interno della zona privata le possibili azioni sono molteplici, tuttavia vi sono importanti restrizioni a seconda del grado di autorizzazione che possiede l'account. I possibili gradi sono i seguenti: ROLE_PARTICIPANT, ROLE_USER, ROLE_SUPERUSER e ROLE_ADMIN. Tutti gli abilitati ad accedere alla zona privata, con qualsiasi grado di autorità, hanno la possibilità di accedere e modificare i propri dati di profilo. Gli utenti appena registrati hanno tutti il grado di ROLE_PARTICIPANT, ciò significa che non hanno la possibilità di svolgere alcuna operazione sui dati dell'erbario. Con questa scelta si da modo ai nuovi iscritti di ambientarsi nella piattaforma e di verificare i propri dati personali, senza il rischio che danneggino parte dell'erbario. Gli utenti aventi grado ROLE_USER invece possono inserire e modificare i fogli di erbario ed effettuare l'upload delle immagini. Tale grado viene assegnato a tutti gli utenti che hanno una conoscenza parziale della materia o che operano per la prima volta con il sistema. L'aggiunta o il cambiamento dei dati di un foglio di erbario risulta un'operazione tutto sommato semplice: raggiunta la pagina, l'utente è guidato nella compilazione sia dai suggerimenti che appaiono una volta che il cursore si posiziona sopra al campo, sia grazie al sistema di feedback della validazione degli input che segnala gli errori di compilazione e fornisce una possibile soluzione. Gli utenti con grado ROLE_SUPERUSER sono in grado di effettuare le operazioni descritte precedentemente e inoltre inserire o modificare i record INHP. Il definire un utente "superuser" significa attribuirgli conoscenze e responsabilità tali da poter trattare dati sensibili come i record INHP. I dati INHP essendo collegati ai fogli d'erbario risultano essere molto significativi in quanto una modifica errata di uno di essi potrebbe compromettere la correttezza di migliaia di fogli. Infine solo gli utenti con autorità ROLE_ADMIN possono modificare i dati degli altri utenti e in particolare abilitarli nel sistema.

Come detto precedentemente, chi ha grado ROLE_ADMIN può modificare i dati degli utenti e abilitarli nel sistema. Dopo la selezione dell'utente interessato, una pagina presenterà tutti i sui dati ad esclusione della password la quale rimane segreta. Proprio in questa pagina sarà possibile abilitare gli utenti nel sistema oppure eliminarli.

Infine l'upload delle immagini è permesso grazie alle API *fileupload*. Nell'apposita pagina viene richiesto sia il file da caricare che il foglio di erbario. L'associazione nel server avviene sia registrando il nome in un campo della tabella Ogg che rinominando il file con la dicitura del codice a barre. Il codice a barre viene usato come chiave primaria e dunque l'unicità del nome dell'immagine è assicurata. Tutti i file vengono salvati in una cartella unica. Se un foglio di erbario possiede già un'immagine, questa viene sovrascritta. Oltre all'upload, viene creata una miniatura di pochi kilobyte da usare nella presentazione dei fogli durante le ricerche. Nonostante il carico computazionale sia maggiore ed inoltre si occupi maggior spazio di archiviazione, questa soluzione permette un risparmio di banda considerevole nell'esposizione dei risultati di una ricerca.

2.2.2 Parte pubblica

La parte pubblica, oltre alla pagina di presentazione del servizio, prevede solo la possibilità di visionare i dati nel database. Una volta effettuata la ricerca tramite la compilazione di un apposito campo e la scelta del tipo di ordinamento, i risultati vengono impilati e formano più pagine di una tabella di consultazione. Annesso ad ogni foglio, sarà possibile visualizzare la miniatura della scannerizzazione del foglio. I risultati possono essere scaricati in formato excel, oppure, una volta selezionato uno specifico record, essere presentati nel dettaglio con tutte le loro informazioni compresa l'immagine nelle sue dimensioni reali. Vista la mole di dati che una singola ricerca può richiamare, abbiamo deciso di utilizzare la tecnica del *lazy load*. Ogni query viene arricchita con l'opzione *LIMIT* e usando AJAX, ad ogni evento di modifica della ricerca, i dati vengono aggiornati senza il bisogno di effettuare il refresh della pagina.

2.3 Funzionalità aggiuntive

Alle funzionalità di base precedentemente descritte ne vengono aggiunte altre che hanno lo scopo di migliorare l'usabilità e l'efficienza del software:

- controllo di regolarità dei dati tramite javascript. Per garantire una più facile immissione delle date, jQuery mette a disposizione un set di funzioni per facilitare l'inserimento di date secondo un certo format configurabile;
- alcune liste come quella dei codici a barre o dei nomi degli INHP, sono trasferite direttamente dal server all'input di competenza, così da fornire un aiuto di tipo compilativo all'utente. In tal modo si accelerano le operazioni di inserimento e soprattutto si evitano errori di battitura;
- composizione dinamica delle pagine per modificare e aggiungere i fogli di erbario. Ogni foglio di erbario può avere un numero di pubblicazioni e di determinazioni differente. Grazie ad opportune funzioni javascript, è possibile aggiungere o rimuovere i campi nella compilazione delle form rendendo di fatto l'impaginazione dinamica.

2.4 Interfaccia

La presenza di una parte pubblica rende necessaria l'integrazione grafica con il template usato dai siti dell'Università di Padova. Il sito è stato quindi adattato al form factor preesistente, in particolare si nota la presenza nell'intestazione del logo dell'università, poco più sotto il menu orizzontale ricorrente in tutte le pagine e sul fianco sinistro una lista di voci, in particolare, la sezione estesa "Patrimonio artistico e culturale". La pagina principale prevede diverse possibilità di navigazione: mette a disposizione l'indirizzamento alla pagina per le ricerche e propone l'accesso alla parte privata (figura 2.1).

La ricerca consiste nel confronto lessico tra l'input inserito dall'utente e alcuni determinati campi che fanno parte degli INHP, dei fogli di erbario e delle determinazioni associate. Tra le

opzioni messe a disposizione vi è l'ordinamento per codice a barre, genere, famiglia e specie. La figura 2.2 mostra un esempio dell'aspetto grafico della sezione. Si è scelto di non usare form ma AJAX. Le comunicazioni con il server sono realizzate sfruttando il metodo *doAjaxPost()* della libreria jQuery. La tabella dei risultati è invece inizializzata con il metodo *dataTable()* importato dalla libreria DataTables. La richiesta del client si compone di tutti gli input presenti nella pagina. I dati vengono inviati con metodologia *GET* così che sia possibile salvare o copiare le ricerche fatte. Il server elabora la richiesta e ritorna una stringa avente come primo dato il numero di record risultanti e successivamente il *JSON* con i dettagli. Se la richiesta AJAX ha avuto successo, la risposta viene suddivisa in parti e in particolare il *JSON* viene utilizzato per popolare la tabella.

L'eventuale selezione di un particolare foglio ne permette la sua visione nello specifico di tutti i campi e la presentazione dell'immagine nelle sue dimensioni reali (figura 2.3).

L'accesso alla parte privata avviene tramite l'apposito link di autenticazione. La pagina di login richiederà username e password (figura 2.4). Nel caso in cui non si possedesse un account, sarà possibile effettuare la registrazione previa accettazione delle condizioni sulla privacy ed il trattamento dei dati personali. Una volta effettuata l'autenticazione, si entra nella parte privata del sito. A seconda del grado d'accesso sarà possibile o no compiere alcune azioni. Il layout della pagina è ripreso da quello della parte pubblica con la differenza che viene sostituito il menu a sinistra con uno adatto alla parte privata del sito (figura 2.5). Tra le prime opzioni troviamo le voci per l'aggiunta e la modifica dei fogli di erbario. La composizione degli input della form è favorita da una nota descrittiva posizionata in ogni campo quando l'utente si posiziona nelle vicinanze. Inoltre l'inserimento delle date, dei nomi di persona, delle pubblicazioni e delle determinazioni sono integrati con del codice javascript per permettere la comparsa di tendine e di schede aggiuntive così da soddisfare tutti i casi di inserimento senza appesantire la pagina. Nel dettaglio tutti i campi *data* hanno un id in modo da identificare l'oggetto e collegarlo con il metodo *datapicker({ dateFormat: yy-mm-dd })* della libreria jQuery. Oltre ad un controllo di validità del dato, il metodo fa apparire un calendario a scomparsa. Per i campi nei quali viene richiesto di indicare il nome o l'abbreviazione di persone si è pensato di fornire un aiuto compilativo tramite la lista di tutti i nomi inseriti nel database nella tabella *per* e l'applicazione del metodo *autocomplite({source: availablePers})* della libreria jQuery. In tal modo alla battitura delle prime lettere, appare una lista con i possibili completamenti. Le pubblicazioni e le determinazioni sono invece gestite modificando il codice html in runtime con javascript per aggiungere blocchi di campi (figura 2.6).

L'upload delle immagini avviene nella sezione Upload File nella quale dopo aver selezionato l'immagine e il codice a barre del foglio, il tasto apposito provvederà ad effettuare la richiesta di archiviazione (figura 2.7).

Anche la sezione di aggiunta e modifica degli INHP risulta intuitiva e simile a quella per i fogli (figura 2.8).

La modifica di un utente esistente avviene prima selezionandolo dall'apposita lista e successivamente, modificando i relativi dati nei campi corrispondenti, avviando l'operazione. Nella stessa pagina è anche possibile eliminare l'utente (figura 2.9). L'eliminazione dell'utente non costituisce una perdita di dati dei fogli in quanto, come presentato nel capitolo 4 (figura 4.1), la tabella *users* non si relaziona con quelle che registrano i dati dell'erbario. Infine è possibile

modificare alcuni dei dati del proprio profilo. Nella stessa pagina è possibile anche richiedere la disattivazione del proprio account. Una volta disattivato l'account, l'utente non potrà effettuare l'accesso se non previa riattivazione dello stesso (figura 2.10).

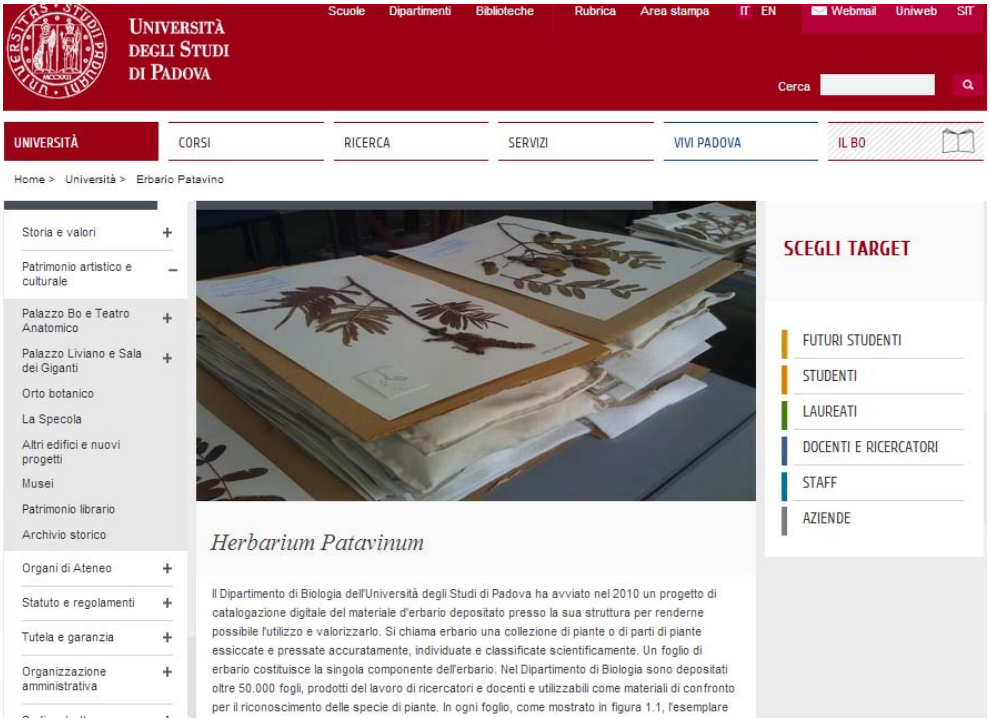

Figura 2.1: *Home page sito dell'Herbarium Patavinum*

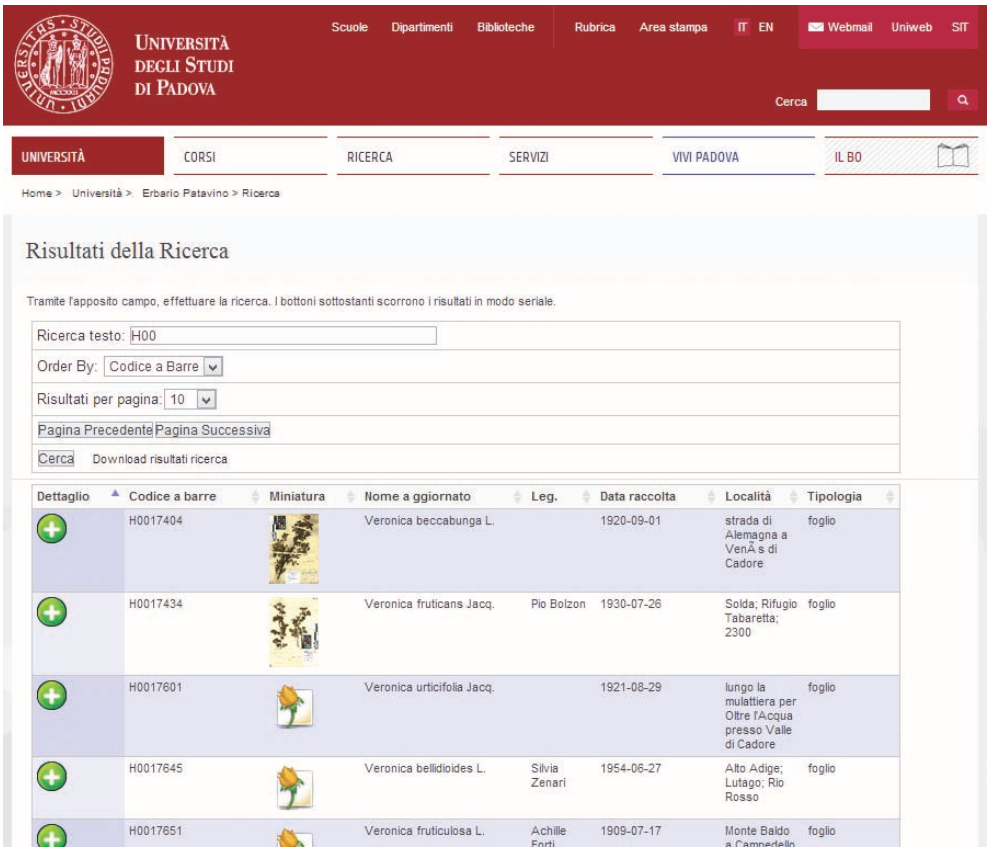

Figura 2.2: *Pagina web della funzione "Ricerca"*

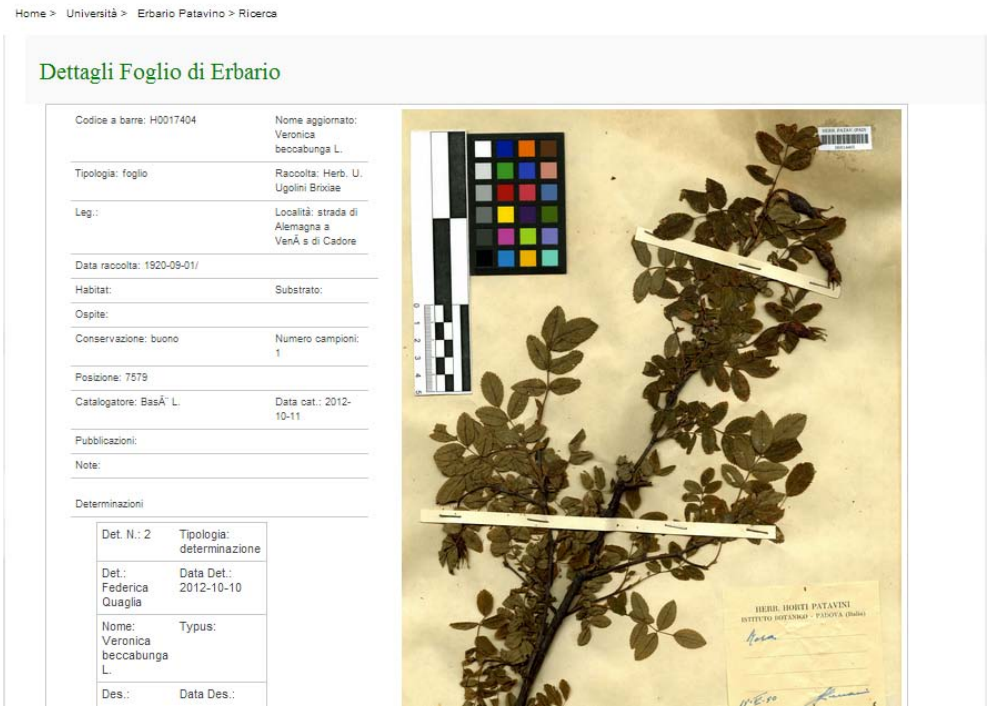

Figura 2.3: *Pagina web di esposizione dettagliata di un foglio di erbario*

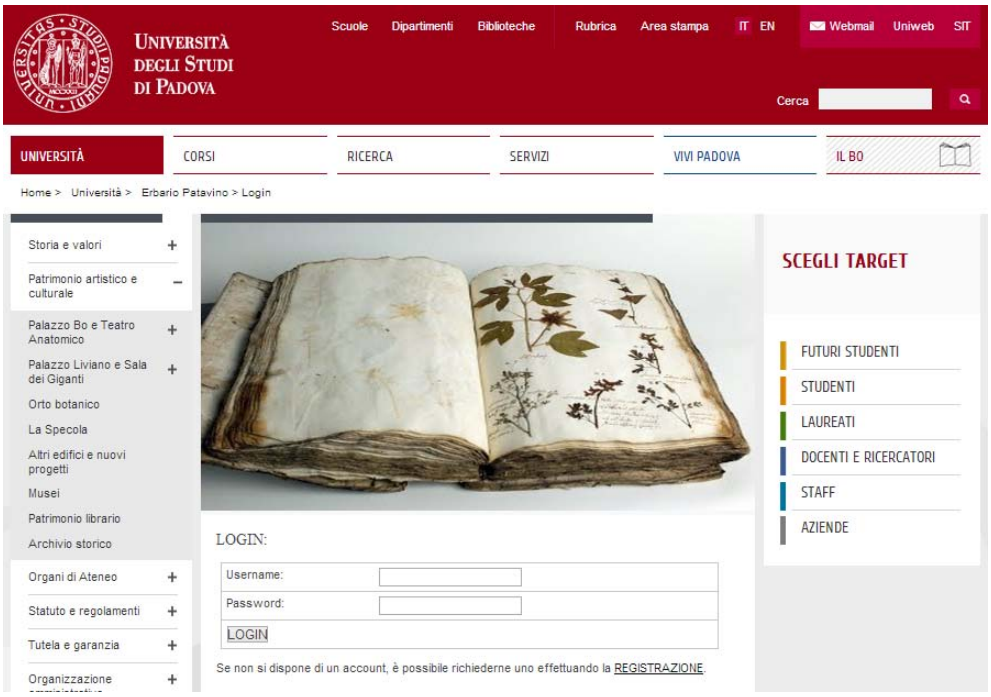

Figura 2.4: *Pagina di autenticazione della piattaforma*

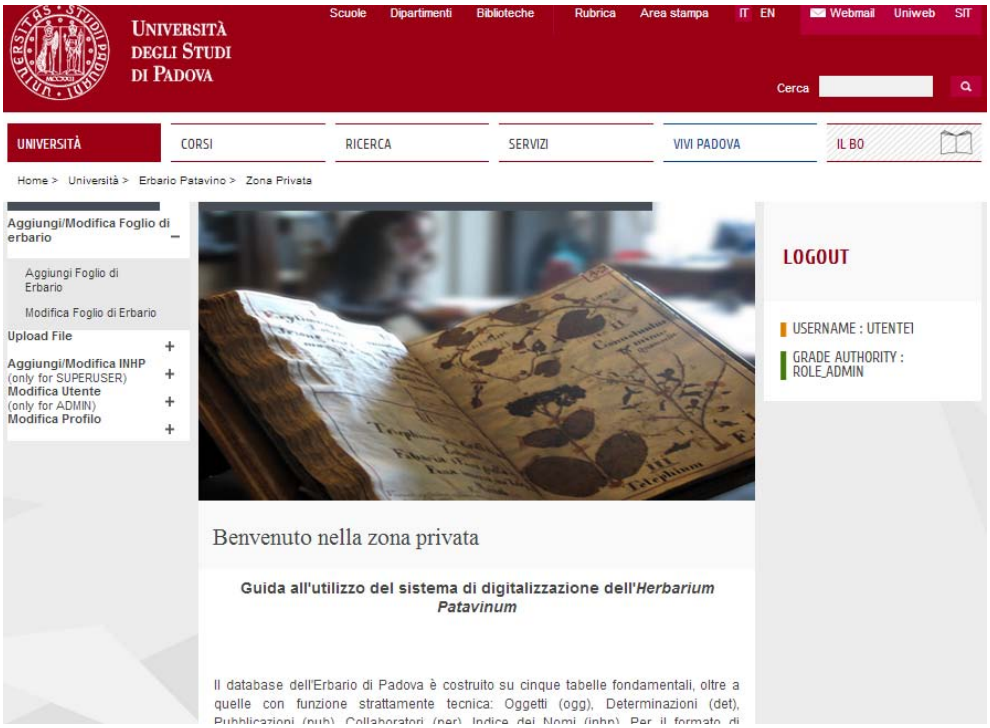

Figura 2.5: *Pagina principale della sezione privata del sito web*

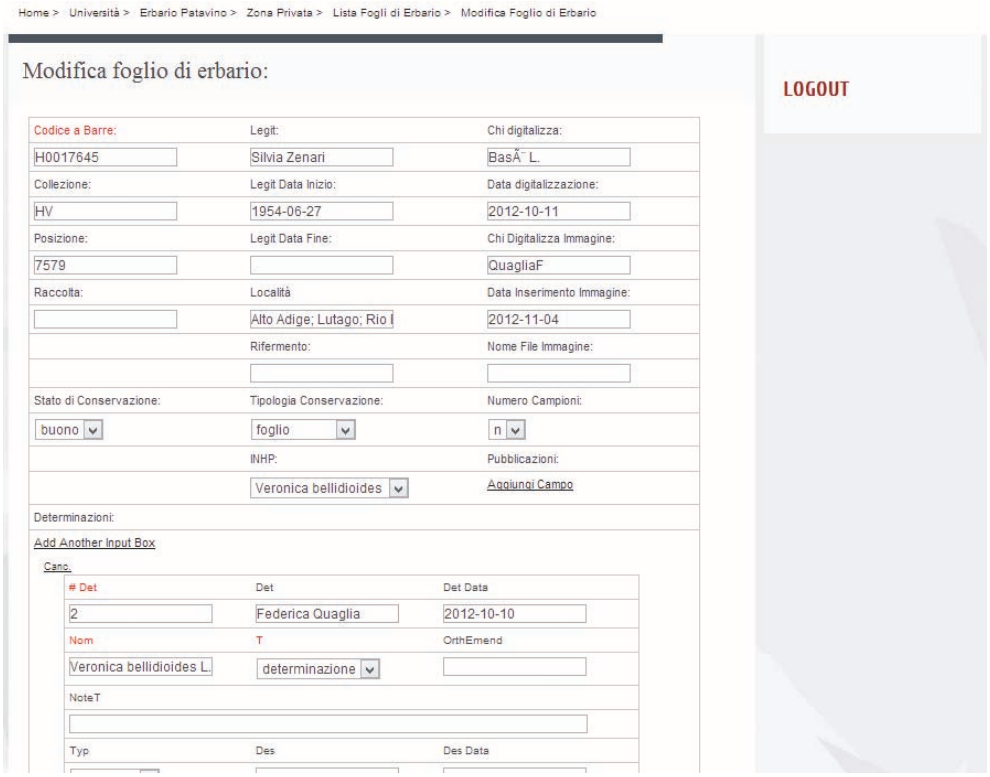

Figura 2.6: *Pagina di inserimento dei dati di un foglio di erbario nel sistema*

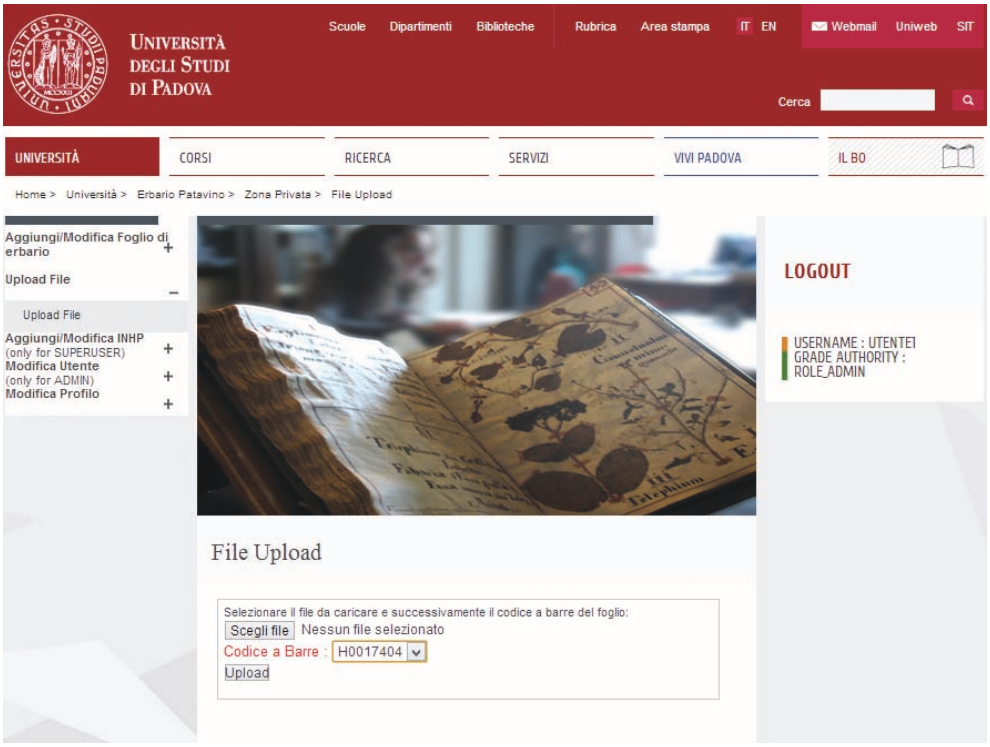

Figura 2.7: *Pagina di caricamento delle immagini dei fogli di erbario*

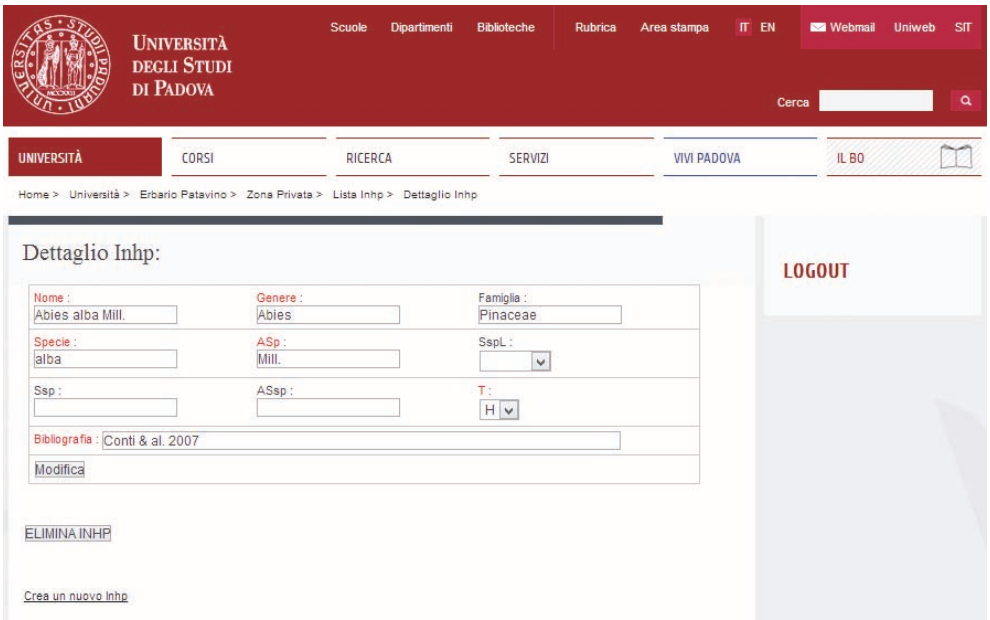

Figura 2.8: *Pagina di inserimento dei dati degli INHP nel sistema*

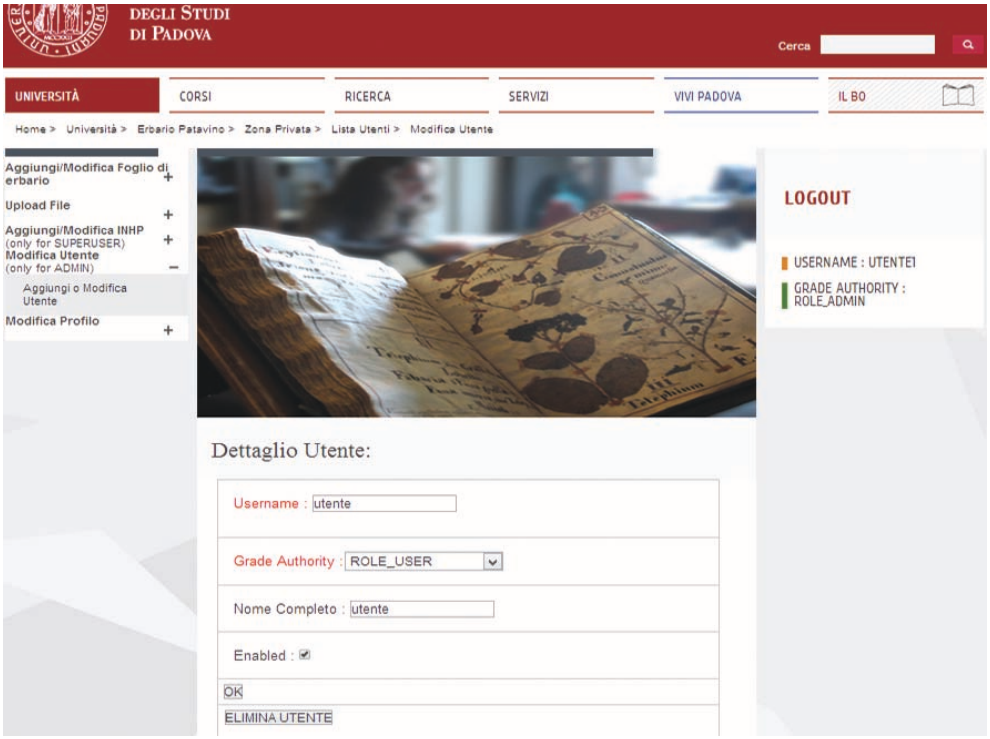

Figura 2.9: *Pagina di modifica dei dati degli utenti*

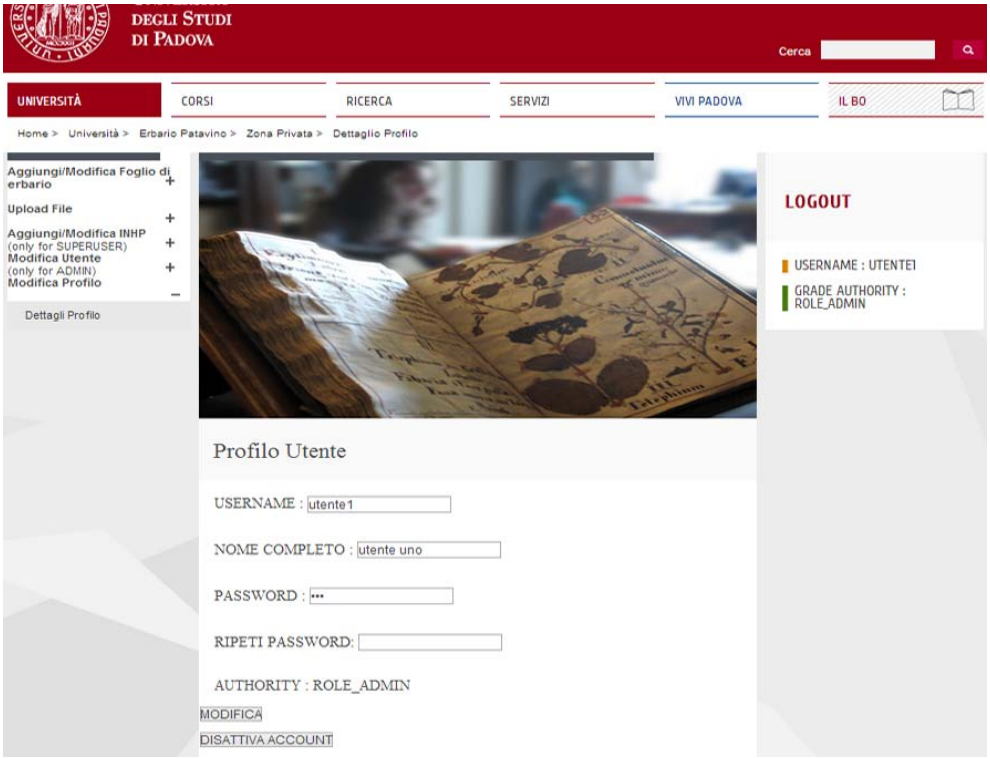

Figura 2.10: *Pagina profilo personale degli utenti*

Capitolo 3

Strumenti e tecnologie utilizzate

Di seguito vengono elencati e descritti i principali strumenti utilizzati nel progetto.

Figura 3.1: *Alcuni loghi degli strumenti utilizzati*

3.1 Spring

Spring [7, 12] è uno dei più famosi framework per lo sviluppo di software Java enterprise. Milioni di sviluppatori utilizzano Spring con l'obiettivo di creare programmi performanti, facili da testare, con un alto tasso di codice riutilizzabile senza lock-in. Spring fornisce strumenti di programmazioni completi e modelli di configurazione per le moderne applicazioni Java-based enterprise per ogni tipo di piattaforma di sviluppo. Visto ormai come una ben solida alternativa al modello basato su Enterprise Javabeans, questo framework supera l'appesantimento portato dall'utilizzo forzato di interfacce EJB di tipo home e remote, troppo invasive nel codice scritto, pur mantenendo la gestione del deployment descriptions in XML grazie a nuovi e innovativi modelli di programmazione, come l'Aspect Oriented Programming (AOP) e l'Inversion of Control

(IoC). Un elemento chiave di Spring è il supporto infrastrutturale a livello di applicazione: Spring si concentra sull' "impianto idraulico" del progetto così i team di sviluppatori sono facilitati nella parte business logic, senza essere vincolati dall'ambiente in cui il software viene distribuito. Ne risulta un framework leggero e grazie alla sua architettura estremamente modulare è possibile utilizzarlo nella sua interezza o solo in parte. La sua adozione in un progetto è molto semplice, può avvenire in maniera incrementale e non ne sconvolge l'architettura esistente. Questa sua peculiarità ne permette anche una facile integrazione con altri framework esistenti, come ad esempio Hibernate e PrimeFaces. Infine, esso mette a disposizione una serie completa di strumenti per gestire la complessità dello sviluppo software, fornendo un approccio semplificato sia ai più comuni problemi di sviluppo (accesso ai database, gestione delle dipendenze, etc.) che di testing.

3.2 Hibernate

Un ORM (acronimo di Object Relational Mapping) svolge essenzialmente una funzione di mappatura fra un database di tipo relazionale e un linguaggio di programmazione ad oggetti, ovvero garantisce la persistenza dei dati e astraendo nel contempo le caratteristiche implementative dello specifico RDBMS utilizzato. Hibernate [8, 3] è un ORM e in sostanza fa da ponte tra java e un database mappando ciascun record di quest'ultimo come oggetto Java. In particolare Hibernate lavora tramite i cosiddetti POJO, ossia oggetti Java piuttosto semplici, dotati di campi più o meno numerosi e di metodi di tipo setters e getters che vanno ad interagire con essi. Tali POJO rappresentano le entità che andranno gestite nel database. è possibile configurare gli oggetti POJO o tramite opportuni file di configurazione XML oppure con le Annotation, in tal modo, viene concesso all' utente di specificare come i POJO dovranno poi essere mappati verso il RDBMS.

3.3 JSTL

JavaServer Pages Standard Tag Library (JSTL) [6] è una collezione di utili tags JSP che esprimono le principali funzionalità di molte applicazioni JSP. JSTL supporta comuni tags, per operazioni strutturali come iterazioni e condizioni, per le manipolazioni di documenti XML, tags di internazionalizzazione e localizzazione e tags SQL. In questo modo, un tag personalizzato incapsula una funzionalità collocata in una classe java detta "tag handler". Alcuni vantaggi derivanti da tale approccio sono:

- la pagina JSP risulta più leggibile: invece che presentare codice html inframezzato da scriptlet Java, essa presenta tag html e tag personalizzati;

- un tag personalizzato è riusabile: la funzionalità contenuta nella classe tag handler è richiamabile più volte all'interno di pagine JSP.

3.4 jQuery

jQuery [9, 10] è una libreria di funzioni javascript cross–browser, che si propone come obiettivo quello di semplificare la programmazione lato client delle pagine HTML. Fornisce un codice sintetico, limitando al minimo l'estensione degli oggetti in modo da mantenere la massima compatibilità con altre librerie. "Write less, do more" è infatti il motto di jQuery che sottolinea allo stesso tempo la semplicità e la potenza di questo framework. Tale framework fornisce dunque metodi e funzioni per gestire al meglio gli aspetti grafici e strutturali come l'accesso e la manipolazione degli elementi e dei loro attributi, il riconoscimento e la programmazione degli eventi e degli effetti che questi comportano.

3.5 Javax Validation

Javax Validation è una API il cui scopo è quello di facilitare la validazione dei campi. Tramite delle specifiche annotazioni è possibile preprocessare gli oggetti ancora prima dell'entrata in esecuzione del metodo chiamato così da assicurare successivamente il corretto funzionamento del programma. L'aspetto più importante di questo tipo di controllo è che non può essere disattivato o manipolato come succede per esempio con le validazioni tramite JavaScript, assicurando così maggiore sicurezza ai dati elaborati.

3.6 MySQL

MySQL [1] è un Database Management System, ovvero un sistema per la gestione dei dati. È un progetto Open Source sotto la licenza GPL, GNU General Public License, disponibile gratuitamente per molte piattaforme e permette il salvataggio, la manipolazione e l'estrazione di dati tramite il linguaggio SQL. La flessibilità della piattaforma è una funzionalità chiave di MySQL, che è in grado di supportare Linux, UNIX e Windows.

3.7 Apache Tomcat

Apache Tomcat [16, 5] (o semplicemente Tomcat) è un contenitore servlet open source sviluppato dalla Apache Software Foundation. Implementa le specifiche Java Server Pages (JSP) e Servlet di Sun Microsystems, fornendo quindi una piattaforma software per l'esecuzione di applicazioni Web sviluppate in linguaggio Java. La sua distribuzione standard include anche le funzionalità di web server tradizionale, che corrispondono al prodotto Apache. è stato scelto Tomcat perchè è semplice da utilizzare, molto stabile, affidabile e gratuito.

3.8 MySQL Workbench

MySQL Workbench è uno strumento visuale unificato per architetti di database, sviluppatori e DBA (Amministratori di Database). MySQL Workbench consente modellazione dei dati, sviluppo SQL e fornisce un set completo di strumenti amministrativi per la configurazione del server e l'amministrazione degli utenti. MySQL Workbench è disponibile in Windows, Linux e Mac OS. Ne risulta quindi uno strumento comodo e soprattutto completo per velocizzare il management di un database. Tale applicativo è stato utilizzato nelle modifiche delle tabelle e nelle migrazioni dei records da basi di dati secondarie che utilizzavano logiche di storage differenti, inoltre si è dimostrato un prezioso strumento per velocizzare la verifica della scrittura/lettura dei dati nel database.

3.9 JavaScript

JavaScript [17] è un linguaggio di scripting orientato agli oggetti comunemente utilizzato in ambito di programmazione web. La caratteristica principale di JavaScript è l'essere un linguaggio interpretato: il codice non viene compilato, ma interpretato (in JavaScript lato client, l'interprete è incluso nel browser che si sta utilizzando). Dai primi sviluppi, quando si chiamava ancora LiveScript, la sintassi è stata forzatamente avvicinata a quella di Java (da qui il nome JavaScript). Nonostante il fatto di essere un linguaggio interpretato, definisce le funzionalità tipiche dei linguaggi di programmazione ad alto livello (strutture di controllo, cicli, ecc.) e consente l'utilizzo del paradigma object oriented. Come detto, JavaScript viene tipicamente utilizzato nei web browser, quindi il codice viene direttamente interpretato sul client e non sul server così da poter lasciare al client il peso di ulteriori elaborazioni sui dati (es.: field validation, rimpiazzo delle immagini, la creazione di finestre pop-up). Nonostante in principio la diffusione di questi script fosse principalmente legata a banner pub- blicitari, negli ultimi anni ha visto una rapida ascesa al successo con la comparsa di AJAX.

3.10 AJAX

AJAX [13] (Asynchronous JavaScript and XML) è una tecnica di sviluppo software per la realizzazione di applicazioni web interattive. Lo sviluppo di applicazioni HTML con AJAX si basa su uno scambio di dati in background fra web browser e server, che consente l'aggiornamento dinamico di una pagina web senza esplicito ricaricamento da parte dell'utente. AJAX è asincrono nel senso che i dati extra sono richiesti al server e caricati in background senza interferire con il comportamento della pagina esistente. L'uso di questa tecnica porta ad una velocità di caricamento delle pagine molto maggiore rispetto alle applicazioni che usano la comunicazione sincrona, poichè viene ridotta in modo considerevole la quantità di dati scambiati. Anche il tempo di elaborazione delle informazioni da parte del server si riduce notevolmente, dato che non deve elaborare i dati relativi all'intera pagina ma solo la parte corrispondente alla richiesta.

3.11 DataTables

DataTables [14] è un plug-in per la libreria jQuery, uno strumento molto flessibile, basato sul progressivo miglioramento delle sue parti le quali aggiungono controlli di interazione alle tabelle. I punti di forza di questo set di funzioni sono: lunghezza di paginazione variabile, filtri in real time, ordinamenti multi colonna con riconoscimento del tipo di dato automatico, impaginazione automatica, elaborazioni di diversi tipi di sorgenti dati (es.: DOM, javascript array, Ajax file) e molto altro. DataTables risulta molto semplice da usare e garantisce una notevole flessibilità nell'adattarsi alle diverse situazioni di utilizzo. Nel progetto si è fatto uso di questo plug-in in combinazione con AJAX: ad ogni ricerca, la risposta dal server viene fornita tramite JSON, di volta in volta viene aggiornata la sorgente dati della tabella e ridisegnata con i nuovi dati.

3.12 Apache Poi

Il progetto Apache Poi si pone come obiettivo quello dello sviluppo di API Java per la manipolazioni di diversi formati basati sugli standard Office Open XML (OOXML) e OLE 2 Compound Document (OLE 2). La libreria permette di leggere e scrivere documenti MS Excel usando Java. In aggiunta è possibile leggere e scrivere file MS Word e MS PowerPoint.

Capitolo 4

Realizzazione

Le fasi di realizzazione del progetto sono state molteplici nel loro insieme, ma principalmente sono divisibili in due macro parti:

- revisione e riprogettazione di una base di dati;
- progettazione e sviluppo dell'applicazione.

4.1 Progettazione della base di dati

Il primo step di sviluppo è stato lo studio e la correzione di alcuni errori nel progetto svolto da un laureando di biologia dell'Università degli Studi di Padova, il cui lavoro riguarda la realizzazione di un sistema di catalogazione digitale della flora. In base all'analisi dei requisiti presentata nel capitolo 2, è stato scelto un tipo di sviluppo *buttom-up,* ovvero, data la conoscenza preliminare di tutte le singole parti, si è sviluppata ogni tabella nel dettaglio e successivamente si è unito il tutto.

Il database utilizzato risulta quello in figura 4.1: si noti che lo schema non presenta ridondanze, e non vi sono relazioni molti a molti.

Nella stessa figura si possono riconoscere due blocchi separati:

- il primo, avente come tabella centrale *ogg*, è il vero e proprio sistema di catalogazione della flora. Ogni foglio d'erbario presenta delle informazioni generali che caratterizzano il foglio in sè (es. chi ha raccolto la pianta, la data di raccolta, il codice di catalogazione, la sua posizione, ecc..). Queste informazioni sono registrate in parte nella tabella *ogg* e in parte suddivise in 5 tabelle ausiliarie relazionate con *ogg*: la relazione N-1 rispettivamente tra *rifpub* e *ogg*, la relazione N-N tra *ogg* trasformata in N-1,N-1 tra *rif* e *ogg*, la relazione N-1 tra *det* e *ogg*, le relazione 1-N tra *per* e *ogg*, la relazione 1-N rispettivamente tra *rifgeo* e *ogg* e infine la relazione 1-N tra *inhp* e *ogg*. La tabella *rifpub* è adibita al salvataggio delle pubblicazioni, le quali possono essere molteplici oppure possono non esserci affatto. *rif* contiene i possibili riferimenti tra diversi fogli. In questo caso, nonostante sia stata implementata la possibilità di inserire un unico riferimento, tale soluzione facilita i possibili sviluppi futuri nel momento in cui si reputino necessari diversi riferimenti allo stesso

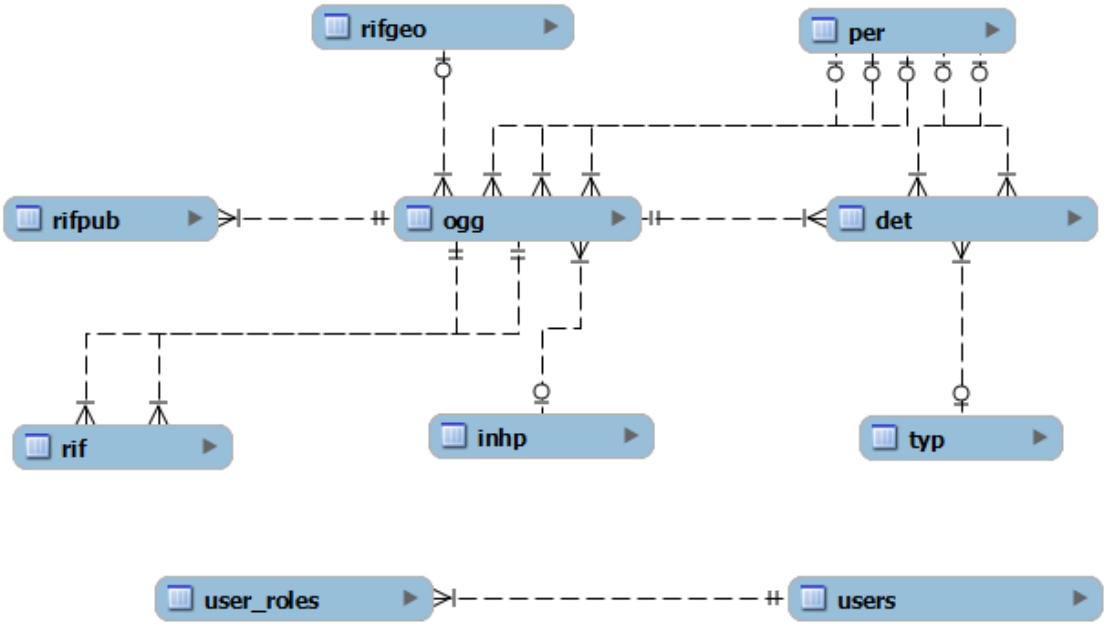

Figura 4.1: *Schema ER della base di dati*

foglio. Per quanto riguarda le determinazioni, invece, sono un'interpretazione soggettiva di catalogazione della pianta per cui nel tempo uno stesso foglio può essere più volte determinato. Inoltre a lungo andare una stessa pianta può chiamarsi in modi differenti a seconda della nomenclatura esistente, è quindi necessario un campo di testo apposito ovvero *Nom* della tabella *det* così da tenere traccia di questi cambiamenti nel tempo e poterli confrontare con la nomenclatura attualmente presente nella tabella INHP.

- il secondo blocco è funzionale alla registrazione degli utenti nel sistema per il login e la determinazione delle aree di accesso. È stato scelto di dividere la tabella contenente i dati utenti (*users*) con quella contente i ruoli (*user_roles*) nonostante il progetto preveda la presenza di un unico ruolo per utente, così da poter utilizzare il sistema di autenticazione *Spring Security:* in tal modo il sistema risulta più standard facilitando le possibili future modifiche.

4.2 Progettazione del software

È stato scelto di sviluppare il software seguendo il modello a cascata ovvero strutturare lo sviluppo in un sequenza lineare di fasi o passi, che comprende:

- analisi dei requisiti;
- progetto;
- sviluppo;

- collaudo;
- manutenzione.

Si è quindi prima suddiviso in fasi sequenziali il processo di sviluppo e ad ogni fase l'output prodotto è stato usato come input per quella successiva. Dall'analisi dei requisiti si è evidenziata la necessità di utilizzare dei framework per restringere i tempi di sviluppo e standardizzare la struttura del programma. In particolare viene usato Spring per la caratteristica web-based del software e Hibernate per gestire la base di dati. È stato scelto di utilizzare Spring in quanto possiede le seguenti peculiarità:

- modulare: pur essendo molto ampio, è possibile scegliere di integrare solo alcuni dei suoi moduli all'interno del proprio progetto;
- leggero: le dipendenze del framework all'interno della propria business logic risultano praticamente nulle;
- integrabile: non sono presenti funzionalità di logging, connection pool ecc, in quanto l'obiettivo di Spring non è quello di sostituire gli altri framework ma di integrarli più facilmente;
- portabile: l'applicazione che fa uso di Spring può essere trasferita senza problemi da un Application Server all'altro;
- POJO-based: ci permette di sviluppare una completa applicazione J2EE usando solo POJO;
- inclusione dei test driver: ci permette di scrivere software facile da testare.

Grazie a Spring si è implementato l'Inversion of Control e il design pattern MVC.

4.2.1 Inversion of Control e Dependency Injection

Il pattern *Inversion of Control (IoC)* sta alla base di Spring. Grazie a questa pratica di programmazione si riesce a minimizzare le dipendenze che intercorrono tra gli oggetti, ciò significa rendere l'applicazione più robusta, modulare e facilitarne il riutilizzo dei componenti. Con IoC si inverte letteralmente il controllo del flusso (*Control Flow*) rispetto alla programmazione tradizionale, la quale lascia al programmatore definire la logica di tale flusso. Cioè non sarà più lo sviluppatore che si farà carico di inizializzare ed invocare i metodi degli oggetti coinvolti nel flusso applicativo, ma bensì il framework, che inietterà le dipendenze direttamente nelle classi.

Una delle tecniche con le quali si può attuare l'IoC è la *Dependency Injection*. Grazie a questa, gli oggetti che verranno utilizzati nel flusso applicativo saranno creati da un componente esterno (*Container*), che si occuperà di creare l'oggetto stesso, le relative dipendenze e di assemblarle attraverso l'uso dell'injection.

Grazie a dei file di configurazione XML, il Container inietta le dipendenze direttamente nei bean (che in Spring può essere rappresentato da una qualunque classe Java) e ne gestisce l'intero ciclo di vita.

4.2.2 Pattern MVC

Spesso, quando si implementa un'applicazione Web utilizzando la tecnologia Java, a causa dell'utilizzo di un protocollo "povero" come HTTP si tende ad usare in modo indiscriminato Servlet e pagine JSP (JavaServer Pages). Se questo da un lato da la possibilità di una maggiore flessibilità e libertà di programmazione, dall'altro può andare contro le norme della qualità del software, come riusabilità, portabilità e manutenibilità.

Per sopperire a tali esigenze, si è designata un'architettura per le Web application chiamata Model-View-Controller. Tale design pattern risulta una pietra miliare per l'architettura di tutte le applicazioni Web-based, poichè riesce brillantemente a separare tutta la logica business e la rappresentazione delle informazioni dalle interazioni utente con quest'ultime. Tale modello è costituito sostanzialmente da 3 strati: la parte business formata da *Model* e *Controller*, e l'interfaccia utente *View*).

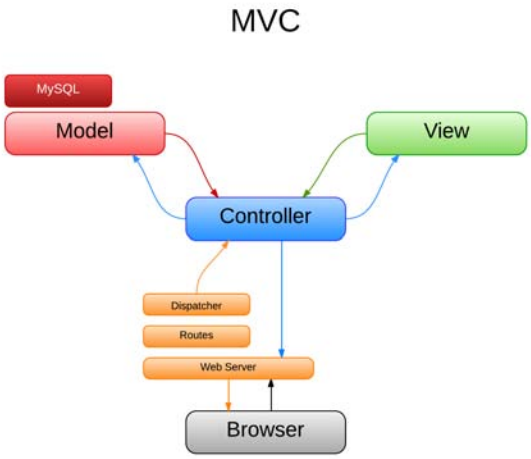

Figura 4.2: *Schema pattern MVC*

Nel dettaglio:

- **Model:** implementa la logica di business, fornendo i metodi utili per l'accesso ai dati dell'applicazione ed ha la responsabilità della gestione del database.
- **Controller:** implementa la logica di controllo, riceve i comandi dell'utente (in genere attraverso lo strato view) e li attua modificando lo stato degli altri due componenti.
- **View:** implementa la logica di presentazione, interpretando i risultati ottenuti dal controller e gestendo l'interazione con gli utenti.

Si noti come il model sia indipendente dal controller e dalla view, esso è uno dei fattori più importanti di questa architettura, poichè permette al modello di essere implementato e testato indipendentemente dallo strato di visualizzazione. Grazie a questi accorgimenti, i benefici derivati

sono molteplici: grazie a tale approccio, è possibile implementare viste multiple, permettendo l'esposizione degli stessi dati allo stesso instante in modi diversi. Si riduce la complessità di sviluppo, e conseguentemente il costo di aggiornamento, permettendo la manutenzione ad uno degli strati senza coinvolgerne altri.

4.2.3 Configurazione di Spring

La configurazione di un progetto che implementa il framework Spring avviene tramite il file XML *web.xml* all'interno della cartella WEB-INF. In quel documento è possibile gestire la creazione delle *dispatcher servlet*, l'importazione di ulteriori file di configurazione per la definizione dei beans, la scelta della pagina iniziale e la mappatura delle cartelle.

Listing 4.1: *Spezzone di codice del file web.xml*

```
<web-app id="WebApp_ID" version="2.4"
      xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
      http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

      <display-name>Spring Application ERBARIO</display-name>

      <!-- Definizione del dispatcher -->
      <servlet>
            <servlet-name>dispatcher</servlet-name>
            <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
            <load-on-startup>1</load-on-startup>
      </servlet>
      <servlet-mapping>
            <servlet-name>dispatcher</servlet-name>
            <url-pattern>/</url-pattern>
      </servlet-mapping>
      <listener>
            <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
      </listener>
      <!-- Importazione dei file di configurazione -->
      <context-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>
    /WEB-INF/spring-database.xml,
    /WEB-INF/applicationContext.xml,
                  /WEB-INF/dispatcher-servlet.xml,
                  /WEB-INF/spring-security.xml
            </param-value>
      </context-param>
                  .
                  .
                  .
                  <!-- Configurazione di JSTL -->
          <jsp-config>
      <taglib>
         <taglib-uri>/spring</taglib-uri>
            <taglib-location>/WEB-INF/tld/spring-form.tld</taglib-location>
         </taglib>
      </jsp-config>
    <!-- Mapping delle cartelle -->
      <servlet-mapping>
         <servlet-name>default</servlet-name>
         <url-pattern>/js/*</url-pattern>
      </servlet-mapping>
```

```
    <servlet-mapping>
        <servlet-name>default</servlet-name>
        <url-pattern>/images/*</url-pattern>
    </servlet-mapping>
</web-app>
```

L'impiego dei beans è di importanza fondamentale per lo sviluppo dell'applicazione. Il software fa uso di tre file xml per la configurazione dei beans:

- dispatcher-Servlet.xml
- spring-security.xml
- applicationContext.xml

Nel file *dispatcher-Servlet.xml* sono inizializzati due oggetti *InternalResourceViewResolver* per mappare in modo automatico tutte le pagine con estensione jsp e html che si trovano all'interno del path *WEB-INF/jsp/*, ovvero tutte quelle che formano la view del progetto. Di seguito vengono inizializzate tutte le classi che svolgono l'attività di controller e in particolare Spring permette di effettuare configurazioni di oggetti tramite annotazioni. È sufficiente inserire *@Controller* prima della dichiarazione della classe e Spring autonomamente la riconoscerà come controllore. Nonostante i molti pregi di Spring, esso non è privo di errori di programmazione: la configurazione di un diverso set di caratteri per le richieste che fanno uso del metodo di invio POST o nell'utilizzo di *ResponseBody* con Ajax, risulta impossibile. Si è resa necessaria la scrittura di una classe ausiliaria che implementa l'interfaccia *BeanPostPocessor* così da sovrascrivere nel modo voluto il metodo *postProcessBeforeInitialization(Object bean, String name)* e infine inizializzarne un oggetto come nuovo bean.

La gestione delle policy d'accesso nelle varie pagine è competenza di Spring Security, un modulo integrabile nel framework Spring. Il file *spring-security.xml* raccoglie tutte le istruzioni di configurazione di questo componente. L'utilizzo di *intercept-url* permette di definire cartella per cartella i limiti di autorizzazione ed eventualmente consente all'utente di raggiungere ed effettuare il login qualora non si possiedano i requisiti richiesti. Il login fa uso del datasource precedentemente creato nel file *spring-database.xml* per cercare e verificare il nome utente e la password inseriti. Nell'appendice A è possibile trovare uno spezzone di codice che ne dimostra l'utilizzo.

L'ultimo file di configurazione, *applicationContext.xml*, contiene informazioni di carattere generale del progetto, l'unico bean presente viene usato per definire la cartella nella quale vengono salvate le immagini.

4.2.4 Hibernate: implementazione e configurazione

Il framework Hibernate fa uso del file *spring-database.xml* per la configurazione della connessione con il database: prima di tutto viene istanziato il datasource che contiene proprietà quali url, username, password e il driver specifico per il RDBMS utilizzato. Ciò significa che l'eventuale scelta di cambiare RDBMS comporta solo la modifica del file scegliendo il driver opportuno. Successivamente viene creato un oggetto *AnnotationSessionFactoryBean* il quale mappa tramite

annotation gli oggetti che rappresentano le entità delle tabelle. Gli oggetti in questione sono dei POJO (Plain Old Java Object) ovvero classi Java le quali non seguono modelli ereditari o framework. I JavaBean sono POJO che non hanno argomenti nel costruttore e permettono l'accesso con i metodi getter e setter seguendo una convenzione legata ai nomi.

Il collegamento tra il progetto e il framework Hibernate avviene con la definizione di un *HibernateTemplate* il quale farà uso dell'oggetto *AnnotationSessionFactoryBean* configurato precedentemente. L'aggiunta e la modifica dei dati nel database avviene con le entità DAO. Questi oggetti estendono la classe *HibernateDaoSupport* e per ogni tabella implementano un tipo di interfaccia. In questo modo facciamo uso della Dipendency Injection e separiamo l'implementazione delle funzioni dal loro utilizzo. Gli oggetti che svolgono questa funzione di interfaccia tra programma e database vengono istanziati una volta sola per sessione e richiamati tramite la comoda funzione di wiring con l'annotazione *@Autowired*. È quindi Spring che si preoccupa di gestire la connessione tra i vari componenti, azione che viene mascherata al programmatore il quale non se ne deve preoccupare.

4.2.5 Model: oggetti DAO e scrittura dei manager

Il DAO (Data Access Object) è una classe che rappresenta un'entità tabellare di un database, viene principalmente usato in applicazioni J2EE e serve a stratificare e isolare l'accesso ad una tabella dalla parte di business logic creando un maggiore livello di astrazione.

Il vantaggio relativo all'uso del DAO è il mantenimento di una rigida separazione tra le componenti di un'applicazione, le quali potrebbero essere il model e il controller in un'applicazione basata sul paradigma MVC. Per ogni tabella si è creato un oggetto DAO il quale definisce i metodi da implementare per la specifica gestione della relativa tabella. L'estensione della classe *HibernateDaoSupport* mette a disposizione la funzione *getHibernateTemplate()* con la quale ottenere un *HibernateTemplate*. La classe HibernateTemplate serve ad automatizzare la fase di inserimento, modifica, cancellazione e ricerca delle query. Grazie agli oggetti DAO si è costruito un set di metodi per la gestione dei dati delle tabella facilitando enormemente la scrittura dei manager. Per rendere compartimentale il progetto si è scelto di scrivere delle comode interfacce per elencare, tabella per tabella, tutte le funzioni possibili. In questo modo si isola il manager che fa uso della particolare funzione dalla sua implementazione. Nell'appendice A è possibile trovare uno spezzone di codice che ne mostra l'utilizzo.

Come per gli oggetti DAO, anche i manager fanno uso di interfacce, lo scopo questa volta è quello di isolare la parte model da quella controller. Sono infatti i controller che richiamano i manager consegnando i dati grezzi provenienti dalla view e gli oggetti DAO inizializzati da Spring. Una delle principali funzioni di queste classi è quella di interpretare i dati e convertirli da tabella a classe ausiliaria per poterli utilizzare nelle view o viceversa. Sono state create cinque interfacce per i cinque manager: *FoglioErbarioManager, FileUploadManager, InhpManager, RicercaManager, UsersManager*. La divisione logica delle funzioni segue principalmente le funzionalità offerte dall'interfaccia, in particolare esiste un manager per aggiungere un foglio di erbario il quale necessita di DAO relativi alle tabelle: *ogg, det, per, users, rifgeo, inhp, rif* e *typ*. Oppure il manager che gestisce l'upload delle immagini il quale non utilizza nessun DAO.

4.2.6 Controller: scrittura dei controller e delle classi ausiliarie alle form

Le classi controller sono tutte caratterizzate dall'annatozione *@Controller* e sono una per ogni sezione del sito. Per dichiararne la presenza a Spring è sufficiente la seguente riga di codice all'interno del file di configurazione *dispatcher-servlet.xml*:

```
<context:component-scan base-package="erbarioOnline.Controller"/>.
```

La parte pubblica è gestita dai controller *RicercaController, RegistrationController* e *LoginController*, quella privata invece: *FileUploadController, FoglioErbarioController, InhpController, ProfiloController, UsersController*. I controller svolgono la funzione di intercettare, gestire e rispondere alle richieste inviate al server. Le richieste vengono suddivise per url e questi sono definiti con l'annotazione *@RequestMapping(value="",...)*. Tra le opzioni di configurazione di *@RequestMapping* vi è la definizione del metodo di invio dei dati con la proprietà *method =RequestMethod.GET/POST*. Di seguito all'annotazione va posto il metodo che gestisce la richiesta il quale avrà come parametri prima di tutto un oggetto contenitore che raccoglie tutte le informazioni inviate dal client, poi a seconda che si voglia validare i dati si inserisce un oggetto *BindingResult* ed infine un *ModelMap*.

La validazione dai capi richiede prima di tutto che si identifichi la classe che svolgerà il controllo tramite il metodo *initBinder(WebDataBinder binder)*. Il controllo sarà richiamabile dopo averlo nominando con l'annotazione *@InitBinder(value="nome")*. In questo modo sarà possibile identificare più oggetti di validazione all'interno della stessa classe e basterà specificare il nome di quello che si desidera utilizzare volta per volta. Il progetto si avvale di sei validator: *DettaglioUtenteValidator, FoglioEbarioFormValidator, InhpFormValidator, ProfiloUtenteValidator, RegistrationValidator, RicercaValidator*. Tutte queste classi implementano l'interfaccia *Validator* del framework Spring e quindi contengono le sovrascritture dei metodi *support(Class clazz)* e *validate(Object target, Errors errors)*. La funzione *support* serve a specificare la classe contenitore sulla quale effettuare i controlli, mentre *validate* è il vero e proprio strumento di controllo dei campi. A seconda dell'oggetto da controllare, vi saranno diversi istruzioni condizionali la cui funzione è di determinare se il particolare campo dell'oggetto trasmesso sia conforme o meno. *Errors* mette a disposizione una lista nella quale raccogliere tutti gli errori che vengono riscontrati nella funzione *validate*. Ogni qual volta un *@RequestMapping* intercetta una richiesta e il metodo associato contiene le annotazione *@ModelAttribute(value="nomeValidator") @Valid*, l'oggetto viene controllato e gli errori elencati nell'oggetto *BindingResult*. Questo tipo di controllo presenta il vantaggio di non poter essere eluso o aggirato visto che viene eseguito dal server. Sistemi di validazione tramite JavaScript invece possono essere modificati in locale e superati. In un sistema in cui la sicurezza non è un fattore trascurabile, l'implementazione di un controllo degli input da parte del server è inevitabile. Purtroppo è anche vero che aumenta sia il carico di lavoro che il tempo di risposta.

Nel momento in cui l'oggetto *BindingResult* contiene almeno un errore si procede con definire all'interno dell'oggetto *ModelMap* l'attributo *error* il quale sarà associato al testo che spiega il motivo del fallimento dell'operazione.

4.2.7 View: utilizzo di JSTL, scrittura delle interfacce web e comunicazioni AJAX

L'utilizzo della libreria JSTL mette a disposizione un set di tag con le quali mascherare il codice Java. Per poter permettere a Tomcat di effettuare questa sorta di associazione è necessario configurare il sistema con il file *spring-form.tld* e dichiararne l'integrazione in *web.xml*. L'operazione risulta di facile attuazione, infatti una volta incluse le risorse è sufficiente utilizzare lo specifico tag nella pagina jsp. Tra le operazioni più usate vi è la taglib "form": l'aggiunta dello specifico tag nella pagina permette di gestire autonomamente le variabili del campo definito nella proprietà path. Oppure ancora, il taglib "core" mette a disposizione le operazioni condizionali.

L'interfaccia si sviluppa in due sezioni distinte: la parte pubblica con le pagine: *index.jsp, login.jsp, registrazione.jsp, ricerca2.jsp* e *dettagliFoglio.jsp*. Mentre la parte privata è posta all'interno della cartella *private* così da poter gestire più agevolmente la configurazione di *Spring Security*. All'interno di *private* vi è, oltre che la pagina di presentazione *welcome.jsp*, un set di cartelle, una per ogni sezione. Infine, i file JavaScript del framework jQuery e delle api DataTables sono posti al di fuori della cartella *WEB-INF* in *js* nello stesso path della cartella *images* destinata a contenere le immagini caricate dagli utenti.

All'interno delle pagine che effettuano operazioni di modifica dei dati sono poste particolari divisioni ( ) usate per segnalare errori nella procedura o il successo di quest'ultima. Questi messaggi appaiono ogni qual volta viene trasmesso dal server l'attributo *error* o *success*.

La ricerca fa uso di AJAX per ricevere e inviare i dati al server. Attraverso il metodo JavaScript *doAjaxPost()* viene richiamata la funzione *$.ajax()*. I dati vengono inviati tramite GET e intercettati dal controller *Ricerca2Controller*. Differentemente dagli altri metodi, per le comunicazioni AJAX si fa uso dell'annotazione *@ResponseBody* da associare all'oggetto di ritorno. In questo modo, una volta composto il dato, nel caso specifico una stringa contenente prima il numero di record ottenuti nella ricerca e successivamente il JSON con i dati dei fogli, il tutto ritorna alla pagina di ricerca nella quale, se non vi sono errori, viene gestito scorporando opportunamente le varie parti. In particolare il JSON viene estratto e assegnato come datasource alla tabella inizializzata dal metodo della libreria DataTables *dataTable()*.

Capitolo 5

Conclusioni

5.1 Obiettivi raggiunti e sviluppi futuri

Il software sviluppato soddisfa tutti i requisiti esposti nel capitolo 2, risulta funzionale ed è già stato integrato nel server cluster del Dipartimenti di Biologia dell'Università di Padova. Al momento sono presenti un migliaio di fogli di erbario e ben quattordici mila entità INHP. Le prime settimane di utilizzo hanno rivelato una stabilità e una velocità di funzionamento che rassicurano sulla qualità del codice.

Nonostante i committenti del progetto si siano dimostrati soddisfatti, i miglioramenti apportabili sono comunque numerosi. Tra le proposte di sviluppo vi è l'attuazione di un sistema di *instant search*, simile a quello di Google, da integrare sia nella ricerca che per suggerire all'utente come compilare i campi negli inserimenti e modifiche dei fogli di erbario. Inoltre è stata proposta l'integrazione di una diversa modalità di ricerca dando all'utente la possibilità di scegliere l'insieme di campi su cui effettuare la richiesta e le condizioni logiche da applicare in modo da ottenere query più mirate.

L'attuale versione del programma è stata presentata in via non ufficiale nel 108° Congresso della Società Botanica Italiana con il tema: "*Il ruolo delle collezioni storiche nella ricerca tassonomica moderna: catalogazione e studio dell'Herbarium Dalmaticum di Roberto de Visiani*" [4] il 20 Settembre a Baselga di Pinè. Inoltre il lavoro sarà oggetto di una pubblicazione scientifica nella rivista l'*Informatore Botanico Italiano*. La presentazione ufficiale e l'avvio di un programma di digitalizzazione da parte del Dipartimenti di Biologia dell'Università di Padova è ancora da definirsi ma si pensa sia comunque nella parte finale del quarto trimestre 2013.

Appendici

Appendice A

Codice

Di seguito vengono presentati alcuni importanti listati di codice utilizzati nel progetto.

A.1 Implementazione di Spring Security

Gestire l'accesso ad aree private con Spring Security si è rivelato facile e intuitivo . Dopo l'integrazione del modulo nel progetto, è bastato creare il file spring-security.xml per configurare internamente ogni singola azione di autenticazione. Di seguito viene presentato uno spezzone di codice per l'intercettazione degli URL e l'accesso al database per la convalida di username e password:

Listing A.1: *Configurazione del modulo Spring Security*

```
<http>
      <!--Intercettazione degli URL e attribuzione dell'authority-->
  <intercept-url pattern='/private/inhp/*' access='ROLE_ADMIN, ROLE_SUPERUSER' />
  <intercept-url pattern='/private/utente/*' access='ROLE_ADMIN'/>
  <intercept-url pattern='/private*' access='ROLE_USER, ROLE_ADMIN, ROLE_SUPERUSER' />
            .
            .
            .
      <!--Configurazione del login-->
  <form-login login-page='/login' default-target-url='/private/welcome'
         always-use-default-target='true' />
  <access-denied-handler error-page="/accessdenied" />
  <session-management>
     <concurrency-control max-sessions="1" error-if-maximum-exceeded="false" />
  </session-management>
</http>

      <!--Richiesta del datasource definito precedentemente e query di autenticazione-->
<authentication-manager>
  <authentication-provider>
    <jdbc-user-service data-source-ref="dataSource"
                    users-by-username-query="
                             select username,password, enabled
                             from users where USERNAME=?"
                             authorities-by-username-query="
                             select u.username, ur.authority from users u, user_roles ur
                             where u.user_id = ur.user_id and u.username =? "
```

```
   />
  </authentication-provider>
</authentication-manager>
```

A.2 Implementazione di Ajax

La comunicazione Ajax viene utilizzata nella pagina di ricerca per dare un senso di fluidità e immediatezza. Di seguito vengono presentate alcune delle righe di codice che permettono questo tipo di connessione:

- lato server

```
//Ricezione richiesta di una nuova ricerca: la risposta e' un JSON inoltrato con @ResponseBody
@RequestMapping(value = "cerca2", produces = "text/plain; charset=UTF-8" )
public @ResponseBody String getCerca(@ModelAttribute("Ricerca") @Valid Ricerca ricerca,
                                     BindingResult result, HttpServletResponse response) {
   response.setCharacterEncoding("UTF-8");

   if (result.hasErrors()) {
      System.out.println("error=" + result.getAllErrors().get(0).getDefaultMessage());
      return "error=" + result.getAllErrors().get(0).getDefaultMessage();
   }
   RicercaManager ricercaManager = new SimpleRicercaManager(risultatoRicercaDao);

   ricerca.setRisultati(ricercaManager.ricerca(ricerca));

   //Converto il risultato della ricerca in un JSON
   return "numeroRisultati=" + ricerca.getNumeroRisultati() + "," +
                        parseJSON(ricerca);
}
```

- lato client

```
$.ajax({
  <!--Inoltro la richiesta al server con i parametri di ricerca: uso il metodo GET
                          cosi' l'utente puo' copiare o salvare la ricerca-->
  type: "GET",
  url: "./cerca2",
  data: "risultatiPerPag=" + risultatiPerPag + "&numeroPagina=
          " + pagina + "&orderBy=" + orderBy + "&testoRicerca=" + testoRicerca,
  success: function(response) {
        <!--Il server ha risposto correttamente-->

        if (response.substring(0, 5)===("error"))
        {
          document.getElementById("errorblock").innerHTML=
                              'Errore: '+ response.substring(6, response.lenght);
        }
        else{
          document.getElementById("errorblock").innerHTML='';
          <!--Estrapolazione del numero dei risultati-->
          numRisultati='/';
          if(response.substring(0, 15)===("numeroRisultati"))
          {
             numRisultati=response.substring(16, response.indexOf(','));
          }
          document.getElementById("numeroRisultati").innerHTML=
                                      'Trovati: '+numRisultati+' risultati';
         <!--Aggiorno la tabella con i nuovi dati: i dati sono in formato JSON-->
```

```
                updateDataTable(response.substring(response.indexOf(",")+1,response.lenght));
              }
        },
        error: function(e) {
              <!--Errore nella risposta del server-->
              alert('Error: ' + e);
              }
  });
```

A.3 Costruzione di un oggetto DAO e interfacciamento con la business logic

Gli oggetti DAO vengono inizializzati, 'legati' ai controller da Spring e successivamente utilizzati per definire i manager. In questo modo il collegamento dei componenti del programma al database rimane mascherato, semplificando così sia la parte di programmazione che i cambi di piattaforme.

La configurazione di Spring per definire i componenti DAO è la seguente:

```
<!-- Session Factory da utilizzare per mapping attraverso JPA Annotations-->
    <bean id="sessionFactory"
    class="org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean">
          <property name="dataSource" ref="dataSource" />
          <property name="annotatedClasses">
                <list>
                      <value>erbarioOnline.Domain.Per</value>
                      <value>erbarioOnline.Domain.Det</value>
                                  .
                                  .
                                  .
                </list>
          </property>
          <property name="hibernateProperties">
                <props>
                     <prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
                <prop key="hibernate.connection.driver_class">com.mysql.jdbc.Driver</prop>
                <prop key="hibernate.connection.url">jdbc:mysql://.../erbariopatavino</prop>
                </props>
          </property>
    </bean>

    <bean id="hibernateTemplate" class="org.springframework.orm.hibernate3.HibernateTemplate">
          <property name="sessionFactory" ref="sessionFactory" />
    </bean>

<!-- Beans DAO -->
    <bean name="perDao" class="erbarioOnline.Domain.PerHibernateDaoSupport">
          <property name="hibernateTemplate" ref="hibernateTemplate" />
    </bean>
    <bean name="detDao" class="erbarioOnline.Domain.DetHibernateDaoSupport">
          <property name="hibernateTemplate" ref="hibernateTemplate" />
    </bean>
```

Gli oggetti DAO sono tutti composti seguendo la medesima struttura: un'interfaccia che ne definisce le azioni e l'implementazione.

```
public interface DetDao {
      public void insert(Det det);
```

```
      public void delete(int id);
      public void update(Det det);
            .
            .
            .
}

public class DetHibernateDaoSupport extends HibernateDaoSupport implements DetDao{
      public void insert(Det det){
         getHibernateTemplate().saveOrUpdate(det);
      }
      public void delete(int id){
         Det det = getHibernateTemplate().get(Det.class, id);
         getHibernateTemplate().delete(det);
      }
                  .
                  .
                  .
```

Infine i controller richiamano gli oggetti DAO utilizzando l'annotazione *@Autowired*:

```
@Controller
public class FoglioErbarioController {

   @Autowired
   UsersDao usersDao;
   @Autowired
   PerDao perDao;
   @Autowired
   DetDao detDao;
            .
            .
            .
```

Appendice B

Descrizione delle tabelle

Il database dell'Erbario di Padova è costruito su sette tabelle fondamentali, con l'aggiunta di altre due con funzione strettamente tecnica per la registrazione degli utenti: Oggetti (*ogg*), Determinazioni (*det*), Pubblicazioni (*rifPub*), Collaboratori (*per*), Indice dei Nomi (*inhp*), Elenco dei Luoghi (*rifGeo*), Riferimenti (*rif*). I dati degli utenti vengono salvati in: Utenti (*users*) e Ruolo utente (*user_roles*).

Il formato di inserimento dei dati rispetta le seguenti regole generali:

- I dati inseriti sono riportati sempre e solo come si presentano sul cartellino originale. Tutte le eventuali indicazioni derivate da osservazioni del catalogatore e non presenti sul cartellino originale sono indicate tra parentesi quadre.

- I dati del cartellino sono riportati nella lingua originale in cui furono scritti.

- In caso alcune lettere sul cartellino originale risultino illeggibili, esse sono sostituite dal trattino basso (_).

- In caso un'intera parola sul cartellino originale risulti leggibile, ma non si sia sicuri di averla interpretata correttamente, è fatta seguire immediatamente dal simbolo [1].

- In caso più parole o intere frasi non risultino leggibili, si usa una nota del catalogatore tra parentesi quadre, ad esempio [località illeggibile].

- I dati distribuiti tra più cartellini o tra diverse aggiunte sullo stesso cartellino, sono indicati, nel presunto ordine cronologico, separati dal simbolo ; , che non è mai utilizzato con un altro significato (eventualmente sostituito con altra punteggiatura).

- I nomi scientifici in cui compaiono diciture quali mihi o nobis, sono riportati con l'abbreviazione standard dell'Autore, secondo il database internazionale IPNI, al posto della dicitura stessa.

B.1 Tabella Oggetti (ogg)

La tabella Oggetti comprende i dati collegati alla raccolta, preparazione, conservazione e catalogazione del campione.

CB:	Il codice a barre del campione, assegnato secondo le linee guida dell' Herbarium Patavinum fa da chiave primaria.
Coll:	La collezione in cui è attualmente inserito il campione, ad esempio HV per Herbarium Venetum.
Tipol:	La tipologia di campione. I valori possibili sono: • fh: Foglio di erbario • bs: Busta • fh, bs: Foglio di erbario + busta • vt: Preparato microscopico su vetrino
Cons:	Stato di conservazione. La scelta è tra: • b: Buono. Il campione, limitatamente alle parti rappresentate, è adatto ad essere studiato dal punto di vista morfologico e non presenta evidenti attacchi di parassiti o danni meccanici importanti • m: Medio. Il campione, limitatamente alle parti rappresentate, è riconoscibile, ma è danneggiato al punto da rendere difficile un'analisi morfologica. • c: Cattivo. Il campione è pressochè irriconoscibile e non utilizzabile per analisi morfologiche.
NCamp:	Numero di parti in cui è diviso il campione. Generalmente, si contano i singoli esemplari di piante di piccole dimensioni, oppure i frammenti in caso di piante di grandi dimensioni divise volontariamente in più parti, mentre non si considera suddiviso in più parti un campione spezzato accidentalmente. La scelta è tra: • 1, 2, 3, 4, 5 • n: si usa quando le parti sono più di cinque oppure è complicato determinare il numero di parti, ad esempio perch il campione è composto di più piante intricate tra loro.

Racc:	La raccolta originale in cui il raccoglitore ha inserito il campione. Generalmente, si trascrive in questo campo l'intestazione prestampata del primo cartellino associato al campione
Pos:	Indicazioni del catalogatore sul posizionamento attuale del campione, utili a reperirlo. Questo dato è gestito in modo differente a seconda delle diverse collezioni
LgDI:	La data in cui è stato raccolto il campione, oppure la data iniziale in caso sia stato indicato un intervallo di tempo. Il formato è rigorosamente solo AAAA-MM-GG, è possibile inserire il valore zero in caso alcune parti della data non siano note. Generalmente, non si inserisce il mese di raccolta se non è presente anche il giorno o l'anno, poich molti campioni di piante vascolari presentano, come annotazione fenologica, il periodo di fioritura della specie in quell'habitat, che può non coincidere con quello di raccolta del campione.
DigD:	La data e l'ora in cui è stato catalogato il campione. Il campo è automatico.
ImgD:	La data in cui è stata realizzata l'immagine del campione. Il formato è rigorosamente solo AAAA-MM-GG.
Prof:	Campo riservato a specie acquatiche; la profondità a cui è stato pescato il campione, in metri. Il valore va inserito come un numero positivo, ad esempio 3 significa a tre metri di profondità.
Hab:	L'habitat da cui proviene il campione, così come riportato sul cartellino. In questo campo vanno indicate anche le eventuali specie associate al campione indicate sul cartellino, a parte per le specie parassite o epifite, per le quali è previsto l'apposito campo Ospite in tabella Oggetti. Le note sul substrato vanno nell'apposto campo Subs in tabella Oggetti.
Subs:	Il substrato su cui è stato raccolto il campione. Per le specie parassite o epifite è previsto l'apposito campo Ospite.
NoteO:	Le altre note originali presenti sul cartellino. In caso di note particolarmente lunghe e/o di scarso rilievo, è consigliabile utilizzare indicazioni tipo [numerose note di tassonomia].
Note:	Le eventuali osservazioni personali del catalogatore. Contiene annotazioni riguardanti la catalogazione o altri approfondimenti sul dato e non è visibile nella versione online del database. Le informazioni potenzialmente utili al visitatore vanno riportate in altri campi, eventualmente tra parentesi quadre.
Ospite:	L'ospite di una specie parassita o epifita. Va riportato così com'è indicato sul cartellino.
Fen:	Eventuali note a riguardo le parti della pianta rappresentate nel campione.
Fos:	Campo impiegato solo per campioni microscopici, serve a distinguere il materiale fossile, cui si attribuisce valore 1 da quello attuale, a cui si attribuisce valore 0.

Img:	Chiave esterna alla tabella *per*. Rappresenta l'abbreviazione del collaboratore che ha realizzato l'eventuale immagine collegata al campione, proveniente dall'elenco in tabella Collaboratori.
ImgP:	Il nome dell'eventuale file dell'immagine che comprende il campione. Di regola, i file hanno per nome il codice a barre del campione, seguito dall'estensione, ad esempio .jpg. In caso di immagini che comprendono più campioni, di regola si usa come nome del file il codice a barre di valore più basso tra quelli rappresentati nell'immagine.
Nom:	Chiave esterna alla tabella *inhp*. L'entità tassonomica proveniente dall'elenco in tabella Indice dei Nomi a cui il catalogatore ritiene che il campione sia da ricondurre, sulla base della più affidabile determinazione in tabella Determinazioni (in genere, la più recente). In caso di dubbi, il catalogatore deve limitarsi a indicare entità di rango superiore oppure omettere di compilare questo campo.
Loc:	Chiave esterna alla tabella *rifGeo* per la gestione delle località
LgDF:	La data finale dell'intervallo di tempo in cui è stato raccolto il campione; la data iniziale va inserita nel campo LgDI in tabella Oggetti. Il formato è rigorosamente solo AAAA-MM-GG, è possibile inserire il valore zero in caso alcune parti della data non siano note.
Dig:	Chiave esterna alla tabella *per*. Rappresenta l'abbreviazione del nome del catalogatore. Il campo è automatico.
Lg:	Chiave esterna alla tabella *per*. Rappresenta l'abbreviazione del raccoglitore del campione, proveniente dall'elenco in tabella Collaboratori. Se il nome dello studioso può essere dedotto dal catalogatore, ad esempio sulla base della calligrafia, ma non è esplicitato, andrà inserito tra parentesi quadre.

Tabella B.1: *Tabella Oggetti (ogg)*

B.2 Tabella Determinazioni (det)

La sezione Determinazioni comprende l'elenco delle successive determinazioni e ridenominazioni attribuite al campione dagli studiosi e riportati sul cartellino o contenitore e i dati ad esse associati.

ID:	È la chiave primaria della tabella.

DetN:	Il numero della determinazione, partendo da 1. Nel caso più semplice e comune, ad ogni determinazione corrisponde un solo nome e, quindi, un solo DetN. È possibile, però, che ad una singola determinazione corrispondano più nomi, ad esempio perch lo studioso ha voluto indicare, oltre al nome che ritiene accettato, anche dei sinonimi. In questo caso, i sinonimi indicati avranno lo stesso DetN del nome principale, perch non rappresentano rideterminazioni o ridenominazioni distinte. Nel caso in cui un sinonimo sia però aggiunto in seguito ad una determinazione realizzata in precedenza, ad esempio qualora uno studioso segni sul cartellino un cambio nomenclaturale, il nuovo sinonimo avrà un nuovo DetN, in quanto rappresenta una ridenominazione distinta.
Des:	Chiave esterna alla tabella *per*. Rappresenta l'abbreviazione dello studioso che ha designato formalmente il campione come tipo del nome in questione, proveniente dall'elenco in tabella Collaboratori. La pubblicazione che formalizza la designazione va indicata in tabella Pubblicazioni.
DesD:	La data in cui è stato designato formalmente il campione come tipo del nome in questione. Il formato è rigorosamente solo AAAA-MM-GG, è possibile inserire il valore zero in caso alcune parti della data non siano note.
OrthEmend:	Il nome attribuito al campione, nella forma originale scorretta, ad esempio Ballote nigra. Qualora il nome indicato da uno studioso sia scritto in forma scorretta, questa va trascritta per intero all'interno di questo campo. Non è necessario fare uso di questo campo nei casi in cui la correzione sia limitata ad un passaggio tra lettere maiuscole e minuscole oppure ad una correzione delle abbreviazioni che indicano un livello sottospecifico.
Det:	Chiave esterna alla tabella *per*. Rappresenta l'abbreviazione dello studioso che ha realizzato la determinazione o ridenominazione, proveniente dall'elenco in tabella Collaboratori. Questo campo è compilato solo in caso il nome dello studioso sia indicato esplicitamente. In particolare, per la prima determinazione, se lo studioso ha semplicemente firmato il cartellino o ha segnato semplicemente legit (o scritte equivalenti), seguito dal proprio nome o abbreviazione, il suo nome va inserito soltanto nel campo Lg in tabella Oggetti. Va invece indicato anche qui in caso abbia segnato esplicitamente anche determinavit (o scritte equivalenti). Se il nome dello studioso può essere dedotto dal catalogatore, ad esempio sulla base della calligrafia, ma non è esplicitato, andrà inserito tra parentesi quadre.
DetD:	La data della determinazione o ridenominazione. Se sul cartellino vi è un'unica data non meglio specificata, si suppone riferita alla raccolta e andrà inserita nel campo LgDI in tabella Oggetti. Il formato è rigorosamente solo AAAA-MM-GG, è possibile inserire il valore zero in caso alcune parti della data non siano note.

Nom:	Il nome attribuito dallo studioso, scritto con l'ortografia esatta, per esteso e nella forma prescritta dal Codice di Nomenclatura. Alcuni delle correzioni più comuni: • il genere va scritto in maiuscolo, gli epiteti specifici e sottospecifici in minuscolo • vanno usate le abbreviazioni var., subsp., f., cv. per indicare i rispettivi livelli sottospecifici e non altre sigle o lettere (es. la numerazione in lettere greche per indicare le varietà) • le parti del nome devono concordare grammaticalmente, salvo errori codificati dalla tradizione (es. Ranunculus acris) • negli epiteti composti va sempre usata la vocale di collegamento i (es. Stachys menthifolia, non Stachys mentaefolia) • negli epiteti derivanti dal genitivo di una latinizzazione del nome di uno studioso, vanno sempre usate le desinenze -ii per il maschile o -iae per il femminile (es. Pteris alpinii, non Pteris alpini) Le abbreviazioni s.s. e s.l. (e analoghe) sono considerate parte del nome e vanno indicate dopo l'eventuale abbreviazione dell'Autore, mentre sono considerate note indicazioni quali sp. nov. oppure var. ?. In caso lo studioso indichi che ha un dubbio sul nome indicando, ad esempio ? oppure cfr. var. grandiflora, il nome va indicato qui come se fosse dato per certo, mentre la parte che indica il dubbio dello studioso è considerata una nota e va inserita nel campo NoteT in tabella Determinazioni. Le correzioni sono da effettuare in ogni caso, anche se ci sono riferimenti bibliografici a opere che riportano il nome errato (es. Ballota nigra, non Ballote nigra, anche se tale nome è riportato in questa forma in Fl. Dalmat.). Se è indicato l'Autore del nome, questo va riportato rispettando fedelmente l'abbreviazione proposta dallo studioso che ha determinato il campione e, se manca, non va aggiunta, nemmeno in caso sia in qualche modo possibile risalire all'Autore a cui fa riferimento lo studioso. Se il nome abbreviato comprende più parti, queste non vanno separate da spazi divisibili, anche se tale spazio è parte integrante del nome dell'Autore (es. K.Maly non K. Maly, deVries, non de Vries. Tale accorgimento serve a semplificare eventuali future rielaborazioni. È sconsigliato ma ammissibile l'uso di spazi indivisibili o del trattino basso.

T:	Il tipo di determinazione o ridenominazione. I valori possibili sono tre: • determinazione: è il caso più comune, si impiega per indicare che lo studioso intende il nome come una nuova determinazione del campione. • sinonimo: si impiega per indicare che lo studioso intende il nome come un sinonimo di uno attribuito al campione contemporaneamente (stesso DetN) o precedentemente (DetN inferiore) al campione. • contenitore: si impiega nei casi in cui al campione non siano stati attribuiti direttamente dei nomi, ma lo studioso che ha organizzato la collezione abbia indirettamente fornito questa informazione, ad esempio inserendolo in una cartella a cui ha attribuito un nome o comunque in un contesto di altro materiale determinato.
Typ:	Assume uno dei seguenti valori, in caso di campioni ritenuti tipi del nome in questione: • possibile tipo • tipo • olotipo • ...
NoteT:	Comprende tutte le eventuali note, sia originali sia del catalogatore, collegate al nome, tra cui: • osservazioni dello studioso riconducibili al nome o al suo lavoro di rideterminazione o ridenominazione • indicazioni originali sullo stato del nome, quali nom. cons., sp. nov. e simili • indicazioni di dubbi sulla determinazione, quali ? o cfr. grandiflora. • eventuali citazioni bibliografiche. Le note del catalogatore vanno inserite tra parentesi quadre. In caso il nome sia leggibile, ma sia stato cancellato, andrà aggiunto [cancellato] in questo campo.

Tabella B.2: *Tabella Determinazioni (det)*

B.3 Tabella Pubblicazioni (rifPub)

La tabella Pubblicazioni comprende i dati a riguardo le pubblicazioni collegate ad un campione. Le pubblicazioni vanno preferibilmente citate secondo la struttura e con le abbreviazioni proposta dal sito www.ipni.org .

ID:	Chiave primaria della tabella.
rifCB:	Riferimento al codice a barre del foglio di erbario corrispondente.
pub:	Campo riservato alla pubblicazione.

Tabella B.3: *Tabella Pubblicazioni (rifPub)*

B.4 Tabella Collaboratori (per)

La tabella Collaboratori comprende le abbreviazioni e i nomi di tutti i collaboratori che possono comparire nelle schede del database: raccoglitori, studiosi, catalogatori e chi contribuisce con l'acquisizione di immagini o in qualunque altro modo.

ID:	Chiave primaria della tabella.
Abb:	L'abbreviazione del collaboratore. Questo valore è quello che sarà visualizzato nei campi in cui va introdotto il nome di un collaboratore. Normalmente, l'abbreviazione si compone di Cognome + Iniziale del nome di battesimo del collaboratore, ad esempio Bguinot A. per Augusto Bguinot. Per i gruppi di più di una persona, essi vanno inseriti nell'ordine e numero in cui sono riportati sul cartellino, ma abbreviati come nel caso precedente e separati da & se sono due oppure da ,.
N:	Note. Normalmente questo campo comprende il nome di battesimo completo del collaboratore. Per i gruppi di più di una persona, il campo è in genere vuoto.

Tabella B.4: *Tabella Collaboratori (per)*

B.5 Tabella Indice dei Nomi (inhp)

La tabella Indice dei Nomi (INHP, indice dei nomi dell' Herbarium Patavinum) comprende un elenco coerente e aggiornato di nomi accettati nel database. Questo elenco si basa su checklist nazionali ed internazionali ed è mantenuto aggiornato da parte degli amministratori del sistema.

Ciascun campione, quando possibile, è associato dal catalogatore ad un nome in questa tabella, cosicch si rendono possibili eventuali rielaborazioni basate su questo tipo di informazione, per le quali è però necessaria una richiesta all'amministratore del sistema. Sarà inoltre possibile ampliare a seconda delle necessità i dati associati a ciascuna entità tassonomica. Con l'aggiornamento di INHP, i dati dei campioni associati a questi nomi (nella tabella Oggetti) sono anch'essi aggiornati in modo massivo e semiautomatico.

ID:	Chiave primaria della tabella
Nom:	Il nome completo dell'entità tassonomica, così com'è visualizzato nel campo Nom della tabella Oggetti. Comprende la concatenazione dei sei campi successivi (Gen, Sp, Aut, SspL, Ssp, ASsp).
Gen:	Il genere dei nomi ammessi in INHP.
Sp:	L'epiteto specifico dei nomi ammessi in INHP. Inoltre, a ciascun genere, è associata anche l'indicazione sp., per consentire di indicare un campione la cui determinazione è esatta o possibile solo fino al rango di genere.
Aut:	L'Autore del nome specifico in questione, abbreviato secondo le sigle proposte su www.ipni.org , ma rimossi gli spazi secondo il criterio già utilizzato nella tabella Determinazioni, per semplificare eventuali future rielaborazioni.
SspL:	L'abbreviazione del livello infraspecifico, a scelta tra var. (varietà), subsp. (sottospecie), f. (forma), cv. (cultivar). Un solo livello infraspecifico è ammesso in INHP. È disponibile anche l'abbreviazione s.l. (sensu lato), utilizzata -impropriamente, come già nella Checklist della Flora d'Italia- ad indicare un campione la cui determinazione è esatta o possibile solo fino al rango di specie.
Ssp:	L'epiteto infraspecifico dei nomi ammessi in INHP.
ASsp:	L'Autore del nome infraspecifico, trattato come nel campo Aut della tabella Indice dei Nomi.
Fam:	La famiglia che comprende il nome in questione, secondo il sistema adottato in INHP (APGIII, per le Angiosperme). Per le otto famiglie per le quali il Codice di Nomenclatura consente la scelta tra un nome descrittivo ed uno tipizzato, è impiegato il secondo (ad esempio si usa Asteraceae e non Compositae).
Bib:	Una breve indicazione della fonte bibliografica da cui proviene il nome, utile all'amministrazione del sistema. L'elenco completo dei riferimenti bibliografici sono disponibili altrove agli utilizzatori del sistema.
T:	Un'abbreviazione che indica il tipo di organismo in questione, utile all'amministrazione del sistema.

Tabella B.5: *Tabella Indice dei Nomi (inhp)*

B.6 Tabella Elenco dei Luoghi (rifGeo)

La tabella Elenco dei Luoghi raccogli tutti i nomi dei luoghi specificati nel cartellino

ID:	Chiave primaria della tabella.
Cartellino:	La località di provenienza del campione. In questo campo vanno inseriti tutti i dati originali utili a caratterizzare il campione dal punto di vista geografico, che saranno in futuro impiegati per la georeferenziazione.

Tabella B.6: *Tabella Elenco dei Luoghi (rifGeo)*

B.7 Tabella dei Riferimenti (rif)

I riferimenti tra fogli di erbari sono gestiti dalla tabella Riferimenti.

ID:	Chiave primaria della tabella.
cb_rif:	Chiave esterna alla tabella Ogg. Questo campo svolge la funzione di identificare il foglio che tiene il riferimento.
rif:	Chiave esterna verso la tabella Ogg. Questo campo svolge la funzione di identificare verso che foglio è il riferimento.

Tabella B.7: *Tabella Riferimenti (rif)*

Bibliografia

[1] Paul Du Bois. *MySQL*. Pearson, 2004.

[2] Forman L. Bridson D. *The herbarium handbook*. Royal Botanic Gardens Kew, 1993.

[3] Gavin King Christian Bauer. *Hibernate in Action: Practical Object/Relational Mapping*. Manning Publications; 1 edition, 2004.

[4] Menegazzo Carlo Clementi Moreno, Miola Antonella. Il ruolo delle collezioni storiche nella ricerca tassonomica moderna: catalogazione e studio dell'herbarium dalmaticum di roberto de visiani. In 108° *convegno della societ botanica italiana*, 2013.

[5] Apache Software Foundation. Apache tomcat. `http://tomcat.apache.org`.

[6] David M. Geary. *Core Jstl: Mastering the Jsp Standard Tag Library*. Prentice Hall Ptr, 2002.

[7] GoPivotal. Spring. `http://spring.io/`.

[8] JBoss. Hibernate. `http://www.hibernate.org/`.

[9] Karl Swedberg Jonathan Chaffer. *Learning jQuery Fourth Edition*. Packt Publishing, 2013.

[10] The jQuery Foundation. jquery. `http://jquery.com`.

[11] Antonella Miola Moreno Clementi. Proposte per l ammodernamento dell'herbarium patavinum. Master's thesis, Università degli Studi di Padova, 2011.

[12] Srinivas Mudunuri. *Spring Framework: A Step by Step Approach for Learning Spring Framework*. CreateSpace Independent Publishing Platform, 2013.

[13] Joe Fawcett Nicholas C. Zakas, Jeremy McPeak. *Professional Ajax*. Wrox, 2009.

[14] SpryMedi. Datatables. `http://datatables.net/`.

[15] Sergio Tornadore, Noemi; Chiesa. *Erbario e collezioni botaniche del Centro interdipartimentale di servizi Musei Scientifici*. Padova : La Garangola, 1991.

[16] Jeff Genender Vivek Chopra, Sing Li. *Professional Apache Tomcat 6*. Wrox, 2007.

[17] Alexei White. *JavaScript Programmer's Reference*. Wrox, 2012.