METODOLOGIA DI SVILUPPO DELLA PIATTAFORMA DOCSOURCES

|  |
| --- |
| Developed by Medici Archive Project (2010)  Permission is granted to copy, distribute and/or modify this document  under the terms of the GNU Free Documentation License, Version 1.3  or any later version published by the Free Software Foundation;  with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.  A copy of the license is included in the section entitled "GNU  Free Documentation License". |

|  |  |  |  |
| --- | --- | --- | --- |
| **Redatto da:** | (firma) | Data | 08/01/10 |
|  | Lorenzo Pasquinelli, Senior Developer  (nome, cognome, qualifica) | | |
|  | | | |
| **Rivisto da:** | (firma) | Data |  |
|  | (nome, cognome, qualifica) | | |
|  | | | |
| **Approvato da:** | (firma) | Data |  |
|  | (nome, cognome, qualifica) | | |
|  | | | |
|  | | | |
|  |  |  |  |
|  |  | | |
|  | | | |
| **Data**: | 08/01/2010 | | |
|  |  | | |
| Il documento è da ritenersi "IN LAVORAZIONE" se provvisto della sola firma: *Redatto da*  Il documento è da ritenersi "VERIFICATO ED EMESSO IN BOZZA" se provvisto anche della firma: *Rivisto da*  Il documento è da ritenersi "DISTRIBUIBILE" se provvisto anche della firma: *Approvato da* | | | |

Sommario

[1 Storia dei cambiamenti 7](#_Toc273913160)

[2 Lista di distribuzione 7](#_Toc273913161)

[3 Scopo del documento 7](#_Toc273913162)

[4 Licenza d’uso 8](#_Toc273913163)

[4.1 GNU Free Documentation License 8](#_Toc273913164)

[4.1.1 0. PREAMBLE 8](#_Toc273913165)

[4.1.2 1. APPLICABILITY AND DEFINITIONS 8](#_Toc273913166)

[4.1.3 2. VERBATIM COPYING 10](#_Toc273913167)

[4.1.4 3. COPYING IN QUANTITY 10](#_Toc273913168)

[4.1.5 4. MODIFICATIONS 10](#_Toc273913169)

[4.1.6 5. COMBINING DOCUMENTS 12](#_Toc273913170)

[4.1.7 6. COLLECTIONS OF DOCUMENTS 12](#_Toc273913171)

[4.1.8 7. AGGREGATION WITH INDEPENDENT WORKS 12](#_Toc273913172)

[4.1.9 8. TRANSLATION 13](#_Toc273913173)

[4.1.10 9. TERMINATION 13](#_Toc273913174)

[4.1.11 10. FUTURE REVISIONS OF THIS LICENSE 13](#_Toc273913175)

[4.1.12 11. RELICENSING 14](#_Toc273913176)

[5 Modello architetturale 15](#_Toc273913177)

[6 Moduli applicativi 18](#_Toc273913178)

[7 Modello di sicurezza applicativa 19](#_Toc273913179)

[8 Modello di navigazione utente 30](#_Toc273913180)

[9 Il processo di validazione dei dati 37](#_Toc273913181)

[10 Frontend applicativo 37](#_Toc273913182)

[11 Configurazione applicativa 38](#_Toc273913183)

[12 Licenza d’uso dell’applicazione 42](#_Toc273913184)

[13 Struttura del progetto 43](#_Toc273913185)

[13.1 File pom.xml 43](#_Toc273913186)

[13.2 Directory src 44](#_Toc273913187)

[13.3 Descrittori JPA 44](#_Toc273913188)

[13.4 Directory src/main/java/org/ 45](#_Toc273913189)

[13.4.1 org.medici.docsources.audit 46](#_Toc273913190)

[13.4.2 org.medici.docsources.command 47](#_Toc273913191)

[13.4.3 org.medici.docsources.command.docbase 51](#_Toc273913192)

[13.4.4 org.medici.docsources.command.geobase 51](#_Toc273913193)

[13.4.5 org.medici.docsources.command.login 51](#_Toc273913194)

[13.4.6 org.medici.docsources.command.peoplebase 51](#_Toc273913195)

[13.4.7 org.medici.docsources.command.user 51](#_Toc273913196)

[13.4.8 org.medici.docsources.command.volbase 52](#_Toc273913197)

[13.4.9 org.medici.docsources.common.ajax 52](#_Toc273913198)

[13.4.10 org.medici.docsources.common.util 52](#_Toc273913199)

[13.4.11 org.medici.docsources.controller 52](#_Toc273913200)

[13.4.12 org.medici.docsources.controller.docbase 54](#_Toc273913201)

[13.4.13 org.medici.docsources.controller.geobase 54](#_Toc273913202)

[13.4.14 org.medici.docsources.controller.peoplebase 54](#_Toc273913203)

[13.4.15 org.medici.docsources.controller.user 54](#_Toc273913204)

[13.4.16 org.medici.docsources.controller.volbase 55](#_Toc273913205)

[13.4.17 org.medici.docsources.dao 55](#_Toc273913206)

[13.4.18 org.medici.docsources.dao.altname 61](#_Toc273913207)

[13.4.19 org.medici.docsources.dao.bibliot 61](#_Toc273913208)

[13.4.20 org.medici.docsources.dao.bioreflink 61](#_Toc273913209)

[13.4.21 org.medici.docsources.dao.country 62](#_Toc273913210)

[13.4.22 org.medici.docsources.dao.document 62](#_Toc273913211)

[13.4.23 org.medici.docsources.dao.eplink 62](#_Toc273913212)

[13.4.24 org.medici.docsources.dao.epltolink 62](#_Toc273913213)

[13.4.25 org.medici.docsources.dao.factchecks 62](#_Toc273913214)

[13.4.26 org.medici.docsources.dao.historylog 62](#_Toc273913215)

[13.4.27 org.medici.docsources.dao.marriage 62](#_Toc273913216)

[13.4.28 org.medici.docsources.dao.month 62](#_Toc273913217)

[13.4.29 org.medici.docsources.dao.peole 63](#_Toc273913218)

[13.4.30 org.medici.docsources.dao.place 63](#_Toc273913219)

[13.4.31 org.medici.docsources.dao.polink 63](#_Toc273913220)

[13.4.32 org.medici.docsources.dao.prclink 63](#_Toc273913221)

[13.4.33 org.medici.docsources.dao.researcher 63](#_Toc273913222)

[13.4.34 org.medici.docsources.dao.rolecat 63](#_Toc273913223)

[13.4.35 org.medici.docsources.dao.serieslist 63](#_Toc273913224)

[13.4.36 org.medici.docsources.dao.synextract 63](#_Toc273913225)

[13.4.37 org.medici.docsources.dao.titleocclist 64](#_Toc273913226)

[13.4.38 org.medici.docsources.dao.topicslist 64](#_Toc273913227)

[13.4.39 org.medici.docsources.dao.user 64](#_Toc273913228)

[13.4.40 org.medici.docsources.dao.volume 64](#_Toc273913229)

[13.4.41 org.medici.docsources.domain 64](#_Toc273913230)

[13.4.42 org.medici.docsources.dto 64](#_Toc273913231)

[13.4.43 org.medici.docsources.exception 64](#_Toc273913232)

[13.4.44 org.medici.docsources.security 64](#_Toc273913233)

[13.4.45 org.medici.docsources.service 65](#_Toc273913234)

[13.4.46 org.medici.docsources.service.docbase 65](#_Toc273913235)

[13.4.47 org.medici.docsources.service.geobase 65](#_Toc273913236)

[13.4.48 org.medici.docsources.service.historylog 65](#_Toc273913237)

[13.4.49 org.medici.docsources.service.mail 65](#_Toc273913238)

[13.4.50 org.medici.docsources.service.peoplebase 65](#_Toc273913239)

[13.4.51 org.medici.docsources.service.recaptcha 65](#_Toc273913240)

[13.4.52 org.medici.docsources.service.user 65](#_Toc273913241)

[13.4.53 org.medici.docsources.service.volbase 65](#_Toc273913242)

[13.4.54 org.medici.docsources.support 66](#_Toc273913243)

[13.4.55 org.medici.docsources.support.recaptcha 66](#_Toc273913244)

[13.4.56 org.medici.docsources.validator 66](#_Toc273913245)

[13.4.57 org.medici.docsources.validator.docbase 66](#_Toc273913246)

[13.4.58 org.medici.docsources.validator.geobase 66](#_Toc273913247)

[13.4.59 org.medici.docsources.validator.peoplebase 66](#_Toc273913248)

[13.4.60 org.medici.docsources.validator.user 67](#_Toc273913249)

[13.4.61 org.medici.docsources.validator.volbase 67](#_Toc273913250)

[13.5 Directory src/test/java 67](#_Toc273913251)

[13.6 Directory src/test/resources 67](#_Toc273913252)

[13.7 Directory src/main/resources 67](#_Toc273913253)

[13.8 Directory WebContent 67](#_Toc273913254)

[13.8.1 WebContent/images/ 67](#_Toc273913255)

[13.8.2 WebContent /META-INF/ 67](#_Toc273913256)

[13.8.3 WebContent/scripts/ 68](#_Toc273913257)

[13.8.4 WebContent/styles/ 68](#_Toc273913258)

[13.8.5 WebContent/WEB-INF/ 68](#_Toc273913259)

[13.8.6 WebContent/WEB-INF/config/ 68](#_Toc273913260)

[13.8.7 WebContent/WEB-INF/jsp/ 68](#_Toc273913261)

[13.8.8 WebContent/WEB-INF/lib/ 68](#_Toc273913262)

[13.8.9 WebContent/WEB-INF/templates/ 68](#_Toc273913263)

[14 Creazione del database a runtime 68](#_Toc273913264)

[15 Data Entry Module 69](#_Toc273913265)

[15.1 VolBase module 69](#_Toc273913266)

[15.1.1 Creazione di un nuovo volume 69](#_Toc273913267)

[16 Bibliografia di riferimento 72](#_Toc273913268)

# Storia dei cambiamenti

|  |  |
| --- | --- |
| **DATA** | **MOTIVO DEL CAMBIAMENTO** |
| 08/01/10 – 09/18/10 | Stesura Iniziale a carico di Pasquinelli Lorenzo |
| 09/19/10 | Stesura appendice per Web Designer |
|  |  |

# Lista di distribuzione

|  |  |
| --- | --- |
| **Nominativo** | **Qualifica / Unità organizzativa** |
| Lorenzo Allori | Technology Director |
| Lorenzo Pasquinelli | Senior Developer |

# Scopo del documento

Scopo del presente documento è descrivere le specifiche di sviluppo relativamente al progetto in oggetto.

Vengono descritti l’architettura scelta, i moduli, le interfacce, le mappe e la base dati.

Il documento è stato redatto in una forma estremamente dettagliata al fine velocizzare al massimo l’inserimento delle persone nel team di sviluppo.

Il presente documento si integra alla proposta di software redesign “The Medici Granducal Archive: Documentary Sources For Arts and Humanities. Software Project Planning Redesign” visionata dal consiglio di amministrazione della Andrew W. Mellon Foundation ed approvata in data 18/06/2010.

Le specifiche di progetto possono essere visionate nel documento di appendire alla proposta, “The Medici Granducal Archive: Documentary Sources For Arts and Humanities. Software Project Planning Redesign. Phase 2: Top Level Software Design Document”.

# Licenza d’uso

Il presente documento viene rilasciato sotto licenza GNU Free Documentation License versione 1.3

## GNU Free Documentation License

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc. <<http://fsf.org/>>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

### 0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

### 1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

### 2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

### 3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

### 4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

* A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
* B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
* C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
* D. Preserve all the copyright notices of the Document.
* E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
* F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
* G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
* H. Include an unaltered copy of this License.
* I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
* J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
* K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
* L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
* M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
* N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
* O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

### 5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

### 6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

### 7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

### 8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

### 9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

### 10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

### 11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

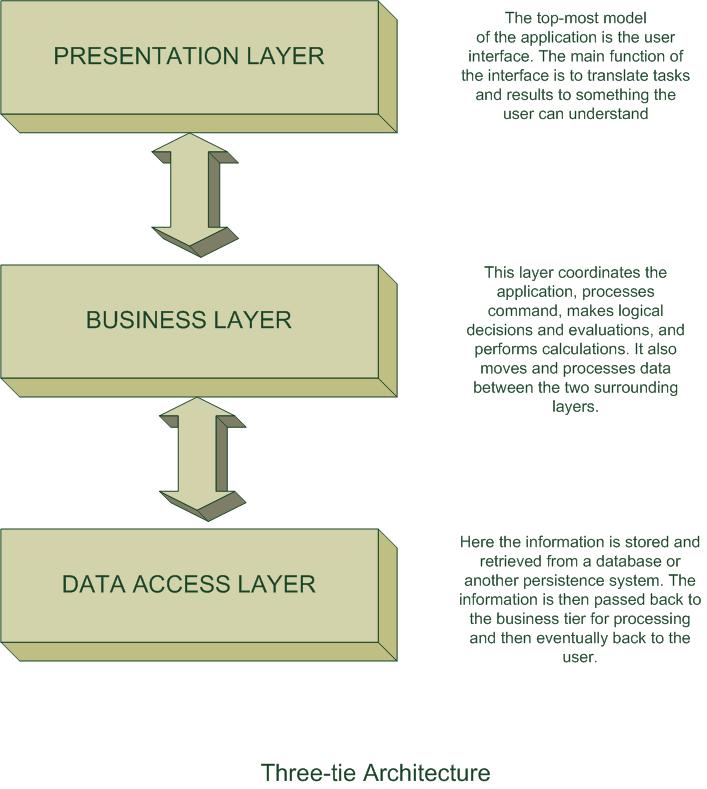
"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

# Modello architetturale

L’applicazione DocSource si basa quindi su di un’architettura a tre livelli come rappresentati in figura 1.



I livelli presentati in modo top-down sono:

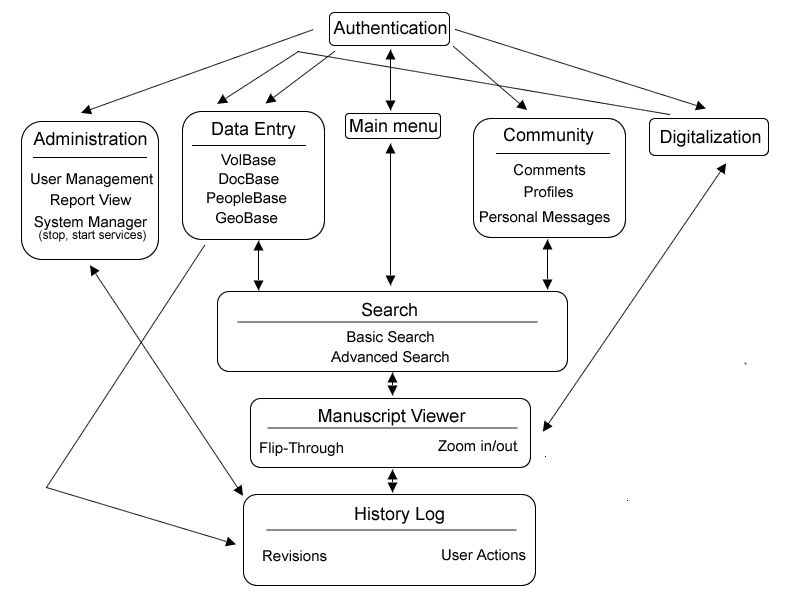
* Livello presentation: ha il compito di generare le pagine web, includendo informazioni
* Livello business: contiene la logica che interpreta ed elabora i dati passati dal client. E’ responsabile dell'implementazione delle regole di business.
* Livello data access: questo livello riunisce le classi responsabili della memorizzazione dei dati e il loro recupero

Avere una suddivisione a livelli come quella vista, in cui a ogni livello è demandato uno specifico compito, è alla base di tutte le moderne tecnologie e permette di gestire l'intera applicazione in modo più semplice per l'intero team di sviluppo.

I requisiti che ogni tecnologia orientata allo sviluppo di applicazioni web deve avere sono:

* elevate prestazioni: i server che realizzano i livelli presentazione e applicazione devono innanzitutto garantire prestazioni elevate, in grado di supportare senza difficoltà un gran numero di richieste provenienti dal livello client. Per realizzare ciò, essi devono offrire le seguenti funzionalità:
  + ambiente multithread: questa caratteristica permette di gestire correttamente accessi concorrenti alle risorse del sistema senza avere un decadimento delle prestazioni;
  + pooling: tecnica che permette la gestione comune delle risorse da parte di processi che elaborano richieste provenienti da client differenti; questa tecnica consiste nell'istanziare un certo numero di risorse già pronte per essere utilizzate all'arrivo di una richiesta di un client. Quando questa è eseguita la risorsa non viene distrutta ma ritorna a essere immediatamente disponibile;
  + gestione dinamica del ciclo di vita degli oggetti.
* Meccanismo di comunicazione tra i processi che vengono eseguiti sul server e che partecipano all'elaborazione di una richiesta del client.
* Bilanciamento del carico: spesso sono presenti piu server per l'elaborazione delle informazioni. Per questo motivo, la tecnologia usata in questo livello deve essere in grado di suddividere il lavoro in maniera equa tra i vari server, al fine di ottenere un servizio migliore e non sovraccaricare eccessivamente un solo componente
* Clustering: ovvero replicazione dello stato del server. Questo requisito può fare la differenza, in quanto permette di gestire i maniera trasparente per gli altri livelli eventuali malfunzionamenti o improvvisi crash di uno dei server.
* Supporto per le transazioni: la possibilità di evitare al programmatore la gestione delle transazioni affidandola invece al sistema. Questo rappresenta uno degli aspetti più interessanti e importanti richiesti da un'applicazione web.
* Gestione di un sistema semplice: la facilita di utilizzo e di configurazione deve essere ottenuta attraverso :
  + riconfigurazione e aggiornamento dinamico al momento dell'introduzione di nuove risorse;
  + spegnimento “safe”, in grado di salvare correttamente lo stato e le informazioni di tutti i processi eseguiti sul server;
  + registrazione (log) e revisione (audit) degli eventi.
* Bassi costi di sviluppo. In particolare devono essere soddisfatti i seguenti aspetti:
  + facilità di integrazione con i sistemi informativi pre-esistenti, per evitare la progettazione ex-novo dell'intero sistema;
  + generazione automatica del codice per la gestione degli aspetti comuni a ogni componente;
  + riuso dei componenti.
* Sicurezza: a causa dell'elevato rischio cui sono soggette le informazioni che transitano sulla rete, i livelli presentazione e applicazione devono fornire strumenti per l'autenticazione e l'autorizzazione all'uso dello risorse del sistema.
* Servizi di gestione dei nomi e directory: permettono di svincolare le richieste del client dall'effettiva locazione fisica delle risorse. Il client richiede un servizio utilizzando un nome logico, mentre e il server che si preoccupa di individuare la collocazione delle risorse, attivarla ed eseguire la richiesta per conto del client.

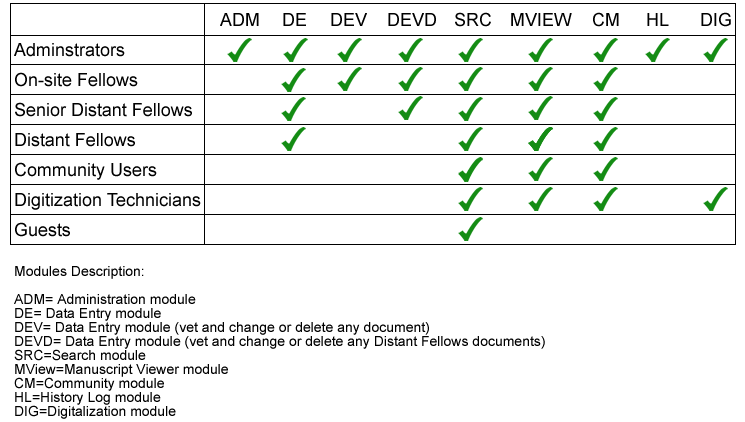
# Moduli applicativi



# Modello di sicurezza applicativa

L’applicazione prevede un modello di accesso ai servizi in base ad un group-policy di appartenenza. Ciascun utente, è assegnato ad uno specifico gruppo ed ogni gruppo ha accesso ad un set specifico di moduli applicativi.

Di seguito viene riportato la tabella di definizione dei vari gruppi :



(Organizzazione dei group-policy in funzione dei moduli applicativi)

Per implementare questo modello, è stato deciso di costruire un albero LDAP composto da due unità, l’unità utenti e l’unità gruppi. L’unità utente (oggetto LDAP di tipo **top** e **organizationalUnit**)contiene, suddiviso in n-foglie quanti sono gli utenti, ciascuna singola utenza di accesso al sistema corredata delle informazioni anagrafiche del titolare dell’accesso. La seconda unità (oggetto LDAP sempre di tipo **top** e **organizationalUnit**) definisce i group-policy (Aministrator, On-site Fellows, etc) corredati a loro volta della lista di utenti che posseggono.

Un’utente è un’oggetto composito generato da quattro classi LDAP differenti :

* organizationalPerson
* person
* extensibleObject
* top

L’estensione di queste quattro classi permette di memorizzare sull’albero tutte le le informazioni di anagrafica previste nella documentazione di progetto, sfruttando attributi standard LDAP, senza la necessità di creare nuovi attributi e oggetti LDAP custom che renderebbe lo schema dell’albero non standard.

Gli attributo dell’oggetto sono :

* l’account è memorizzato come **common-name**, identifica direttamente la foglia dell’utente sull’albero ldap
* Il cognome è un attributo di tipo **surname**
* Il nome dell’utent è memorizzato in un attributo **givenName**
* La nazione dell’utente è definita in un attributo **c**
* La città dell’utente è definita in un attributo **l**
* L’indirizzo di residenza dell’utente è definito in un attributo **street**
* L’indirizzo di posta elettronica è memorizzato in un attributi **mail**
* L’organizzazione di appartenenza dell’utente è definitia in un attributo **ou**
* Il titolo della persona (???) è definito in un attributo di tipo **personalTitle**
* La password per accedere al servizio è memorizzata in un attributo **userPassword** con cifratura SHA1
* Gli interessi personali dell’utente, sono memorizzati nell’attributo **info**
* Le group-policy sono memorizzate sotto forma di array nell’attributo **member**

E’ stato deciso di memorizzare l’associazione utente-group policy direttamente all’interno dell’’oggetto utente al fine di facilitare l’implementazioni di funzionalità di ricerca basate su group-policy.

Sotto riportiamo lo scheletro di un utente

|  |
| --- |
| dn: cn=medici,ou=Users,dc=docsources,dc=medici,dc=org  objectClass: organizationalPerson  objectClass: person  objectClass: extensibleObject  objectClass: top  cn: medici  surname: Person Surname  c: IT  givenName: People Name  info: Person interets  l: Florence  mail: medici@medici.org  member: COMMUNITY\_USERS  ou: Medici Archive Organization  personalTitle: Person Title  street: People address  userPassword:: e1NIQX01Z3h0WGlITlgzdURkUWtsSkd3aHVncktxV289 |

Le ricerche utenze sull’albero LDAP devono sempre specificare come base DN **ou=Users,dc=docsources,dc=medici,dc=org**.

Un group-policy è definito come un ruolo che fornisce all’utente i moduli applicativi ai quali è possibile accedere. Sull’albero LDAP è memorizzata la sola definizione dei gruppi, la composizione di ciascun gruppo è invece definita nel file di configurazione applicativa applicationSecurity.xml

Un group-policy è un’oggetto composito generato da due classi LDAP differenti :

* groupOfNames
* top

Gli attributi definiti sono :

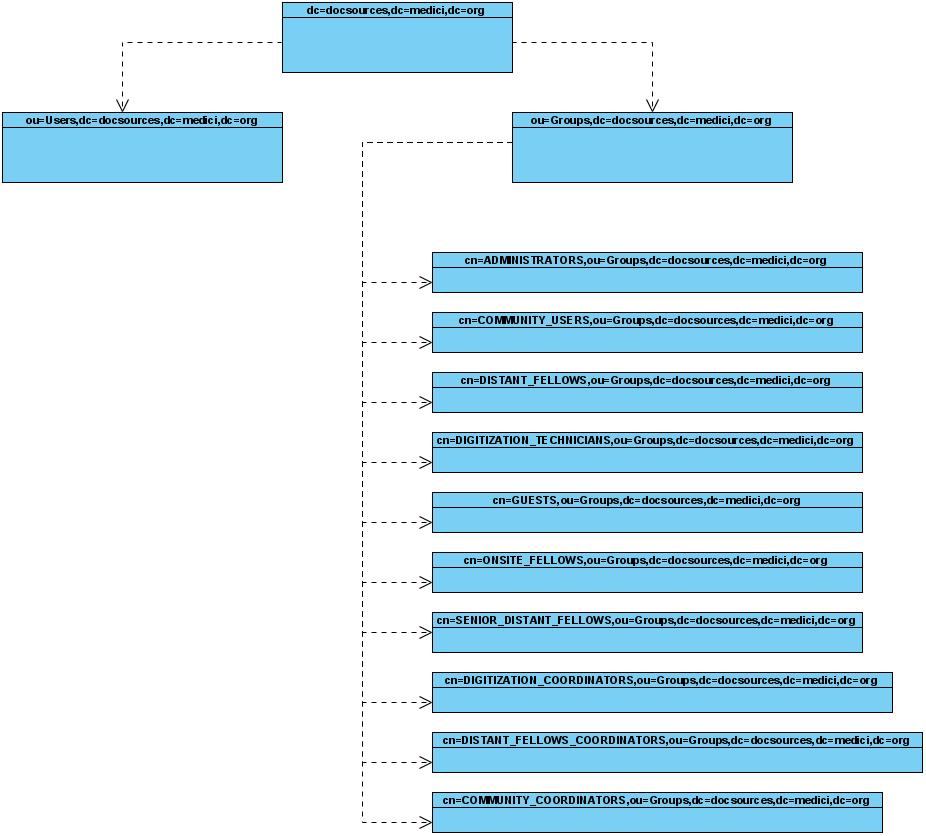
* L’identificativo del gruppo è definito nell’attributo **cn**
* La descrizione del gruppo è memorizzata nell’attributo description

Le ricerche dei group-policy disponibili devono sempre specificare come base DN **ou=Groups,dc=docsources,dc=medici,dc=org**.

Sotto è riportato lo schema di esempio del gruppo Community Users

|  |
| --- |
| dn: cn=COMMUNITY\_USERS,ou=Groups,dc=docsources,dc=medici,dc=org  objectClass: groupOfNames  objectClass: top  cn: COMMUNITY\_USERS  description: Community User Role |

Di seguito viene riportato lo schema logico dell’albero

****

Oltre ai 7 group-policy base esistono tre policy speciale :

* DIGITIZATION\_COORDINATORS
* DISTANCE\_FELLOWS\_COORDINATORS
* COMMUNITY\_COORDINATORS

Queste vengono assegnate ai coordinatori dei corrispettivi gruppi primari e forniscono alcune funzioni specifiche.

La gestione dei moduli è invece effettuata sfruttando la struttura degli url applicativi che si addicono estremamente bene a questa gestione.

Come definito del documentazione di progetto, l’applicazione è composta da nove moduli :

* adm
* de
* dev
* devd
* src
* mview
* cm
* hl
* dig

E’ stato deciso quindi di creare sulla web-application 9 path di primo livello, ciascuno dei quali rappresenta uno specifico modulo. All’interno di ciascun path, vengono inserite tutte le funzioni previste dal singolo modulo, nel caso in in cui una funzione dovesse essere resa disponibile in piu’ di un path.

L’associazione moduli / group policy di abilitazione viene fatta come abbiamo detto nel file applicationSecurity.xml, sfruttando il blocco della sicurezza http (security:http) . In questo tag, vengono definiti n tag <security:intercept-url>, uno per ciascun modulo, piu’ una serie di basic path che devono essere resi disponibili indipendentemente dall’autenticazione o meno dell’utente. Questi basic path, sono quelli che forniscono alcuni servizi base, come le immagine statiche che compongo gli elementi di decorazione grafico del portale, eventuali file javascript referenziati nel front-end, gli url per effettuare la login, ed altri ancora.

Per maggiori dettagli sul funzionamento del meccanismo di integrazione fra il server LDAP e il framwork springSecurity si consiglia la lettura della paragrafo LDAP Authentication della reference guide versione 2.0 (<http://static.springsource.org/spring-security/site/docs/2.0.x/reference/ldap.html>).

Qua sotto viene riportata la configurazione del tag <security:http>

|  |
| --- |
| <security:http auto-config="true">  <security:intercept-url pattern="/Login\*" access="IS\_AUTHENTICATED\_ANONYMOUSLY" />  <security:intercept-url pattern="/logoutSuccess\*" access="IS\_AUTHENTICATED\_ANONYMOUSLY" />  <security:intercept-url pattern="/css/\*\*" access="IS\_AUTHENTICATED\_ANONYMOUSLY" />  <security:intercept-url pattern="/images/\*\*" access="IS\_AUTHENTICATED\_ANONYMOUSLY" />  <security:intercept-url pattern="/scripts/\*\*" access="IS\_AUTHENTICATED\_ANONYMOUSLY" />  <security:intercept-url pattern="/user/RegisterUser.do" access="IS\_AUTHENTICATED\_ANONYMOUSLY" />  <security:intercept-url pattern="/user/ajax/\*\*.do" access="IS\_AUTHENTICATED\_ANONYMOUSLY" />  <security:intercept-url pattern="/adm/\*\*" access="ROLE\_ADMINISTRATORS" />  <security:intercept-url pattern="/de/\*\*" access="ROLE\_ADMINISTRATORS, ROLE\_ONSITE\_FELLOWS, ROLE\_DISTANT\_FELLOWS" />  <security:intercept-url pattern="/dev/\*\*" access="ROLE\_ADMINISTRATORS, ROLE\_ONSITE\_FELLOWS" />  <security:intercept-url pattern="/devd/\*\*" access="ROLE\_ADMINISTRATORS, ROLE\_ONSITE\_FELLOWS" />  <security:intercept-url pattern="/src/\*\*" access="ROLE\_ADMINISTRATORS, ROLE\_ONSITE\_FELLOWS, ROLE\_DISTANT\_FELLOWS, ROLE\_COMMUNITY\_USERS, ROLE\_DIGITIZATION\_USERS, ROLE\_GUESTS" />  <security:intercept-url pattern="/mview/\*\*" access="ROLE\_ADMINISTRATORS, ROLE\_ONSITE\_FELLOWS, ROLE\_DISTANT\_FELLOWS, ROLE\_COMMUNITY\_USERS, ROLE\_DIGITIZATION\_USERS" />  <security:intercept-url pattern="/cm/\*\*" access="ROLE\_ADMINISTRATORS, ROLE\_ONSITE\_FELLOWS, ROLE\_DISTANT\_FELLOWS, ROLE\_COMMUNITY\_USERS, ROLE\_DIGITIZATION\_USERS" />  <security:intercept-url pattern="/hl/\*\*" access="ROLE\_ADMINISTRATORS" />  <security:intercept-url pattern="/dig/\*\*" access="ROLE\_ADMINISTRATORS, ROLE\_DIGITIZATION\_USERS" />    <security:intercept-url pattern="/\*\*.do" access="ROLE\_ADMINISTRATORS, ROLE\_ONSITE\_FELLOWS, ROLE\_DISTANT\_FELLOWS, ROLE\_COMMUNITY\_USERS, ROLE\_DIGITIZATION\_USERS, ROLE\_GUESTS" />    <security:intercept-url pattern="/user/\*\*.do" access="ROLE\_ADMINISTRATORS, ROLE\_ONSITE\_FELLOWS, ROLE\_DISTANT\_FELLOWS, ROLE\_COMMUNITY\_USERS, ROLE\_DIGITIZATION\_USERS, ROLE\_GUESTS" />  <security:form-login login-page="/Login.do"  login-processing-url="/loginProcess"  default-target-url="/Home.do"  authentication-failure-url="/Login.do?login\_error=1" />  <security:logout logout-url="/Logout.do" logout-success-url="/Login.do" />  </security:http> |

Cerchiamo adesso di spiegare ogni singolo tag :

Prima di tutto dichiara l’intercpt-url per il path che inizia con /login, e definisce che l’accesso per qualunque tipologia di ruolo dell’utente. Lo stesso concetto viene applicato anche ai 6 path successivi, e questo garantisce la disponibilità pubblica di alcuni url che servono per effettuare le operazioni di registrazione alla piattaforma autenticazione e l’operazione di logout.

Poi dichiara che l’accesso al path che inizia per /adm/, deve essere garantito soltanto se l’utente appartiene al ruolo ROLE\_ADMINISTRATORS. Seguono le dichiarazioni degli altri otto ruoli con l’accesso dei corrispettivi group-policy di associazione definiti in base allo schema in tabella all’inizio del paragrafo.

Poi dichiara che l’accesso al path che iniza per /\*\*.do deve essere dato a tutti gli utenti appartenenti ai nove group-policy, questa regola serve perché sotto il path / inserire alcune funzionalità base, come l’home in cui l’utente verrà reindirizzato effettuata l’autenticazione, i path per accedere alle opzioni applicative e queste funzioni devono essere disponibili per tutti gli utenti che si sono loggati ed hanno un ruolo fra quelli definiti.

Il blocco di intercept-url successivo dichiara le regole di accesso al path /user/. Questo path viene utilizzato per le funzionalità di accesso all’anagrafica utente. E’ una dichiarazione al momento provvisoria in quanto sarà da rimodellare dato che i path dovrebbero essere solo identificativi dei corrispettivi moduli funzionali applicativi, ma momentaneamente per effettuare dei test d’integrazione con ldap è stato posizionato qua dentro.

Il blocco <security:form-login login> definisce le regole per effettuare le operazioni di autenticazione, logout, ed i path di destinazione in caso di autenticazione con successo e autenticazione fallita. I nomi di questi path sono dichiarati secondo gli standard del modulo JAAS (Java Authentication and Authorization Schema).

Qui termina la sezione di sicurezza dichiarativa del modulo http. Con questa sezione abbiamo definiti i ruoli di accesso alle varie sezioni dell’applicazione. Tenuto conto che le jsp che l’applicazione contiene sono tutte dichiarative private e non sono pubblicamente accessibili, è chiaro che l’utente potrà invocare funzionalità che sono gestite esclusivamente dai Controllers MVC applicativi e che, essendo questi sotto la supervisione e il controllo del modulo Spring Security, non esiste la benché minima funzionalità per un’utente di accedere a funzionalità di cui non dispone i diritti di accesso perché non previsti dal suo group-policy.

Andiamo adesso a spiegare la seconda parte e forse anche la piu’ complessa della sicurezza applicativa, la gestione del processo di autenticazione dell’utente.

|  |
| --- |
| <security:authentication-manager>  <security:authentication-provider>  <security:user-service>  <security:user name=*"guest"* password=*"guest"* authorities=*"ROLE\_GUESTS"* />  </security:user-service>  </security:authentication-provider>  <security:authentication-provider ref=*"ldapAuthenticationProvider"*>  <security:password-encoder ref=*"passwordLdapShaEncoder"*/>  </security:authentication-provider>  </security:authentication-manager> |

Il tag <security:authentication-manager> dichiara tutti i provider previsti per effettuare il processo di autenticazione.

L’applicazione, deve prevedere due tipologie di accesso al sistema :

* Accesso come visitatore (GUEST)
* Accesso con credenziali personali

La prima tipologia, è una tipologia di utente con permessi di solo accesso alla community e senza possibilità di interagire con tutte le funzionalità del sistema; la seconda tipologia di accesso è vincolata all’accettazione da parte dell’archivio della richiesta di registrazione dell’utente. Poiché la prima tipologia di accesso è condivisa, e non prevede la memorizzazione, è stato deciso di configurare un provider d’accesso specifico per gli utenti guests, con uno user-service di tipo “memory”. Per la seconda tipologia invece è stato definito un Authentication Provider di tipo LDAP.

Questi provider vengono invocati dal container Spring Security in cascata, secondo l’ordine di definizione.

, quale è il riferimento al bean che si occupa del servizio di gestione degli utenti (user-service-ref) e quale è il bean che si occupa di gestire l’encoding delle password. L’ldapUserDetailsService, ha il compito di effettuare tutte le operazioni necessarie per effettuare il recupero dell’utente che si desidera autenticare.

Questo bean è stato dichiarato utilizzando l’istanza specifica di SpringSecurity per gestire gli utenti su albero LDAP. Viene istanziato sfruttando il costruttore a due parametri, che riceve in ingresso come primo parametro il bean preposto alla ricerca dell’utente, e come secondo il bean preposto alla valorizzazione delle autority a cui l’utente è abilitato. Viene inoltre definita una proprietà che serve per specificare il servizio che è in grado di mappare gli attributi di dettaglio dell’utente presenti sulla base dati da cui si recupera i dati e le proprietà dell’implementazione UserDetails; in questo caso viene utilizzato un mapper custom sviluppato dal team.

|  |
| --- |
| <bean id="ldapUserDetailsService" class="org.springframework.security.userdetails.ldap.LdapUserDetailsService">  <constructor-arg index="0">  <ref local="userSearch" />  </constructor-arg>  <constructor-arg index="1">  <ref local="ldapAuthoritiesPopulator" />  </constructor-arg>  <property name="userDetailsMapper" ref="ldapUserDetailsMapper"></property>  </bean> |

La classe preposto al recupero dei dettagli utente come abbiamo detto ha bisogno pero’ di integrarsi con il bean preposto ad effettuare la ricerca di un utente (userSearch). Questo bean, che istanzia l’implementa di ricerca su albero ldap mediante filtri, deve conoscere il nodo dell’albero LDAP sotto il quale sono memorizzati gli utenti (ou=Users,dc=docsources,dc=medici,dc=org), l’attributo dell’utente (cn=) che ne identifica l’account. La stringa {0} è una stringa identificativo per specificare che deve essere sostituita con il valore che la classe riceverà in ingresso dal frame work e sulla quale verrà creato il filtro di ricerca vero e proprio. Tale stringa non sarà altro che il valore inserito nella textfield del campo account della maschera di login dall’utente. Il bean di ricerca poi dichiara un secondo parametro per il costruttore, che è il securityContextSource, esso non è altro che il bean contenente tutte le informazioni di contesto necessarie per potersi integrare alla sorgente dati. Per ultima, ma non meno importante, è definita una proprietà del filtro che permette di estendere la ricerca non solo al nodo ou=Users,dc=docsources,dc=medici,dc=org ma anche ad eventuali sottonodi.

|  |
| --- |
| <bean id=”userSearch" class="org.springframework.security.ldap.search.FilterBasedLdapUserSearch">  <constructor-arg index="0">  <value>ou=Users,dc=docsources,dc=medici,dc=org</value>  </constructor-arg>  <constructor-arg index="1">  <value>cn={0}</value>  </constructor-arg>  <constructor-arg index="2">  <ref local="securityContextSource" />  </constructor-arg>  <property name="searchSubtree" value="true" />  </bean> |

Il securityContextSource, definito anche questo mediante la specifica istanza che permette di gestire queste informazioni nel formato specifico per LDAP, dichiara l’albero LDAP su cui si trova l’anagrafica utenti, dato che il nostro albero prevede l’autenticazione per poter effettuare le operazioni di ricerca, dichiara anche quale è il gestore di autenticazione per l’albero da usare (authenticationSource) e per ultime alcune proprietà di base (timeout di connessione e nome della classe LDAP preposta alla gestione degli attributi binari).

|  |
| --- |
| <bean id="securityContextSource" class="org.springframework.security.ldap.DefaultSpringSecurityContextSource">  <constructor-arg value="ldap://localhost:10389" />  <property name="authenticationSource" ref="authenticationSource"/>  <property name="baseEnvironmentProperties">  <map>  <entry key="com.sun.jndi.ldap.connect.timeout" value="60000" />  <entry key="java.naming.ldap.attributes.binary" value="objectGUID"/>  </map>  </property>  </bean> |

L’autenticationSource a sua volte dichiara, le credenziali (defaultUser e default Password) quindi per poter effettuare far fare l’autenticazione al securityContextSource e quale è la classe di riferimento che effettua l’autenticazione vera e propria (springSecurityAuthenticationSource).

|  |
| --- |
| <bean id="authenticationSource" class="org.springframework.ldap.authentication.DefaultValuesAuthenticationSourceDecorator">  <property name="target" ref="springSecurityAuthenticationSource" />  <property name="defaultUser" value="cn=Administrator,ou=Users,dc=docsources,dc=medici,dc=org" />  <property name="defaultPassword" value="admin" />  </bean> |

Come classe preposta all’autenticazione server to server per poter fare le operazioni di ricerca sull’albero LDAP, abbiamo l’istanza specifica per LDAP.

|  |
| --- |
| <bean id="springSecurityAuthenticationSource" class="org.springframework.security.ldap.SpringSecurityAuthenticationSource"/> |

Per effettuare l’autenticazioe c’e’ inoltre bisogno di dichiarare nello specifico l’authentication-provider come custom

|  |
| --- |
| <bean id="ldapAuthenticationProvider" class="org.springframework.security.providers.ldap.LdapAuthenticationProvider">  <security:custom-authentication-provider/>  <constructor-arg index="0">  <ref local="bindAuthenticator" />  </constructor-arg>  <constructor-arg index="1">  <ref local="ldapAuthoritiesPopulator" />  </constructor-arg>  </bean> |

Oltre a questo per poter funzionare il meccanismo di autenticazione, il frame work di Spring Security richiede la definizione di un bindAuthenticator, cioe’ l’oggetto che in sostanza effettua l’autenticazione come utente. Questo richiede : un securityContextSource per conoscere gli estremi dell’ldap a cui agganciarsi e la modalità di ricerca utenti (bean userSearch)

|  |
| --- |
| <bean id="bindAuthenticator" class="org.springframework.security.providers.ldap.authenticator.BindAuthenticator">  <constructor-arg>  <ref local="securityContextSource"/>  </constructor-arg>  <property name="userSearch" ref="userSearch" />  </bean> |

Oltre alla gestione dell’autenticazione non dobbiamo ricordarci di gestire anche il processo di autorizzazione di un utente, dato ad inizio del file applicationSecurity abbiamo definito n-path che sono sottoposti ad un controllo per verificare se un client appartiene ad un determinato group-policy, come faccio pero’ a recuperare queste informazioni? Il bean ldapAuthoritiesPopulator serve proprio a questo :

sfruttando il contextSource per conoscere l’albero LDAP a cui effettuare le richieste, il bean è istruito per prelevare la policy a cui l’utente appartiene, mediante un’interrogazione sull’albero ou=Groups (del base DN dc=docsource,dc=medici,dc=org), andando a cercare in ogni sotto albero rappresentante un ruolo, la presenza dell’informazione dell’utente. Se questa informazione viene trovato il nodo padre è considerato la policy-group a cui l’utente appartiene (questa regola è specificata da groupRoleAttribute).

Negli url-intercept definiti ad inizio file le policy si chiama ROLE\_ e sono in formato upper case, mentre i nodi figli di ou=GROUPS sono nomi che inizia per la prima lettera maiuscola ma non hanno il suffisso ROLE\_.

ldapAuthoritiesPopulator è stato pero’ configurato al fine di immettere il suffisso ROLE\_ davanti alla policy dell’utente(rolePrefix=ROLE\_) , e il risultato lo converte in upper case (convertToUpperCase=true). Questo svelato quindi il meccanismo di autorizzazione fra gli intercept url e l’albero LDAP delle policy ou=Groups.

|  |
| --- |
| <bean id="ldapAuthoritiesPopulator" class="org.springframework.security.ldap.populator.DefaultLdapAuthoritiesPopulator">  <constructor-arg ref="contextSource"/>  <constructor-arg value="ou=Groups"/>  <property name="groupRoleAttribute" value="cn"/>  <!-- the following properties are shown with their default values -->  <property name="searchSubtree" value="true"/>  <property name="rolePrefix" value="ROLE\_"/>  <property name="convertToUpperCase" value="true"/>  </bean> |

Il password Encoder bean è l’oggetto referenziato dal tag security:authentication-provider per la gestione dell’encoding del campo password dell’utente che si sta autenticando. La password dell’utente viene memorizzata dentro l’oggetto class dell’utente sotto il nodo Users,**dc=docsources,dc=medici,dc=org**, all’interno della proprietà userPassword. Tale campo memorizzat, con encoding sha verisone 1, la password che l’utente ha definito durante il processo di autenticazione.

|  |
| --- |
| <bean id="passwordEncoder" class="org.springframework.security.providers.encoding.ShaPasswordEncoder">  <constructor-arg value="1"/>    <property name="encodeHashAsBase64" value="false"></property>  </bean> |

Lo userDetailsMapper è una classe applicativa che serve per mappare i dettagli di un’utente presenti sul relativo nodo LDAP. nel corrispettivo oggetto java. La classe estende la classe di spring security LdapUserDetailsMapper, e ne sovrascrive il metodo **public** UserDetails mapUserFromContext(DirContextOperations ctx, String username, GrantedAuthority[] authorities)

|  |
| --- |
| <bean id="ldapUserDetailsMapper" class="org.medici.docsources.security.DocSourcesLdapUserDetailsMapper"/> |

Oltre a questi bean, troviamo altri due, che sono i bean per l’accesso all’albero LDAP da parte dello UserDao : ldapTemplate e contextSource.

L’ldapTemplate è un’oggetto per gestire tramite un’oggetto di esempio, la generazione di una struttura LDAP da leggere o da persistere sull’albero.

|  |
| --- |
| <bean id="ldapTemplate" class="org.springframework.ldap.core.LdapTemplate">  <constructor-arg ref="contextSource" />  <property name="ignorePartialResultException" value="true"/>  </bean> |

L’istanziamento dell’oggetto viene effettuato mediante il costruttore che riceve in ingresso un contextSource contenente le informazioni di autenticazione. E’ un bean simile al SecurityContextSource, al suo interno vengono definte le proprietà, l’url del server LDAP a cui il bean effettua la connessione per il recupero delle informazioni, base DN Ldap, utenza e password per effettuare l’autenticazione, piu’ una mappa contenente le opzioni di enviroment.

|  |
| --- |
| <bean id="contextSource" class="org.springframework.ldap.core.support.LdapContextSource">  <property name="url" value="ldap://localhost:10389" />  <property name="base" value="dc=docsources,dc=medici,dc=org" />  <property name="userDn" value="cn=Administrator,ou=Users,dc=docsources,dc=medici,dc=org" />  <property name="password" value="admin" />  <property name="baseEnvironmentProperties">  <map>  <entry key="com.sun.jndi.ldap.connect.timeout" value="60000" />  <entry key="java.naming.ldap.attributes.binary" value="objectGUID"/>  </map>  </property>  </bean> |

Il meccanismo di autenticazione viene innescato dall’utente quando, dopo aver richiesto l’url /login.do, ne popola il relativo form e invia la richiesta, il frame work di Spring inizia ad invocare i vari filtri http fino all’AuthenticationProcessingFilter. Il filtro per il processamento dell’autenticazione, invoca l’handler NameSpaceAuthentication che tramite il metodo autenticate provvede ad autenticare l’utente mediante il fornitore del DataAccess Object di autenticazione. Questo oggetto per autenticare l’utente, effettua :

* recupero l’utente dalla base dati
* effettua i controlli di pre autenticazione
* effettua gli eventuali controlli di autenticazione addizionali
* effettua i controlli di post autenticazione
* crea un oggetto di tipo SuccessAuthentication.

Per gestire all’interno delle View restituite all’utente, le policy di sicurezza, sfrutturemo i tag library standard di SpringSecurity.

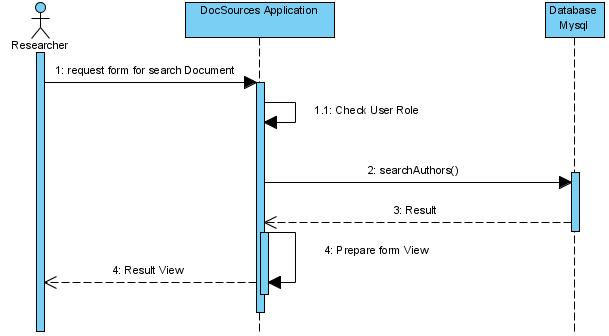
# Modello di navigazione utente

In questo paragrafo effettueremo l’analisi del modello di navigazione adottato, in particolare ci interessa focalizzarci sullo studio dell’interazione fra l’utente e l’applicazioni per poter studiare come l’applicazione risponde ad una richiesta, quali componenti vengono attiviate e perché si attivano alcuni componenti piuttosto che altri.

Per questo studio, abbiamo deciso di prendere in esempio un’azione di ricerca documenti che prevede una maschera di ricerca contenente una combo box di autori su cui effettuare un restringemento di selezione. Supponiamo che alcuni specifiche logiche che l’utente deve soddisfare siano presenti (autenticazione sul sistema, appartenenza dell’utente ad un gruppo di policy contenente la funizone di ricerca attivata).

Il primo step che effettua l’utente è selezionare il link dall’interfaccia che fornisce la maschera di inserimento dati per effettuare la ricerca. Come possiamo vedere dal grafico qui sotto, l’applicazione esegue una serie di operazioni :

* L’utente inoltra al sistema una richiesta di ricerca Documenti (punto 1)
* L’applicazione ricevuta la richiesta, effettua come prima operazione la verifica delle user Role per controllare che l’utente sia abilitato alla funzione richiesta, in caso contrario restituisce una pagina di errore (punto 1.1)
* Dato che la maschera deve presentare una lista di autori non statici, questi devono essere estratti dal database, per effettuare questa operazione, l’applicazione interroga il database mysql (punto 2) il quale provvede a restituire i dati richiesti (punto 3)
* Ricevuta la risposta dell’interrogazione, l’applicazione provvede ad organizzare i dati ricevuti in base ad una logica interpretabile dalla View di risposta che dovrà esssere restituita all’utente. L’applicazione quindi provvede a prepare la View della maschera di ricerca contentente un campo combo box, popolato con il risultato dell’interrogazione alla base dati. (punto 4)
* La view viene restituita al client che puo’ cosi’ procedere allo step successivo

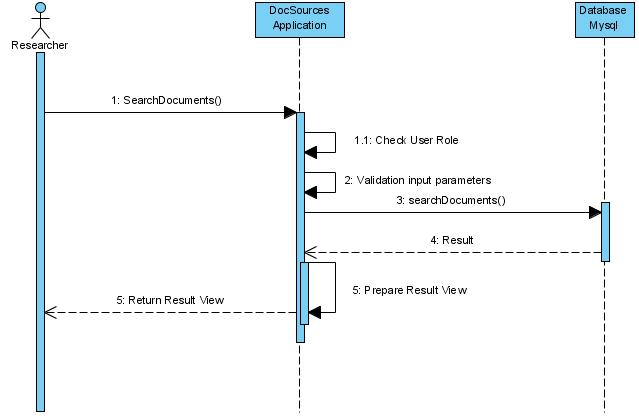


(richiesta form di ricerca documenti)

Effettuata la richiesta del form, l’utente provvederà a riempirlo con i dati ed inviare una richiesta di elaborazione di ricerca documenti all’applicazione.

Nel sequence diagram sotto riportato viene mostrata i macro step che vengono generati quando arriva una elaborazione di un form applicativo :

* L’utente inoltra la richiesta al server applicativo (punto 1)
* L’applicazione ricevuta la richiesta, effettua come prima operazione la verifica delle user Role per controllare che l’utente sia abilitato alla funzione richiesta, in caso contrario restituisce una pagina di errore (punto 1.1)
* L’applicazione effettua la validazione logica dei dati ricevuti insieme alla richiesta (punto 2)
* Se il processo di validazione è andato a buon fine, provvede ad elaborarla effettuando le interrogazioni del caso sulla base dati (punto 3 – 4)
* L’applicazione prepara la View di risposta popolando con i dati estratti dal database (punto 5)
* L’utente riceve la pagina del risultato



(elaborazione della richiesta “ricerca documenti”)

A livello macroscopico possiamo supporre che gran parte delle richieste che l’applicazione dovrà soddisfare rientrerà in questo modello logico.

Adesso andremo a vedere come nel dettaglio avviene l’elaborazione della richiesta di ricerca :

L’applicazione è composta da una serie di oggetti differenti, ognuno con il proprio compito specifico assegnato.

Dal diagramma possiamo chiaramente identificare le seguenti componenti :

1. SimpleSearchDocumentsController

Questo oggetto è il controller Spring che è preposto alla generazione della maschera di ricerca che all’elaborazione della richiesta. Rappresenta il punto di ingresso di ognuni specifica richiesta di ricerca semplice di documenti. Il controller ha possibilità di accedere ai validator (per effettuare le validazioni) e ad oggetti di tipo Service per effettuare la business logica del flusso. Da sottolineare come il controller non abbia l’accesso diretto allo strato DAO che si occupa di interrogare il database.

1. SimpleSearchDocumentsValidator

Questo oggetto di occupa di prendere in carico tutte le validazioni logiche che il flusso richiede.

1. DocumentsService

Questo oggetto rappresenta il layer di business specifico dei documents. Al suo interno sono presenti una serie di metodi che possono essere invocati dal controller per soddisfare la richiesta in ingresso. Ogni service puo’ accedere ad uno dei 24 dao che necessita per poter fare le eventuali interrogazioni al database. Nel caso che abbiamo preso in esame, il service sfrutta due differenti dao, quelli dei Document, e quello per l’accesso alle entità Place.

1. DocumentDAO

Questo oggetto, rappresenta il piu’ basso livello applicativo prima del database, e serve per effettuare le interrogazioni sull’entità dei documenti.

1. PlaceDAO

Questo oggetto, rappresenta il piu’ basso livello applicativo prima del database, e serve per effettuare le interrogazioni sulle entità dei luoghi.

1. MySQL Database

Questo oggetto rappresenta la base dati, il cui accesso è permesso soltanto alle classi DAO.

1. SimpleSearchDocumentsView

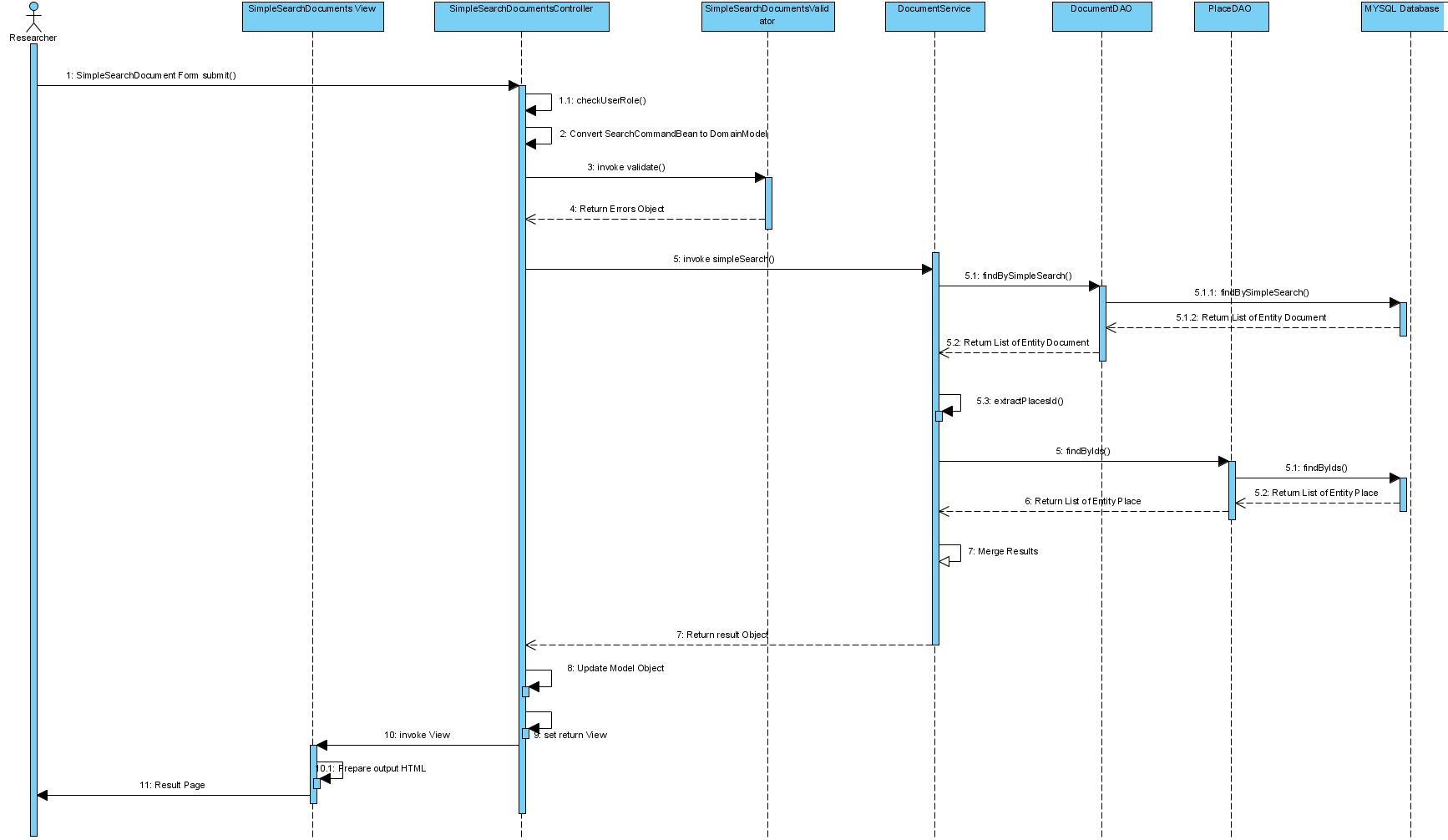
Questa è la view di elaborazione del flusso che viene restituita all’utente da parte del controller al termine dell’esecuzione della business logic del flusso. Implementativamente si tratta di una view tiles composta da un insieme di jsp che hanno il compito di scrivere codice HTML

Il flusso segue una logica molto semplice, è intenzione del team di sviluppo cercare il piu’ possibile di mantenersi a questo modello :

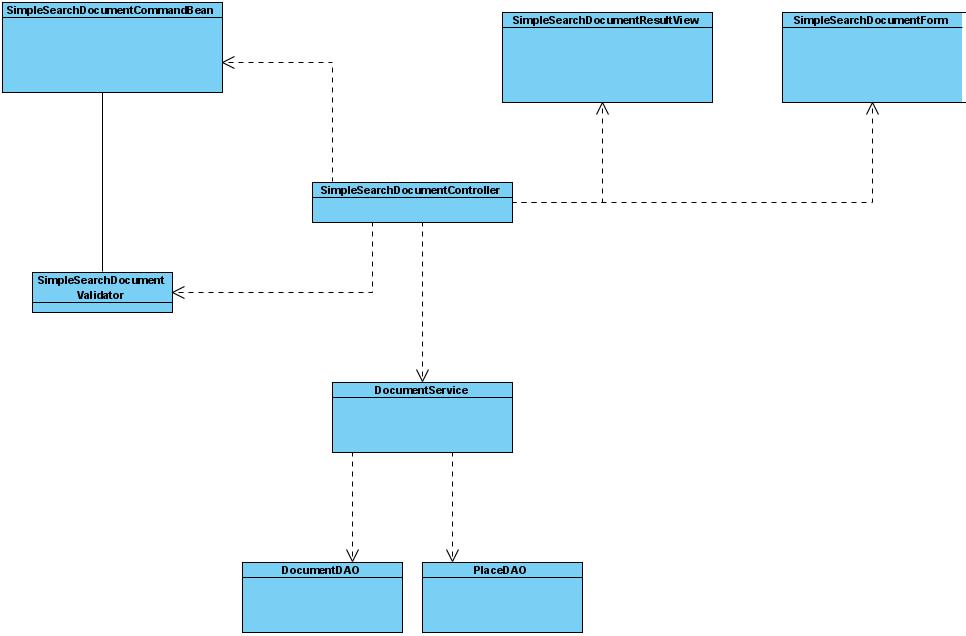
* L’utente inoltra la richiesta all’applicazione, questa viene intercettata dal controller SimpleSearchDocumentsController (punto 1)
* Il controller verifica (in automatico tramite Spring Security) i permessi associati al gruppo dell’utente per capire se la richiesta è fornita dell’autorizzazione richiesta (punto 1.1)
* Il controller effettua la validazione dei dati fornito con la richiesta (punto 2-3)
* I parametri della richiesta di ricerca (a questo punto validati ndr) vengono forniti in automatico al controller dal framework sotto la forma di un CommandBean denominato SimpleSearchDocumentsCommandBean, tale bean viene convertito in un formato comprensibile dall’oggetto DocumentService (punto 4)
* Il controller invoca lo specifico metodo del DocumentService per effettuare la ricerca dei documenti, al parametro viene passato una versione comprensibile del SimpleSearchDocumentsCommandBean (punto 4)
* Il service provvede ad invocare sull’oggetto DocumentDAO, il metodo di ricerca. Questo metodo, sfruttando il framework JPA, interroga il database e restituisce un risultato formato da una lista di entità Document che matchano con i parametri di ricerca inseriti dall’utente (punto 4.1 e punto 4.2)
* Si suppone a questo punto che il service, ha bisogno di comunicare al controller anche i riferimenti completi dei luoghi presenti nei documenti estratti, quindi necessita di interrogare il database per ottenere le informazioni di questi luoghi. Prima di tutto provvede ad estrarre gli identificativi dei luoghi presenti nei documenti estratti (punto 4.3)
* Tramite l’oggetto PlaceDAO, effettua un’interrogazione sul database per ottenere i luoghi a partire dalla lista dei loro identificativi (punti 5,5.1, 5.2)
* Ottenuto la lista dei luoghi (punto 6) , provvede a costruire un oggetto da restituire al controller contenente tutte le informazioni utili per comporre il risultato atteso e lo invia al controller (punto 7)
* Il controller a questo punto preleva il risultato ottenuto dalla richiesta di servizio e iniziare ad impostare nel model MVC, gli oggetti necessari alla composizione del risultato della view (punto 8)
* Impostati tutti i dati necessario, notifica al framework MVC, quale View deve essere istanziata per fornire il giusto output all’utente (punto 9)
* Il framework invoca la View, la quale provvede alla costruzione del codice html che deve essere restituito all’utente ed una volta terminata l’operazione, lo notifica al framework Spring MVC (punto 10.1)
* Il framework a questo punto si occupa semplicemente di passare il codice html generato dalla View al browser dell’utente che ne ha fatto richiesta

Da questo sequence diagram vogliamo in particolare sottolineare la separazione per strati e l’organizzazione del codice applicativo.

Le richieste che un’utente effettua vengono sempre prese in carico da un Controller, il quale deve prima di tutto verificare le policy di autorizzazione (questo per evitare che un utente con solo permessi di consultazione, tenti di effettuare un’accesso non autorizzazione a funzionalità di modifica dei documenti), successivamente si deve occupare di validare i dati presenti nella richiesta, se un parametro manca, se è stato specificato un termine che non esiste deve essere notificato. La validazione pero’ è un processo non strettamente legato al Controller ma al formato dei dati in ingresso. Il framework Spring MVC, in congiunzione con la funzionalità Validation inserita nel jdk 6, fornisce un’ottimo strumento per incapsulare il codice di validazione in un’oggetto specifico (denominato appunto Validator) che garantisce un’maggior livello di isolamento del codice di verifiche a tutto vantaggio dei test funzionali. Per questo motivo tutto cio’ che è inerente la validazione dei dati viene inserito sia nell’oggetto che rappresenta i dati in ingresso (CommandBean), sotto la forma di annotazione java che in un oggetto Validator, che ha il compito di effettuare le validazione logiche. A questo punto il Controller per eseguire la richiesta, deve utilizzare soltanto oggetti di tipo Service. I Service hanno il compito di prendere i parametri, invocare le classi DAO preposte all’interrogazioni delle varie tabelle ed elaborare degli oggetti che rappresentino cio’ che è stato richiesto. Ogni service puo’ referenziare tanti dao quanti ne sono necessari per riuscire ad effettuare tutte le interrogazioni necessarie. Ottenuto il risultato dal Service, il Controller procedere con il popolamento dell’oggetto Model del View di risposta. Il framework Spring MVC è in grado di intercettare ad questo punto la View che deve essere fornita come risposta al client ed elaborarla.

(schema dettagliato dell’elaborazione applicativa di una richiesta di processing form)

Di seguito viene riportato il diagramma ad oggetti di composizione di un flusso con tutti gli attori previsti :



(schema di composizione di un flusso standard DocSources)

# Il processo di validazione dei dati

Il modello architetturale prevede l’implementazione di un processo di validazione dei dati inseriti dai client durante le richieste di elaborazione.

Questo processo viene attivato automaticamente quando viene richiesto al server di apportare una modifica alla base dati applicativa.

# Frontend applicativo

L’applicazione DocSources fornisce un frontend web based, composto da un set di pagine a contenuto dinamiche con orientamento web 2.0 .

Il core del presentation layer sarà un insieme di pagine html (w3c compliant), che verranno gestite tramite l’integrazione fra Spring e il frame work Apache Tile, ognuna di queste pagine, sarà il risultato di un elaborazione di uno specifico flusso applicativo presente sull’applicazione. Ogni di questi flussi, avrà una templateView definito nel file di configurazione (applicationTemplate.xml); questo templateView verrà referenziato nel controller Spring preposto all’esecuzione del flusso applicativo.

L’orientamento web 2.0 verrà implementato mediante l’utilizzo della libreria jQuery abbinata ad una serie di plugin, che fornirà quindi dei widget dinamici senza la necessità di effettuare il refresh delle pagine in cui verrà utilizzata. Molti di questi widget sfrutteranno dei meccanismi di callback verso il portale per il reperimento dei dati in ingresso. Dal punto di vista invece della risposta che il server invierà alla callback, è necessaria l’implementazione di un meccanismo ad elevata flessibilità in quanto, quando si lavora con frame work come jQuery generalmente è possibile richiedere tipologie differenti di oggeti (alberi DOM completi, alberi DOM parziali, oggetti JSON, oggetti XML etc). Tenuto conto di questa necessità, e dell’ulteriore aspetto che la stesura del codice lato server deve seguire dei canoni quick and simple (al fine di evitare la proliferazioni di blocchi **spaghetti code**), è stato deciso di configurare il frame work spring al fine di gestire in maniera autonoma i seguenti contenuti :

* Html
* XML Object
* JSON Object

Questa gestione è possibile grazie all’introduzione nella versione 3 di Spring di un ContentNegotiatingViewResolver che decide quale ViewResolver istanziare in base al content-type richiesto dal client. Questo oggetto è stato quindi configurato per differenziare il content da restituire in base all’estensione dell’url invocato :

* I path con estensione .do, generano Java Server Pages
* I path con estensione .json, generano oggetti JSON
* I path con estensione .xml, generano documenti XML

Data la necessità web 2.0, abbiamo elaborato una configurazione di Spring che ci permette di gestire comodamente, sia la produzione di View con contenuto

Per ottenere un’elevato livello di linearià del ciclo di sviluppo, il team di lavoro effettuerà la creazione di un sito pilota in codice html contenente tutte le funzioni applicative richieste dalle specifiche di sviluppo della piattaforma.

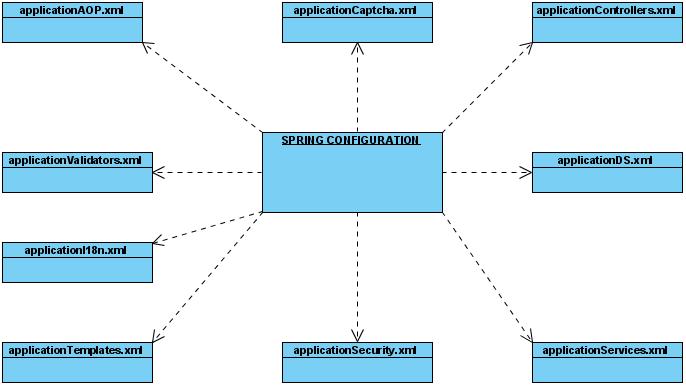
Lo sviluppo del portale demo dovrà essere effettuato seguendo tutte le specifiche e le nomenclature del portale applicativo, rispettando i path dei flussi che verranno definiti, i path di definizione delle immagini e dei script javascript dovranno essere identici.

Al fine di garantire, durante la stesura del sito pilota, una sincronizzazione del lavoro fra tutti i membri del team, utilizzeremo anche per questo il subversion, mediante la creazione di un repository ad hoc, all’interno del quale verranno inserite tutte le componenti che compongono il sito pilota.

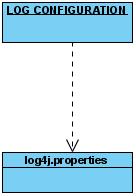
# Configurazione applicativa

I file di configurazione sono quindici, suddivisi in tre gruppi :

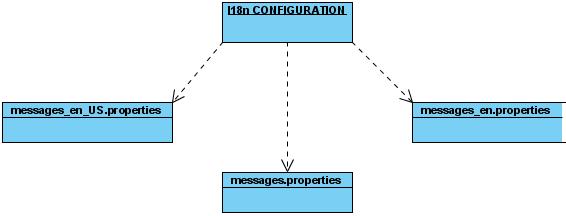
1. File di configurazione di Spring (nove file)
2. File di logging applicativo (un singolo file)
3. File di internazionalizzazione delle stringhe



(Schema dei file di configurazione applicativi del frame work Spring MVC)



(Schema dei file di configurazione applicativi del logging applicativo)



(Schema dei file di internazionalizzazione dell’applicazione)

I file di configurazione di Spring sono dieci, suddivisi per layer funzionali :

1. applicationAOP.xml, questo file definisce le componenti ad aspetti dell’applicazioni. Qua sono definiti i meccanismi di logging applicativo e le transazioni per le compenenti JPA
2. applicationCaptcha.xml, In questo file è definita la configurazione dei Bean preposti alle funzionalità di Captcha per le maschere che devono essere sottoposte a controllo antispam
3. applicationControllers.xml, in questo file è definita la configurazione delle componenti Controller dell’MVC di Spring, in particolare sono definite, il component-scan per il deteching della spring annotation @Controller, gli interceptor (per gestire l’history log e il locale con cui localizzare le pagine fornite ai client), il risolutore del content type per gestire in base all’estensione dei flussi i canali XML e JSON, il bean preposto all’auto istanziamento dei validator dichiarativi dei CommandBean, ed il resolver di locale in funzione della sessione http.
4. applicationDS.xml, in questo file sono contenute le configurazioni per l’interfacciamento al database e per la configurazione del provider utilizzato dal frame work JPA
5. applicationI18n.xml, file contenente il bean che definisce i file di properties contenenti le internazionalizzazioni di tutti i messaggi applicativi
6. applicationMail.xml, in questo file sono definiti i bean preposti all’invio delle email applicative dalle funzionalità di recupero anagrafico che vengono fornite all’utente mediante degli appositi form html.
7. applicationSecurity.xml, questo file contiene i bean di configurazione della sicurezza applicativa. Definisce i livelli di sicurezza di tutti i path applicativi in funzione dei gruppi, le coordinate dell’albero ldap contenente l’anagrafiche utenti e gruppi, le configurazioni per effettuare le ricerche degli stessi.
8. applicationServices.xml, questo file dichiara il componenent scan per gli oggetti Services, e tutti i Services che sono presenti nell’applicazione.
9. applicationTemplates.xml, in questo file sono definiti i templates di Tiles di tutte le View che un client puo’ visualizzare, sia i form per l’inserimento dati, che le View di risposta alle elaborazioni
10. applicationValidators.xml, in questo file sono dichiarati tutti i validators che i Controller della componente MVC possono utilizzare.

L’applicazione prevede la localizzazione dei messaggi nella sola lingua inglese. Per gestire la componente i18n è stato deciso di sfruttare le funzionalità contestuali di Spring (messageSource). Sono stati definiti quindi tre file di risorsa, contenenti tutti i messaggi applicativi che vengono visualizzati all’utente :

* messages.properties, file di localizzazione di default
* messages\_en\_US.properties, file di localizzazione del locale en\_US
* messages\_en\_US.properties, file di localizzazione del locale en

Si ricorda che le componenti che prevedono una dichiarazione all’interno di questi file, non possono non essere dichiarate al loro interno, pena l’impossibilità da parte di Spring di ottenerne un’istanza durante il runtime.

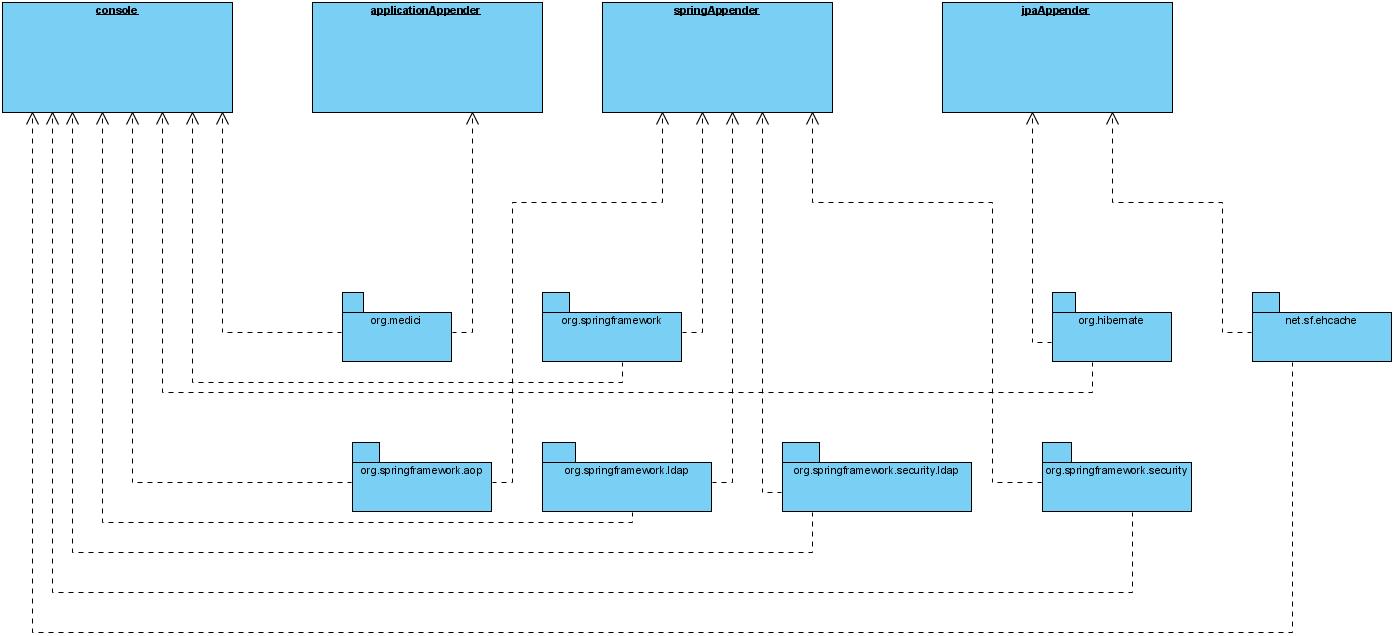
Il logging applicativo è basato sul frame work log4j ed utilizza un file di properties in formato testo semplice, log4j.properties. E’ presente, un root logger (non utilizzato direttamente), un appender di tipo console, e 3 file appender, suddivisi, fra appender applicativo (applicationAppender), appender per le classi spring (springAppender) ed un appender per le operazioni di persistenza (jpaAppender). I logger vengono dichiarati in base ai package di appartenenza e sono tutti agganciati almeno al console Appender.

Sono stati individuati dei package generiche all’interno di tutte le classi applicativi e dei frame work che sono stati sfruttati per definire i livelli di tracing delle varie componenti :

1. Le classi org.medici sono loggate con level DEBUG sull’applicationAppender e sulla console
2. Le classi org.spring.framework sono loggate con level DEBUG sullo springAppender e sulla console ad eccezioni di alcuni sottopackage specifici che hanno una propria configurazione
3. Le classi del org.spring.framework.aop sono loggate con level INFO sullo springAppender e sulla console
4. Le classi del org.spring.framework.ldap sono loggate con level INFO sullo springAppender e sulla console
5. Le classi del org.spring.framework.security.ldap sono loggate con level INFO sullo springAppender e sulla console
6. Le classi del org.spring.framework.security sono loggate con level DEBUG sullo springAppender e sulla console
7. Le classi org.hibernate sono loggate con level INFO sul jpaAppender e' sulla console
8. Le classi net.sf.ehcache sono loggate con level INFO sul jpaAppender e sulla console

Questa illustrata è una configurazione di logging base per permettere un’analisi dei log applicativi durante lo sviluppo del codice. Per quanto riguarda la configurazione una volta che l’applicazione andrà in operativo sui server di produzione, sarà differente e scelta in funzione della stabilità e del carico a cui sarà sottoposta.

Di seguito potete osservare il diagramma ad oggetti della configurazione del log4j :



(Diagramma delle dipendenze fra i packages e i logger applicativi)

# Licenza d’uso dell’applicazione

L’applicazione viene rilasciata sotto licenza GNU GPL versione 2, al termine del progetto verrà richiesta una certificazione per l’omologazione ad essa. E’ quindi necessario, inserire all’interno di ciascuna classe i termini della licenza. L’inserimento va eseguito alla prima riga del file di testo del sorgenti della classe, ed il testo è il seguente :

|  |
| --- |
| /\*  \* <ClassName>.java  \*  \* Developed by Medici Archive Project (2010-2012).  \*  \* This file is part of DocSources.  \*  \* DocSources is free software; you can redistribute it and/or modify  \* it under the terms of the GNU General Public License as published by  \* the Free Software Foundation; either version 2 of the License, or  \* (at your option) any later version.  \*  \* DocSources is distributed in the hope that it will be useful,  \* but WITHOUT ANY WARRANTY; without even the implied warranty of  \* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  \* GNU General Public License for more details.  \*  \* You should have received a copy of the GNU General Public License  \* along with this library; if not, write to the Free Software  \* Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA  \*  \* As a special exception, if you link this library with other files to  \* produce an executable, this library does not by itself cause the  \* resulting executable to be covered by the GNU General Public License.  \* This exception does not however invalidate any other reasons why the  \* executable file might be covered by the GNU General Public License.  \*/ |

# Struttura del progetto

La struttura del progetto si basa sul modello Dynamic Web Project, con i seguenti aspetti :

* Dynamic Web Module (2.5)
* Java (1.6)
* JavaScript (1.0)
* JPA (2.0)

Nella home del progetto sono presenti le seguenti componenti :

|  |  |  |
| --- | --- | --- |
|  |  |  |
| .settings | Directory | Directory delle impostazioni |
| .svn | Directory | Directory contenente le informazioni di versionamento |
| branches | Directory | Directory svn contenente i branch dell’albero head |
| build | Directory | Directory di compilazione dei sorgenti |
| src | Directory | Directory contenente i sorgenti java |
| tags | Directory | Directory svn contenente i tags di versionamento |
| target | Directory | Directory di distribuzione (dove viene generato il war applicativo) |
| trunk | Directory | Directory svn contenente i trunk |
| uml | Directory | Grafici uml applicativi |
| WebContent | Directory | Contenuti web dell’applicazione (front-end + file di configurazioni) |
| .classpath | File | File di eclipse contenente le informazioni di classpath |
| .project | File | File di eclipse contenente le informazioni del progetto |
| pom.xml | File | File descrittore maven per la compilazione e la distribuzione |
|  |  |  |

I sorgenti applicativi sono suddivisi fra la directory src (per le componenti Model & Controller) e WebContent (View).

## File pom.xml

Questo è il file del frame work Maven per gestire la compilazione e la distribuzione del progetto. Il file è già stato configurato per fornire lo scheletro applicativo funzionante. Le sezioni definite nel file di progetto sono :

* Tipologia di packaging (war)
* groupId ed artifactId del prodotto
* Definizione del team che lavora sul progetto
* Licenza adottata nell’applicazione
* Lista dei repository ufficiali sui quali poter effettuare la risoluzione delle dipendenze
* Dichiarazione della procedura di build con definizione dei plugin utilizzati (maven-war-plugin, mave-compiler-plugin, maven-eclipse-plugin)
* Lista di tutte le dipendenze necessarie all’applicazione

## Directory src

All’interno della directory src troviamo due directory :

* main, seguita java, in questa sono presenti una directory META-INF, contenente i descrittori del frame work JPA (orm.xml e persistence.xml) e il primo package applicativo org. Allo stesso livello della directory java, troviamo la directory resources, dove sono memorizzati due file importanti, import.sql per effettuare l’import del database a runtime, e spring-javadoc.css, contenente il layout css per la documentazione javadoc
* test, seguita da java, contiene le classi per effettuare lo unit testing dell’applicazione.

## Descrittori JPA

|  |  |
| --- | --- |
| Quando si progetta un'applicazione di solito è possibile separare gli oggetti persistenti in gruppi indipendenti che possono essere trattati separatamente, magari all'interno di un oggetto DAO diverso, se si utilizzano oggetti DAO.Il framework JPA1 ha introdotto cosi’ il concetto di unità di persistenza. Un’unità di persistenza fornisce un modo conveniente per specificare una serie di file di metadati, e delle classi, e vasetti che contengono tutte le classi ad essere mantenuto in un raggruppamento.  Le unità di persistenza vengono definite all’interno di un file denominato persistence.xml, che deve risiedere nella directory META-INF.  Nell’applicazione DocSources è stata definita un’unica unità di persistenza   |  | | --- | | <?xml version="1.0" encoding="UTF-8"?>  <persistence version="2.0" xmlns="http://java.sun.com/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/persistence\_2\_0.xsd ">  <persistence-unit name="DocSourcesProject" transaction-type="RESOURCE\_LOCAL">  <provider>org.hibernate.ejb.HibernatePersistence</provider>  <properties>  <!-- proprietà per generare lo schema -->  <property name="hibernate.hbm2ddl.auto" value="update"/>  </properties>  </persistence-unit>  </persistence> | |

In questo file descrittore, viene specificato :

* Versione JPA supportata (2.0)
* Nome della persistent unit DocSourcesProject
* Tipologia di transazioni (RESOURCE\_LOCAL)
* Il provider che fornisce le funzionalità di persistenza (org.hibernate.ejb.HibernatePersistence)
* Un properites contenente le proprietà specifiche del provider che servono per inizializzarlo o specificare determinate configurazioni

Il provider scelto per l’applicazione è hibernate versione 3.5.1-Final. A livello di configurazione è stato scelto di specificare soltanto la modalità di gestione dello schema del database. Con l’impostazione di update, quando l’applicazione crea il SessionFactory (all’avvio ndr), il motore di persistence, effettua un check fra il database a cui si connette e la definizione di tutti gli oggetti del domain model, provvedendo ad apportare le dovute modifiche sul database nel caso in cui questo contenga delle tabelle non definite correttamente. Per quanto riguarda la configurazione del database si rimanda al capitolo di gestione dei file di configurazione applicativi.

In questa directory è poi presente il secondo descrittore fondamentale per il jpa, che è il file orm.xml. Tramite questio file è possibile mappare esattamente le varie entità con le corrispettive tabelle sul database. Il frame work fornisce pero’ anche un’altra strada per effettuare questi mapping, l’utilizzo delle java annotation direttamente nelle classi del domain model. Il team di sviluppo ha scelto di sfruttare la seconda strada in quanto l’annotazione del mapping direttamente nel codice java, permette allo sviluppatore di effettuare le verifiche di integrità fra la classe e la corrispettiva tabella sul database piu’ velocemente rispetto al dover aprire un file xml e confrontare i tag xml con il codice java della classe. Per utilizzare le java annontation in modo esclusivo è sufficiente dichiarare il file orm.xml vuoto. Il file deve comunque esistere non è possibile sfruttare il jpa senza la definizione del file (il frame work segnalerebbe un’errore durante la fase di inizializzazione a runtime.

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <entity-mappings xmlns="http://java.sun.com/xml/ns/persistence/orm" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence/orm http://java.sun.com/xml/ns/persistence/orm\_2\_0.xsd"  version="2.0">  </entity-mappings> |

## Directory src/main/java/org/

Questo è il primo package applicativo. Tutte gli oggetti java, risiedono all’interno del package org.medici.docsources, suddivise per macrofunzionalità.

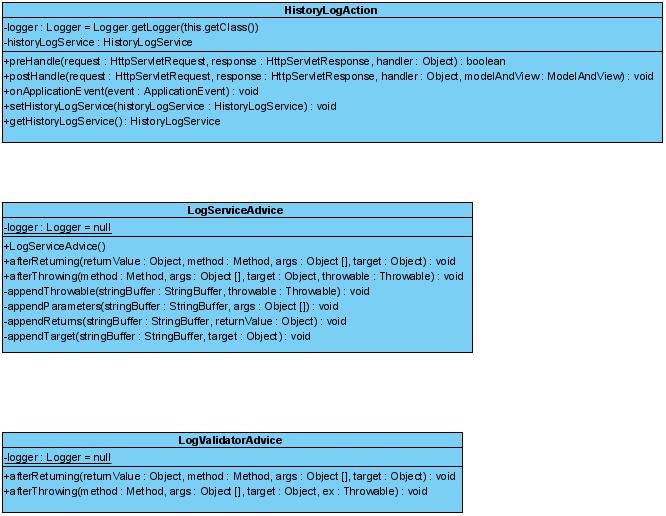
Esistono alcune regole generali da applicare a tutte le classi :

* Utilizzare sempre sintassi in lingua inglese
* Ricordarsi che ogni classe deve avere la licenza d’uso contenente nella seconda riga il nome della classe seguito dall’estensione .java
* Effettuare gli import per classe e non per package
* Eliminare gli import non utilizzati
* Mantenre i metodi di classe ordinati per ordine alfabetico
* Eliminare sempre le istanze di classe che non vengono utilizzate
* Non mischiare classi di un layer con quelle di un altro layer
* Mantenere il modello mvc sempre e comunque
* Documentare il sorgente di classe tramite i javadoc
* Nel javadoc di classe, inserire sempre l’annotation @author con i propri riferimenti se si è lavorato su quella classe.

### org.medici.docsources.audit

In questo package vengono definite tutte le funzionalità applicative di auditing :

* History Log
* Servizio di Logging dei Services
* Servizio di Logging dei Validator



### org.medici.docsources.command

In questo package, suddivisi in un’ulteriore package che identifica il contesto, vengono definiti i command Bean dei Controller MVC, ossia un bean che serve a contenere i parametri della richiesta e che è valorizzato automaticamente dal controller. Ciascun controller deve avere un unico command bean configurato nel seguente modo :

* Il nome del bean deve essere in inglese, non è consentito l’uso di nomenclature in italiano
* Il nome del bean deve essere contestualizzato con l’azione del controller in cui verrà utilizzato. In generale il formato è sempre “azione + contesto + **Command**” (per esempio il command preposto alla memorizzazione delle informazioni di creazione di un utente si chiamerà CreateUserCommand).
* La classe deve essere dichiarata public
* La classe non deve estendere alcuna superclasse differente da Object
* La classe non deve implementare alcuna interfaccia
* Le proprietà definite al suo interno devono sempre alla categoria Object, è proibito utilizzare i tipi primitivi (per evitare problemi nel processo di binding effettuato da spring)
* Ogni proprietà deve essere privata, ed aderire alla specifica JavaBean :
  + Il nome della proprietà deve essere in inglese, non è permesso l’uso di nomenclature in italiano
  + Il nome della proprietà deve iniziare per lettera minuscola e nel caso in cui sia composto da piu’ parole, le successive devono iniziare per lettera maiuscola. L’uso di caratteri di underscore è consentito solo in caso proprietà che referenziano l’uso specifico di frame work (per esempio per la creazione di command che devono mappare richieste di librerie ajax già esistenti che forniscono un modello ben definito per i nomi dei parametri da passare)
  + deve presentare sempre un metodo di set che riceve un parametro in ingresso del tipo di classe uguale a quella di definizione della proprietà
  + deve presentare quasi sempre un metodo di get che restituisce un’oggetto di classe uguale a quella di definizione della proprietà. L’eccezione di quasi sempre è riservata agli oggetti Boolean che non devono avere questo metodo
  + Nel caso in cui la proprietà sia un booleano, deve avere il metodo isNomeProprietà al posto del metodo di get e restituire un oggetto Boolean
* Se una proprietà ha dei vincoli di validazione base (controllo sulla dimensione, campo obbligatorio), questi devono essere definiti al suo interno tramite le annotazioni di validazione

@NotNull

@Size(min=4, max=15)

**private** String account;

Per maggiori dettagli su tutte le validazioni case che si possono inserire si rimanda alla documentazione jdk 6

<http://download.oracle.com/javaee/6/api/javax/validation/constraints/package-summary.html>

* E’ permesso l’implementazione del metodo toString()
* Non è permessa la dichiarazione esplicita di logger applicativi

Esempio di un CommandBean per la maschera di registrazione di un’utente :

|  |
| --- |
| **package** org.medici.docsources.command.user;  **import** javax.validation.constraints.NotNull;  **import** javax.validation.constraints.Size;  /\*\*  \*  \* **@author** Lorenzo Pasquinelli (<a href=mailto:l.pasquinelli@gmail.com>l.pasquinelli@gmail.com</a>)  \*  \*/  **public** **class** RegisterCommand {  @NotNull  @Size(min=8, max=15)  **private** String account;  @NotNull  **private** String firstName;  **private** String countryDescription;  /\*@NotNull  @Size(min=2, max=2)\*/  **private** String countryCode;  @NotNull  **private** String password;  @Size(min=8, max=15)  **private** String confirmPassword;  **private** String recaptcha\_challenge\_field;  **private** String recaptcha\_response\_field;  **private** String ip;    /\*\*  \* **@return** the recaptcha\_response\_field  \*/  **public** String getRecaptcha\_response\_field() {  **return** recaptcha\_response\_field;  }  /\*\*  \* **@param** recaptcha\_response\_field the recaptcha\_response\_field to set  \*/  **public** **void** setRecaptcha\_response\_field(String recaptcha\_response\_field) {  **this**.recaptcha\_response\_field = recaptcha\_response\_field;  }  **public** **void** setAccount(String account) {  **this**.account = account;  }    **public** String getAccount() {  **return** account;  }    **public** **void** setPassword(String password) {  **this**.password = password;  }    **public** String getPassword() {  **return** password;  }  **public** **void** setFirstName(String firstName) {  **this**.firstName = firstName;  }    **public** String getFirstName() {  **return** firstName;  }  /\*\*  \* **@param** countryCode the countryCode to set  \*/  **public** **void** setCountryCode(String countryCode) {  **this**.countryCode = countryCode;  }  /\*\*  \* **@return** the countryCode  \*/  **public** String getCountryCode() {  **return** countryCode;  }  /\*\*  \* **@param** confirmPassword the confirmPassword to set  \*/  **public** **void** setConfirmPassword(String confirmPassword) {  **this**.confirmPassword = confirmPassword;  }  /\*\*  \* **@return** the confirmPassword  \*/  **public** String getConfirmPassword() {  **return** confirmPassword;  }  /\*\*  \* **@param** countryDescription the countryDescription to set  \*/  **public** **void** setCountryDescription(String countryDescription) {  **this**.countryDescription = countryDescription;  }  /\*\*  \* **@return** the countryDescription  \*/  **public** String getCountryDescription() {  **return** countryDescription;  }  /\*\*  \* **@param** recaptcha\_challenge\_field the recaptcha\_challenge\_field to set  \*/  **public** **void** setRecaptcha\_challenge\_field(String recaptcha\_challenge\_field) {  **this**.recaptcha\_challenge\_field = recaptcha\_challenge\_field;  }  /\*\*  \* **@return** the recaptcha\_challenge\_field  \*/  **public** String getRecaptcha\_challenge\_field() {  **return** recaptcha\_challenge\_field;  }  /\*\*  \* **@param** ip the ip to set  \*/  **public** **void** setIp(String ip) {  **this**.ip = ip;  }  /\*\*  \* **@return** the ip  \*/  **public** String getIp() {  **return** ip;  }  } |

### org.medici.docsources.command.docbase

In questo package sono memorizzati tutti i command bean che vengono utilizzati dai controller preposti a lavorare sul sottomodulo doc (document *ndr*) del modulo data entry.

### org.medici.docsources.command.geobase

In questo package sono memorizzati tutti i command bean che vengono utilizzati dai controller preposti a lavorare sul sottomodulo geo (geographic *ndr*) del modulo data entry.

### org.medici.docsources.command.login

In questo package sono memorizzati tutti i command bean che vengono utilizzati dai controller preposti a lavorare sul modulo di autenticazione.

### org.medici.docsources.command.peoplebase

In questo package sono memorizzati tutti i command bean che vengono utilizzati dai controller preposti a lavorare sul sottomodulo people del modulo data entry.

### org.medici.docsources.command.user

In questo package sono memorizzati tutti i command bean che vengono utilizzati dai controller preposti a lavorare sull’anagrafica utente.

### org.medici.docsources.command.volbase

In questo package sono memorizzati tutti i command bean che vengono utilizzati dai controller preposti a lavorare sul sottomodulo vol (volume *ndr*) del modulo data entry.

### org.medici.docsources.common.ajax

In questo package troviamo alcune classi di utility per funzionalità ajax.

### org.medici.docsources.common.util

In questo package troviamo alcune classi di utility.

### org.medici.docsources.controller

In questo package, suddivisi in un’ulteriore package che identifica il contesto, vengono definiti i controller MVC preposti alla ricezione delle richieste ed alla fornitura delle risposte. In ciascun package saranno quindi disponibili n controller per n azioni applicativi, ed un controller aggiuntivo preposto alla gestione delle richieste provenienti dalle richieste AJAX. In questa prima fase ci focalizzeremo all’analisi della scrittura di un controller non AJAX, aventi le seguenti caratteristiche :

* Il package dove risiede deve essere contestualizzato alla funzionalità

Per esempio nel caso di un controller preposto alla gestione della maschera di creazione di un’utente, il package dovrà essere :

**package** org.medici.docsources.controller.user;

* Il nome del controller deve essere in inglese, non è consentito l’uso di nomenclature in italiano. La nomenclatura deve sempre essere generata usando la sintassi azione che il controller implementa, seguita dal nome dell’oggetto su cui l’azione è eseguita e deve terminare con il suffisso Controller :

1. I controller che visualizzano un oggetto devono utilizzare, come identificativo di azione, Show
2. I controller che creano un’oggetto un contenuto devono utilizzare, come identificativo di azione, Create
3. I controller che modificano un’oggetto un contenuto devono utilizzare, come identificativo di azione, Modify
4. I controller che cancellano un’oggetto un contenuto devono utilizzare, come identificativo di azione, Delete
5. I controller che forniscono funzionalità di ricerca semplice (per alias ndr) di un’oggetto devono utilizzare l’identificativo Search
6. I controller che forniscono funzionalità di ricerca avanzata di un’oggetto devono utilizzare l’identificativo AdvancedSearch

* Non è concesso l’uso di nomenclature che riportano riferimenti a nuove versioni di controller pre-esistenti
* La classe deve essere dichiarata public
* La classe non deve estendere una classe che sia differente da Object
* La classe non deve implementare alcuna interfaccia
* La classe deve definire prima del nome di classe, l’annotation @Controller e l’annotation @RequestMapping che comunica il path che assolve.Questa regola non viene applicata ai controllere ajax. L’estensione del path nel valore dell’annotation @RequestMapping non dovrà mai essere specificata, al fine di garantire al frame work la possibilità di gestire la view in funzione del content richiesto.

@Controller

@RequestMapping("/user/create")

**public** **class** CreateController {

* Nel caso in cui il controller debba fornire la maschera di inserimento dati, verrà implementato un metodo setupForm (in GET ndr) che dovrà essere utilizzato per l’estrazione di eventuali oggetti dal database che devo essere impiegati per valorizzare campi della maschera (combo box, parametri nascosti). La firma di questo metodo deve prevedere la restituzione di un’oggetto di tipo ModelAndView, e la ricezione di un commandBean per i paramentri in input. Se il form prevede dei campi in input che devono essere preventivamente validati, i campi vanno inseriti in un’appositivo commandBean che si deve chiamare con il suffisso RequestCommand, questo bean inoltre deve essere impostato specificando l’annotation ModelAttribute con attribute "requestCommand". Se la richiesta del form non prevede parametri in ingresso si puo’ passare direttamente il command bean usato nel metodo di elaborazione del Form POST.

Esempio di richiesta form con validazione dei parametri

@RequestMapping(method = RequestMethod.*GET*)

**public** ModelAndView setupForm(@Valid @ModelAttribute("requestCommand") ResetUserPasswordRequestCommand command, BindingResult result) {

Esempio di richiesta form che non prevede parametri in input

@RequestMapping(method = RequestMethod.*GET*)

**public** ModelAndView setupForm(@ModelAttribute SearchUserCommand searchUserCommand);

* Il controller fornisce l’elaborazione della richiesta soltanto in POST. La firma di questo metodo è identica per tutti i metodi e restituisce un’oggetto ModelAndView che punta alla View da renderizzazione sul client

@RequestMapping(method = RequestMethod.*POST*)

**public** ModelAndView processSubmit(@Valid @ModelAttribute("command") RegisterCommand command, BindingResult result) {

Rispetto alla firma in GET, questa presenta una maggiore complessità e alcuni parametri opzionali,

L’annotation @Valid deve essere inserita nel caso in cui il command bean contenga delle validazioni base delle sue proprietà.

* Se un controller referenzia il validator per l’elaborazione del form, il validator andrà definito con un autowiring di questo tipo :

@Autowired(required = **false**)

@Qualifier("createUserValidator")

**private** Validator validator;

Il validator dovrà essere logicamente definito con questo id nel file di configurazione dei validators

applicativi.

* Se un controller referenzia sia il validator per l’elaborazione del form che quello di richiesta form, i due validators andranno definiti con un autowiring di questo tipo :

@Autowired(required = **false**)

@Qualifier("resetUserPasswordRequestValidator")

**private** Validator requestValidator;

@Autowired(required = **false**)

@Qualifier("resetUserPasswordValidator")

**private** Validator validator;

I due validator dovranno essere logicamente definito con questo id nel file di configurazione dei

validators applicativi.

Nel caso in cui l’elaborazione necessiti di validazioni logiche, o di integrità, queste devono essere inserite in una classe validator specifica per il controller (si rimanda al paragrafo del package org.medici.docsources.validators), e tale classe deve essere iniettata nel controller mediante autowiring.

**Quando si utilizza un validator, al fine di permettere a Spring l’esecuzione del corretto flusso, è necessario inserire l’attributo @Valid sul command bean che deve essere inoltre sempre agganciato all’MVC mediante l’annontation @ModelAttribute , è inoltre indispensabile il passaggio dell’oggetto BindingResult per poter recuperare nella View i messaggi di errore di validazione. Se si definisce l’attribute di modello come @Valid è obbligatorio passare il parametro subito dopo il command altrimenti non sarà possibile recuperare i messaggi. Il recupero dei messaggi lato View, puo’ essere fatto in automatico definendo nel codice il tag <form:form method="post"> e recuperando all’interno i vari <form:errors /> delle proprietà del command.**

### org.medici.docsources.controller.docbase

In questo package sono memorizzati tutti i controller preposti alla ricezione delle richieste inerenti al sotto modulo doc (document *ndr*) del modulo dataentry.

### org.medici.docsources.controller.geobase

In questo package sono memorizzati tutti i controller preposti alla ricezione delle richieste inerenti al sotto modulo geo (geographic *ndr*) del modulo dataentry.

### org.medici.docsources.controller.peoplebase

In questo package sono memorizzati tutti i controller preposti alla ricezione delle richieste inerenti al sotto modulo people del modulo dataentry.

### org.medici.docsources.controller.user

In questo package sono memorizzati tutti i controller preposti alla ricezione delle richieste inerenti all’anagrafica utente.

Ad oggi sono stati sviluppati (non tutti sono funzionanti, alcuni sono solo in formato scheletro) i seguenti controller :

* AjaxController, controller per l’eleborazioni di richieste delle componenti applicative ajax
* CreateUserController, Controller per la creazione di un utente, funzionalità amministrativa
* LockUserController, Controller per effettuare il blocco operativo di un utente
* ModifyUserController, Controller per effettuare le modifiche di anagrafica di un utente dal modulo amministrativo
* RegisterUserController, Controller per permettere ad un nuovo utente di registrarsi
* SearchUserController, Controller per effettuare la ricerca sull’anagrafica utente
* SendUserOrPasswordController, Controller per recuperare le informazioni via mail
* ShowUserPhotoController, Controller che visualizza la foto dell’utente
* UnregisterUserController, Controller che permette la cancellazione di un utente
* UpdateUserController, Controller che permette l’aggiornamento dei dati utente.
* UpdateUserPasswordController, Controller che permette l’aggiornamento della password dell’utente.

### org.medici.docsources.controller.volbase

In questo package sono memorizzati tutti i controller preposti alla ricezione delle richieste inerenti al sotto modulo doc (document *ndr*) del modulo dataentry.

### org.medici.docsources.dao

In questo package, suddivisi in un’ulteriore package che identifica il contesto, vengono definite le classi dao preposte al recupero delle entità.

Le classi DAO rappresentano la modellazione a oggetti di componenti che gestiscono e nascondono agli altri “layer” applicativi le tematiche di persistenza degli oggetti.

Architetturalmente si è scelto di avere una metodologia che fornisca allo sviluppatore un’insieme di strumenti performanti e ben organizzati per permettergli di scrivere codice snello, intuitivo e di chiara interpretazione. Per fare questo è stato deciso di seguire un modello di associazione fra le entità e il dao :

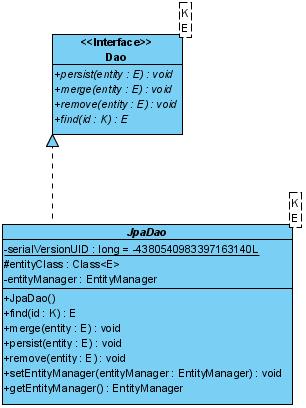
Ciascuna entità deve avere un proprio dao di accesso

Ad oggi, la base dati è composta di 24 tabelle, relazionate fra di loro tramite opportune foreign keys, ed in base a quanto enunciato sopra, la composizione di accesso è composta da 24 package differenziati per entità, ognuno dei quali fornisce l’accesso ad una specifica entità :

1. org.medici.docsources.dao.altname
2. org.medici.docsources.dao.bibliot
3. org.medici.docsources.dao.bioreflink
4. org.medici.docsources.dao.country
5. org.medici.docsources.dao.document
6. org.medici.docsources.dao.eplink
7. org.medici.docsources.dao.epltolink
8. org.medici.docsources.dao.factchecks
9. org.medici.docsources.dao.historylog
10. org.medici.docsources.dao.jewishdocument
11. org.medici.docsources.dao.marriage
12. org.medici.docsources.dao.month
13. org.medici.docsources.dao.people
14. org.medici.docsources.dao.place
15. org.medici.docsources.dao.polink
16. org.medici.docsources.dao.prclink
17. org.medici.docsources.dao.researcher
18. org.medici.docsources.dao.rolecat
19. org.medici.docsources.dao.serieslist
20. org.medici.docsources.dao.synextract
21. org.medici.docsources.dao.titleocclist
22. org.medici.docsources.dao.topicslist
23. org.medici.docsources.dao.user
24. org.medici.docsources.dao.volume

Accedere ad un’entità sul database, significa avere una classe che fornisce dei metodi che permettono di effettuare operazioni basilari (scrittura di un nuovo record, aggiornamento per chiave primaria, cancellazione per chiave primaria, ricerca per chiave primaria) ed altre operazioni piu’ complesse specifiche del contesto dell’entità (interrogazioni incrociate con altre entità, interrogazioni per insieme di chiavi).

Tenuto conto che con ventiquattro dao le operazioni basilari sono novantasei, se i dao venissero scritti seguendo il modello standard jpa, otterremmo 24 classi contenenti molto codice ridondato. Per cercare di armonizzare questo layer, è stato deciso di sviluppare un modello generale basato su di una classe astratta JpaDAO (corredata di una specifica interfaccia) generica, che fornisce le operazioni basi; questa classe viene poi estesa da ciascun specifico DAO .



Di seguito viene mostrato il codice dell’interfaccia e della relativa classe astratta :

|  |
| --- |
| **package** org.medici.docsources.dao;  **import** java.io.Serializable;  **import** javax.persistence.PersistenceException;  /\*\*  \*  \* **@param** <K>  \* **@param** <E>  \*  \* **@author** Lorenzo Pasquinelli (<a href=mailto:l.pasquinelli@gmail.com>l.pasquinelli@gmail.com</a>)  \*  \*/  **public** **interface** Dao<K, E> **extends** Serializable {  **public** **void** persist(E entity) **throws** PersistenceException;  **public** **void** merge(E entity) **throws** PersistenceException;  **public** **void** remove(E entity) **throws** PersistenceException;  **public** E find(K id) **throws** PersistenceException;  } |

(Interfaccia Dao generica)

Come è possibile osserva dal codice dell’interfaccia, ogni Dao fornirà quattro metodi base, che permettono le operazioni di :

1. Persistenza di un’entità
2. Aggiornamento di un’entità
3. Cancellazione di un’entità
4. Metodo di ricerca per chiave primaria dell’entità

|  |
| --- |
| **package** org.medici.docsources.dao;  **import** java.lang.reflect.ParameterizedType;  **import** javax.persistence.EntityManager;  **import** javax.persistence.PersistenceContext;  **import** javax.persistence.PersistenceException;  **import** org.springframework.stereotype.Repository;  @Repository  **public** **abstract** **class** JpaDao<K, E> **implements** Dao<K, E> {  /\*\*  \*  \*/  **private** **static** **final** **long** *serialVersionUID* = -4380540983397163140L;  **protected** Class<E> entityClass;    @PersistenceContext  **private** EntityManager entityManager;    @SuppressWarnings("unchecked")  **public** JpaDao() {  ParameterizedType genericSuperclass = (ParameterizedType) getClass().getGenericSuperclass();  **this**.entityClass = (Class<E>) genericSuperclass.getActualTypeArguments()[1];  }    **public** E find(K id) **throws** PersistenceException { **return** getEntityManager().find(entityClass, id); }  **public** **void** merge(E entity) **throws** PersistenceException { getEntityManager().merge(entity); }  **public** **void** persist(E entity) **throws** PersistenceException{ getEntityManager().persist(entity); }    **public** **void** remove(E entity) **throws** PersistenceException { getEntityManager().remove(entity); }    **public** **void** setEntityManager(EntityManager entityManager) {  **this**.entityManager = entityManager;  }  **public** EntityManager getEntityManager() {  **return** entityManager;  }  } |

(Classe DAO astratta)

L’utilizzo dell’annotation @Repository serve per specificare il ruolo di repository e per gestire la il la trasformazione di eventuali eccezioni rilanciate dai metodi di questo layer quando raggiungono lo strato dei service.

Il JPADao viene dichiato utilizzando un generics type composito, formato da una chiave generica <K> e da un’entità <E>. Questa associazione viene dichiarata al momento della scrittura del specifico e mira a creare un’associazione diretta fra il DAO e l’entità. La chiave generica permette di considerare già in fase di sviluppo la chiave primaria dell’entità permettendo cosi’ di fornire un metodo di find specifico per chiave primaria.

Per quanto riguarda le regole di dichiarazione di un dao, queste sono :

1. Il package dove risiede deve essere contestualizzato all’entità di cui è specializzato

Supponiamo di avere un’entità che si chiama AltName, il package di definizione deve essere posizionato dentro org.medici.dao e contenere il nome dell’entità in caratteri minuscoli :

org.medici.docsources.dao.altname

1. Ogni Dao deve essere composto da un’interfaccia pubblica e da una classe di implementazione pubblica. Il nome dell’interfaccia deve essere composto dal nome dell’entità seguito dalla parola DAO. Per quanto riguarda l’implementazione, il nome deve contenere il nome dell’entità, seguito dalla parola DAO, seguito dal nome del frame work tramite il quale si effettuerà l’accesso all’entità (attualmente sono previste le dicitura Jpa per l’accesso alla base dati e Ldap per l’accesso all’albero ldap).

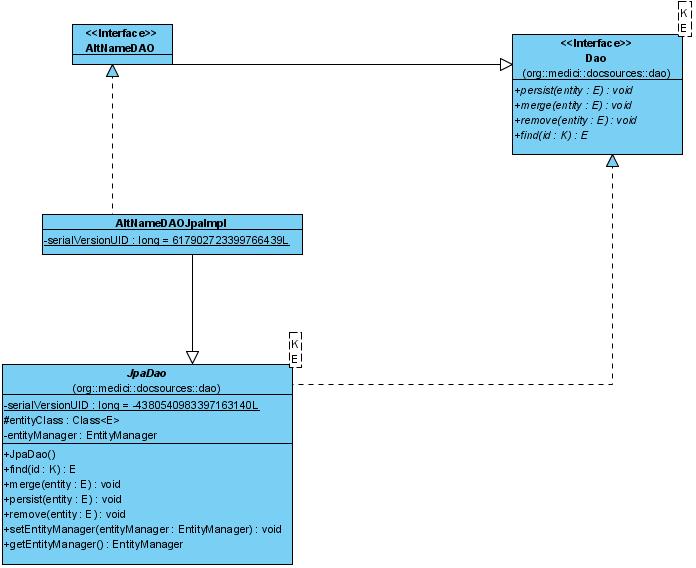
Supponiamo di avere un’entità che si chiama AltName, le due classi saranno :

* 1. AltNameDAO
  2. AltNameDAOJpaImpl

1. Il nome dell’implementazione deve essere in inglese, non è consentito l’uso di nomenclature in italiano
2. L’interfaccia DAO se è contestualizzata ad un’entità accessibile via JPA, deve essere dichiarata in una classe pubblica e deve estendere l’interfaccia *org.medici.docsources.dao.Dao* con la definizione dell’associazione chiave – entità
3. L’interfaccia DAO se è contestualizzata ad un’entità accessibile via LDAP, deve essere dichiarata in una classe pubblica e non deve estendere altre interfacce
4. La classe di implementazione deve estendere la classe astratta *org,medici.docsources.dao.JpaDao* con la definizione dell’associazione chiave – entità, ed implementare la propria interfaccia di riferimento
5. La classe di implementazione deve dichiarare il campo privato
6. La classe di implementazione deve dichiarare l’annotation @Repository
7. Non è concessa la definizione di classi DAO che non fanno riferimento ad una specifica interfaccia

### org.medici.docsources.dao.altname

In questo package è localizzato il dao di accesso all’entità AltName. Attualmente le funzionalità presenti sono quelle basi del dao JpaDao.



### org.medici.docsources.dao.bibliot

In questo package è localizzato il dao di accesso all’entità BiblioT. Attualmente le funzionalità presenti sono quelle basi del dao JpaDao.

### org.medici.docsources.dao.bioreflink

In questo package è localizzato il dao di accesso all’entità BioRefLink. Attualmente le funzionalità presenti sono quelle basi del dao JpaDao.

### org.medici.docsources.dao.country

In questo package è localizzato il dao di accesso all’entità Country. Attualmente le funzionalità presenti sono quelle basi del dao JpaDao.

### org.medici.docsources.dao.document

In questo package è localizzato il dao di accesso all’entità Document. Attualmente le funzionalità presenti sono quelle basi del dao JpaDao.

### org.medici.docsources.dao.eplink

In questo package è localizzato il dao di accesso all’entità EpLink. Attualmente le funzionalità presenti sono quelle basi del dao JpaDao.

### org.medici.docsources.dao.epltolink

In questo package è localizzato il dao di accesso all’entità EplToLink. Attualmente le funzionalità presenti sono quelle basi del dao JpaDao.

### org.medici.docsources.dao.factchecks

In questo package è localizzato il dao di accesso all’entità FatChecks. Attualmente le funzionalità presenti sono quelle basi del dao JpaDao.

### org.medici.docsources.dao.historylog

In questo package è localizzato il dao di accesso all’entità HistoryLog. Attualmente le funzionalità presenti sono quelle basi del dao JpaDao.

### org.medici.docsources.dao.marriage

In questo package è localizzato il dao di accesso all’entità Marriage. Attualmente le funzionalità presenti sono quelle basi del dao JpaDao.

### org.medici.docsources.dao.month

In questo package è localizzato il dao di accesso all’entità Month. Attualmente le funzionalità presenti sono quelle basi del dao JpaDao.

### org.medici.docsources.dao.peole

In questo package è localizzato il dao di accesso all’entità People. Attualmente le funzionalità presenti sono quelle basi del dao JpaDao.

### org.medici.docsources.dao.place

In questo package è localizzato il dao di accesso all’entità Place. Attualmente le funzionalità presenti sono quelle basi del dao JpaDao.

### org.medici.docsources.dao.polink

In questo package è localizzato il dao di accesso all’entità PoLink. Attualmente le funzionalità presenti sono quelle basi del dao JpaDao.

### org.medici.docsources.dao.prclink

In questo package è localizzato il dao di accesso all’entità PrcLink. Attualmente le funzionalità presenti sono quelle basi del dao JpaDao.

### org.medici.docsources.dao.researcher

In questo package è localizzato il dao di accesso all’entità Reseracher. Attualmente le funzionalità presenti sono quelle basi del dao JpaDao.

### org.medici.docsources.dao.rolecat

In questo package è localizzato il dao di accesso all’entità RoleCat. Attualmente le funzionalità presenti sono quelle basi del dao JpaDao.

### org.medici.docsources.dao.serieslist

In questo package è localizzato il dao di accesso all’entità SeriesList. Attualmente le funzionalità presenti sono quelle basi del dao JpaDao.

### org.medici.docsources.dao.synextract

In questo package è localizzato il dao di accesso all’entità SynExtract. Attualmente le funzionalità presenti sono quelle basi del dao JpaDao.

### org.medici.docsources.dao.titleocclist

In questo package è localizzato il dao di accesso all’entità TitleOccList. Attualmente le funzionalità presenti sono quelle basi del dao JpaDao.

### org.medici.docsources.dao.topicslist

In questo package è localizzato il dao di accesso all’entità TopicList. Attualmente le funzionalità presenti sono quelle basi del dao JpaDao.

### org.medici.docsources.dao.user

In questo package è localizzato il dao di accesso all’entità User. Questo è un dao con accesso all’albero ldap dell’anagrafica utenti, non estende il dao astratto JPA. Attualmente sono disponibili i metodi di ricerca utente e gruppo di appartenenza, oltre ai metodi per effettuare il salvataggio, l’aggiornamento e la cancellazione di un utente.

### org.medici.docsources.dao.volume

In questo package è localizzato il dao di accesso all’entità Volume. Attualmente le funzionalità presenti sono quelle basi del dao JpaDao.

### org.medici.docsources.domain

Il package di domain model è il fulcro dell’associazione fra le entità logiche ad oggetti e la base dati. Infatti all’interno di questo package sono memorizzate le entità persistenti dell’applicazione.

### org.medici.docsources.dto

In questo package sono contenute le classi applicative che gestiscono le eccezioni rilanciate dai vari service layer applicativi.

### org.medici.docsources.exception

In questo package sono contenute le classi applicative che gestiscono le eccezioni rilanciate dai vari service layer applicativi.

### org.medici.docsources.security

In questo package sono memorizzate le classi applicative che si integrano con il frame work Spring Security.

### org.medici.docsources.service

### org.medici.docsources.service.docbase

In questo package sono contenute le classi di business per l’esposizione dei metodi di lavoro sul sotto modulo doc (document *ndr*) del modulo data-entry.

### org.medici.docsources.service.geobase

In questo package sono contenute le classi di business per l’esposizione dei metodi di lavoro sul sotto modulo geo (geographic *ndr*) del modulo data-entry.

### org.medici.docsources.service.historylog

In questo package sono contenute le classi di business preposte a fornire i servizi di historylog.

### org.medici.docsources.service.mail

In questo package sono contenute le classi di business preposte all’invio delle email per effettuare il recupero delle informazioni di autenticazione dell’utente.

### org.medici.docsources.service.peoplebase

In questo package sono contenute le classi di business per l’esposizione dei metodi di lavoro sul sotto modulo people del modulo data-entry.

### org.medici.docsources.service.recaptcha

In questo package sono contenute le classi di business preposte alla gestione dei processi antispam applicativi per le funzionalità di registrazione ed aggiornamento dati utente.

### org.medici.docsources.service.user

In questo package sono contenute le classi di business che forniscono accesso all’anagrafica utente.

### org.medici.docsources.service.volbase

In questo package sono contenute le classi di business per l’esposizione dei metodi di lavoro sul sotto modulo vol (volume *ndr*) del modulo data-entry.

### org.medici.docsources.support

In questo package sono memorizzate le classi di supporto applicativo.

### org.medici.docsources.support.recaptcha

In questo package sono memorizzate le classi di supporto per le funzionalità di recpatcha. C’e’ soltanto una classe, denominata ReCaptchaConfiguration, che si occupa di memorizzare la configurazione della componente ReCaptcha.

### org.medici.docsources.validator

In questo package vengono posizionate tutte le classi di validazione (suddivise per categorie). Ciascun controller che prevede una validazione dei dati in ingresso, deve referenziare una classe Validator specifica.

Il nome della classe deve contenere, l’azione specificata nel commandBean su cui effettua la validazione di cui sarà competente e dal suffisso Validator. Nel caso in cui un validator si occupi della validazione di un form, il suo nome dovrà contenere la parola Request prima del suffisso Validator.

Il codice applicativo di un Validator è molto semplice, deve implementare l’interfaccia *org.springframework.validation.Validator*, questa interfaccia contiene due metodi pubblici, *public boolean supports(Class givenClass)* e *public void validate(Object object, Errors errors)*. Il primo metodo dichiara la tipologia di bean che è in grado di validare, mentre il secondo metodo contiene il codice di validazione vero e proprio. A livello di frame work Spring, questa tipologia di oggetti è stata pensata in modo che fosse fosse slegata da un particolare layer applicativo in quanto il processo di validazione

Ciascuna classe ha una relazione diretta 1:1 con il relativo controller che la utilizza. Ogni validator viene usato da un controller specifico. Ogni controller puo’ referenziare zero, uno o due classi di validazione. Le regole di referenzia mento del controller sono queste :

* Se un controller non deve validare dati non sarà collegato a nessun Validator
* Se un controller deve validare i dati di elaborazione di un form, referenzierà un Validator
* Se un controller deve validare sia la richiesta di visualizzazione di un form, che la sua elaborazione, referenzierà due Validator (i cui nomi di definizioni saranno ActionObjectNameValidator e ActionObjectNameRequestValidator.

Quando si definisce un nuovo validator, è obbligatorio inserire la sua definizione nel file applicativo di defizione dei validators applicativi (applicationValidators.xml) utilizzando come identificativo del bean il nome della classe modificando il primo carattere nel corrispettivo minuscolo.

### org.medici.docsources.validator.docbase

In questo package sono memorizzati tutte le classi che effettuano la validazione del sotto modulo doc (document ndr) del modulo data entry.

### org.medici.docsources.validator.geobase

In questo package sono memorizzati tutte le classi che effettuano la validazione del sotto modulo doc (geographic *ndr*) del modulo data entry.

### org.medici.docsources.validator.peoplebase

In questo package sono memorizzati tutte le classi che effettuano la validazione del sotto modulo people del modulo data entry.

### org.medici.docsources.validator.user

In questo package sono memorizzati tutte le classi che effettuano la validazione dei processi che lavorano sull’anagrafica utente.

### org.medici.docsources.validator.volbase

In questo package sono memorizzati tutte le classi che effettuano la validazione del sotto modulo vol (volume *ndr*) del modulo data entry.

## Directory src/test/java

In questa directory vengono memorizzate tutte le unit test per i test funzionali applicativi.

## Directory src/test/resources

In questa directory vengono memorizzate le risorse aggiuntive dell’applicazione utili ai fini dei test funzionali.

## Directory src/main/resources

In questa directory vengono memorizzate le risorse aggiuntive dell’applicazione che in fase di packaging devono essere inserite nel classpath.

All’interno troverete un file import.sql, contenente le istruzioni sql di inizializzazione di alcune tabelle. Il motore di persistenza Hibernate permette, nel caso in cui sia stata configurata la proprietà hibernate.hbm2ddl.auto con il valore create oppure con il valore create-drop, di lanciare uno script sql che deve essere posizionato in testa al classpath applicativo. Dato l’applicazione in sviluppo è una web-application, tale file deve essere inserito nella directory WEB-INF/classes del war di distribuzione.

## Directory WebContent

La directory WebContent è la base delle componenti di presentazione e di configurazione dell’applicazione, al suo interno troviamo, i file di configurazione, le pagine jsp, i template tiles, i file degli stili, le risorse javascript e le librerie applicativie

### /WebContent/images/

Conteniene le immagini utilizzate per costruire il front-end dell’applicazione web.

### /WebContent /META-INF/

### /WebContent/scripts/

In questa directory vengono memorizzati i file contenenti il codice javascript utilizzato dal layer di presentazione.

### /WebContent/styles/

In questa directory vengono memorizzati tutti i file di stili che l’applicazione utilizza nella costruzione del lay di presentazione.

### /WebContent/WEB-INF/

### /WebContent/WEB-INF/config/

All’interno di questa directory sono memorizzati tutti i file di configurazione applicativi ad eccezione dei file di definizione del container JPA (che devono essere obbligatoriamente posizionati nella directory META-INF dei sorgenti java).

### /WebContent/WEB-INF/jsp/

All’iinterno di questa directory sono memorizzate tutte le pagine di front-end. Questi oggetti non sono direttamente invocabili dai client degli utenti ma vengono gestite tramite i Controllers ed i template di Tiles. Questo meccanismo di privatizzazione delle pagine di front-end permette una semplice integrazione con il frame work SpringSecurity al fine di evitare di inserire controlli manuali nelle pagine sulle autorizzazioni di cui dispone l’utente che richiede la pagina.

L’home directory contiene i file generici, footer.jspf, head.jsp, header.jspf, home.jsp, login.jsp e menu.jspf.

Tutte le altre jsp sono memorizzate in sotto directory suddivise in base alla funzionalità che assolvono.

### /WebContent/WEB-INF/lib/

Questa directory viene popolata al momento della distribuzione dell’applicazione con le librerie che sono referenziate dalla web-application. Questa operazione viene effettuata automaticamente dal maven. Sul repository svn viene memorizzata vuota.

### /WebContent/WEB-INF/templates/

All’interno di questa directory vengono memorizzati template di Tiles definiti per il front-end applicativo.

# Creazione del database a runtime

Il motore di persistenza Hibernate mette a disposizione una funzionalità per gestire il processo database definition language durante la fase di startup del persistence engine. Questa funzionalità è pilotata dalla proprietà (definita nel persistence.xml ndr) hibernate.hbm2ddl.auto, che puo’ assumere uno fra i seguenti valori :

* validate, effettua la validazione fra lo schema già esistente e quello definito nelle classi di persistenza;
* update, aggiorna lo schema nel caso in cui le classi di persistenza presentino dei campi non allineati;
* create, crea il database alla creazione della SessionFactory;
* create-drop, crea il database alla creazione della SessionFactory e quando la SessionFactory viene distrutta (allo shutdown applicativo), effettua il drop del database.

Per effettuare la creazione del database allo startup, è sufficiente impostare la proprietà con uno dei due valori fra create e create-drop e ricordarsi che lo script import.sql deve obbligatoriamente trovarsi nella root del classpath applicativo.

# Data Entry Module

In questo paragrafo vengono elencati le metodologie di sviluppo del modulo Data Entry

## VolBase module

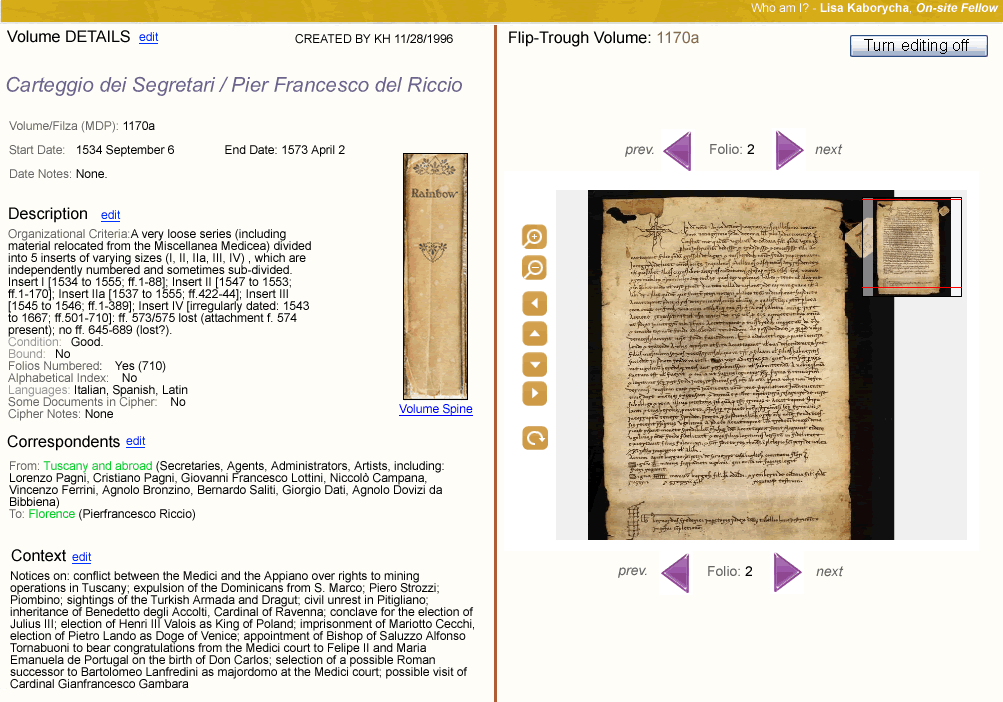
Questo modulo si occupa della gestione dei volumi.

### Creazione di un nuovo volume

La creazione di un nuovo documento è un’azione che si trova nel menu **Entry Menu**.



Quest’azione scatena l’invocazione al flusso applicativo /de/volbase/CreateVolume.do, il quale provvederà a costruire una pagina di visualizzazione del documento suddiviso per sezioni.



In questa View, l’unica azione possibile sarà la selezione del pulsante Edit relativo alla sezione Volume Details. Tutti gli altri pulsanti edit, non saranno attivi fino a quando non verrà effettuato l’editing del primo.

La pressione del tasto edit sulla prima sezione, scatenerà una richiesta al server per ottenere l’edit form del volume con id vuoto. Il server nell’elaborare la richiesta provvederà ad effettuare una scelta se estrarre il volume dal database oppure presentare un form vuoto. Questa sceltà sarà discriminata dalla valorizzazione del campo id. La view di risposta alla richiesta di elaborazione conterrà un div contenente il form con tutti i campi per effettuare l’editing dei dettagli del volume. Lato frontend la visualizzazione del form potrebbe essere implementata mediante l’utilizzo del plugin di jquery jqModal, che visualizza una finestra in cui viene caricato la pagina di editing richiesto al server mediante una chiamata in get. Un’esempio funzionante di questo meccanismo è la funzionalità di upload della foto dell’utente presente nella pagina di /user/UpdateUser.do

Per implementare il meccanismo mediante jqModal occorre :

* predisporre sulla pagina da cui si scatenerà l’editing, un div che dovrà essere il contenitore della visualizzazione del documento vuoto
* un tag ancor con l’attributo class riportante la scritta editing su cui verrà scatenata l’aperta della popup window
* un div con identificativo univoco che sarà predisposto alla visualizzazione del form
* definire uno script jquery per far scatenare la creazione della finestra alla pressione del link

L’apertura è scatenata da un oggetto denominato trigger, che si aggancia sul class dell’href.

Supponiamo quindi di avere all’interno della view del documento vuoto, un link che punta all’azione di edit dei dettagli del volume ed un div nascosto id editWindow :

|  |
| --- |
| <a href=”#” class=”editVolumeDetails”> --  <div class=*"jqmWindow"* id=*"editWindow"*>  </div> |

sarà sufficiente inserito all’interno della pagina il seguente controllo javascript per ottenere l’obbiettivo prefissato :

|  |
| --- |
| <script type=*'text/javascript'*>  $().ready(**function**() {  $('#editWindow').jqm({ajax: '/DocSources/de/volbase/EditDetailsVolume.do', trigger: 'href.editVolumeDetails'});  });  </script> |

L’azione specificata nel controllo ajax ’/DocSources/de/volbase/EditDetailsVolume.do’ non è altro che un’invocazione al server applicativo che dovrà restituire un form (in formato partial dom), che verrà inserito all’interno dell’elemento <div id=”editWindow”> dalla libreria jqModal autonomamente.

Questo meccanismo rappresenta quindi la gestione dell’apertura del nostro form di editing della sezione, adesso procediamo ad analizzare come gestire una delle specifiche dell’applicazione fondamentale :

l’aggiornamento della pagina del documento senza effettuare il refresh di tutta la pagina visualizzata nel browser.

Per ottemperare a questa specifica applicativa, il nostro form di editing dovrà essere in grado di gestire le seguenti specifiche :

* inoltrare il del form contenente i dati inseriti dal client al server
* intercettare la risposta che riceverà dal server
* Posizionare la risposta all’interno della pagina di visualizzazione documento (caso in cui l’elaborazione sia andata a buon fine)
* Sostituire il codice contenuto all’interno del window con quello che riceve nella risposta di elaborazione dal server (validazione KO, e ripresentazione del form dei dati con i messaggi di errore)
* Chiudere nel caso in cui l’elaborazione si andata a buon fine la jqModal window

Queste specifiche verrano implementate mediante un’apposito controllo jquery che sarà posizionato in fondo al form di editing (verrà generato nella risposta inviata dal server per la richiesta '/DocSources/de/volbase/EditDetailsVolume.do'. Il controllo javascript dovrà essenzialmente, inviare i dati mediante un comando post, ed effettuare un’analisi della presenza nella risposta di un css che identifica un messaggio di validazione fallita (mediante il comando class select di jquery è possibile individuare la presenza di determinati class : se si riceve un KO dal processo di validazione applicativa, il server restituisce il form di inserito dati inviato, con aggiunti degli elementi span con class errorClass, quindi su questi class specifici è possibile costruire un meccanismo di selezione del div in cui visualizzare il risultato) se è presente dovrà caricare la risposta ricevuta nel form della popup window, se invece è andato a buon fine dovrà sostituirlo nel contenitore di visualizzazione del documento, occupandosi dopo di chiudere la finestra di popup.

Per quanto concerne il discorso della sicurezza applicativa, questa è gestita in automatica dal frame work di spring.

I vari link di editing verrano inseriti nelle Java Server Pages dell’applicazione mediante degli appositi tag library di spring security che permettono l’inserimento di codice in una pagina solo se l’utente che richiede la pagina ha determinati permessi di sicurezza. Nel caso in cui un’utente un po’ malizioso, avesse avuto modo di intercettare l’url per invocare la creazione di un volume, se non ha i permessi abilitati, riceverà in risposta una pagina che lo infomerà dell’insufficienza di permessi per effettuare l’azione richiesta.

Nel caso in cui l’editing del dettaglio del nuovo volume sia andata a buon fine quindi, il server restituirà direttamente una risposta html che conterrà un partial DOM rappresentante l’intero volume suddiviso per le sue quattro sezioni principali (questo perché la distinzione fra documento da creare e già creato è effettuata lato server e non si desidera inserire logica di business nel presentation layer). Ogni sezione avrà il pulsante di editing che punterà all’identificativo univoco del volume che a questo punto è stato creato sul database.

Per quanto riguarda invece l’editing delle altre 3 sezioni, il comportamento sarà leggermente diverso sulla gestione del codice di View che viene restituito se l’elaborazione è andata a buon fine : questo codice di view sarà sempre un blocco parziale contenente la sola sezione editata. Se effettuiamo l’editing della sezione Context, l’azione eseguita dal form di editing restituirà sempre un partialDOM contenente il codice di visualizzazione della sezione Context e non conterrà anche le altre sezioni del documento.

# Bibliografia di riferimento

Mike Keith, Merrick Schincariol - Pro JPA 2 Mastering the Java Persistence API

*Gamma, E., Helm, R., Johnson, R. e Vlissides, J. - Design Patterns: Elements of Reusable Object-Oriented Softwar*

Brett Porter, Carlos Sanchez, John Casey, Vincent Massol - Better Builds With Maven

Deepak Alur, Dan Malks, John Crupi - Core J2EE Patterns

Core JSTL - Mastering the JSP Standard Tag Library

Data Structures And Algorithms In Java 4Th

Ehcache UserGuide version 1.7.1

John Wiley & Sons -- Data Structures And Algorithms In Java 4Th Edition

log4j The Complete Manual

Lafore 2nd E - Data Structures and Algorithms in Java

Manning - AspectJ in Action

Manning - Dependency Injection (2009)

Manning - Eclipse In Action - A guide for Java Developers

Manning - Hibernate In Action (2006)

Manning - Hibernate Search In Action (2006)

Manning - Java Persistence with Hibernate (Nov 2006)

Manning - jQuery in Action (2008)

MWSS - Data Structures And Algorithms In Java

O'Reilly - Hibernate A Developers Notebook

Prentice - Core JSTL Mastering the JSPT Standard Tag Library nov-2002

Thinking in Java 4th Edition