Osservatorio Astronomico di Cagliari

2015

INAF-OAC Admin Report

OACAMM_0002

OAC-RSM Manuale Tecnico

Andrea Saba, Maria Renata Schirru

Reviewer: Marco Buttu

Released: 18/02/2015





Indice generale

OAC-RSM – Manuale Tecnico

Introduzione

OAC-RSM è la web application dell'Osservatorio Astronomico di Cagliari per la creazione delle Richieste di Spesa Motivate e le relative determine a contrarre.
Il presente manuale tecnico è indirizzato agli sviluppatori che debbano apportare modifiche alla web application.
Verrà quindi illustrata la struttura di OAC-RSM e in particolare la parte scritta tramite il framework Django, la struttura del Database e la configurazione del web server che ospiterà l'applicazione.
Le competenze richieste prevedono la conoscenza dei database relazione, il linguaggio di programmazione Python e l'amministrazione di un web server Apache.

OAC-RMS

OAC-RSM è basata sul framework Django che a sua volta è completamente scritto in Python.
Il framework si può interfacciare nativamente a diversi RDBMS come per esempio PostgreSQL, MySQL, SQLite e Oracle, tramite gli object-relational mapper (ORM). Questi permettono di descrivere la struttura del database tramite classi scritte in Python con un'astrazione che permette di interagire con il database senza la necessità di scrivere istruzioni in SQL.
Il framework si può interfacciare a diversi Web Server; tramite l'interfaccia WSGI con Apache o nginx, oppure tramite il modulo flup con Cherokee.
Un altra caratteristica del framework è quello di separare la definizione del contenuto delle pagine web, tramite quelle che vengono chiamate views, dal loro aspetto grafico che viene definito in quelli che vengono chiamati templates. Questo permette di dividere il compito tra il lavoro del grafico e quello dello sviluppatore permettendo a quest'ultimo di dedicarsi maggiormente al comportamento della web application.
Il framework mette a disposizione da subito un'interfaccia amministrativa che permette di gestire il contenuto del database, gli utenti e i loro permessi di accesso.

Descrizione della web application

La web application OAC-RSM è fondamentalmente un generatore di documenti per poter procedere all'acquisizione di beni e servizi e l'esecuzione di lavori in economia.
Le sue funzioni principali sono:

- la compilazione e creazione della Richiesta di Spesa Motivata da parte del RUP,
- la compilazione e creazione della Determina a contrarre da parte del Responsabile Amministrativo che verrà sottoposta al Direttore.

Inoltre il sistema permette al RUP di tenersi aggiornato sull'iter della Richiesta.
Gli accessi da parte dei due attori sono possibili solo tramite autenticazione.
Da parte del RUP è possibile una procedura passo passo per la compilazione della Richiesta con percorsi dinamici a seconda delle informazioni inserite.
Il Responsabile Amministrativo ha invece accesso all'interfaccia amministrativa tramite la quale può procedere a completare la Richiesta, inviare e-mail di notifica legate alle varie fasi dell'iter e generare in automatica la Determina da sottoporre al Direttore.
Il Direttore ha un accesso limitato al sistema che gli permette, sempre tramite autenticazione, di autorizzare l'inizio dell'iter della Richiesta a seguito di una e-mail inviata in automatico dal sistema e contenente le informazioni essenziali della Richiesta stessa.
Il RUP ha accesso, come anticipato, anche allo storico di tutte le Richieste fatte per visualizzare lo stato della singola richiesta ed eventualmente procedere ad una nuova stampa della stessa. Le richieste appaiono con uno sfondo di colore differente a seconda che siano state completate, siano in fase si lavorazione o siano state annullate.

Struttura dell'applicazione

L'applicazione si compone delle seguenti parti:

1. L'applicazione vera e propria basata sul Framework Django e composta quindi da moduli

scritti in Python

2. Un database relazionale
3. Un web server

L'applicazione è basata su Django 1.4.5 e ha la seguente struttura:

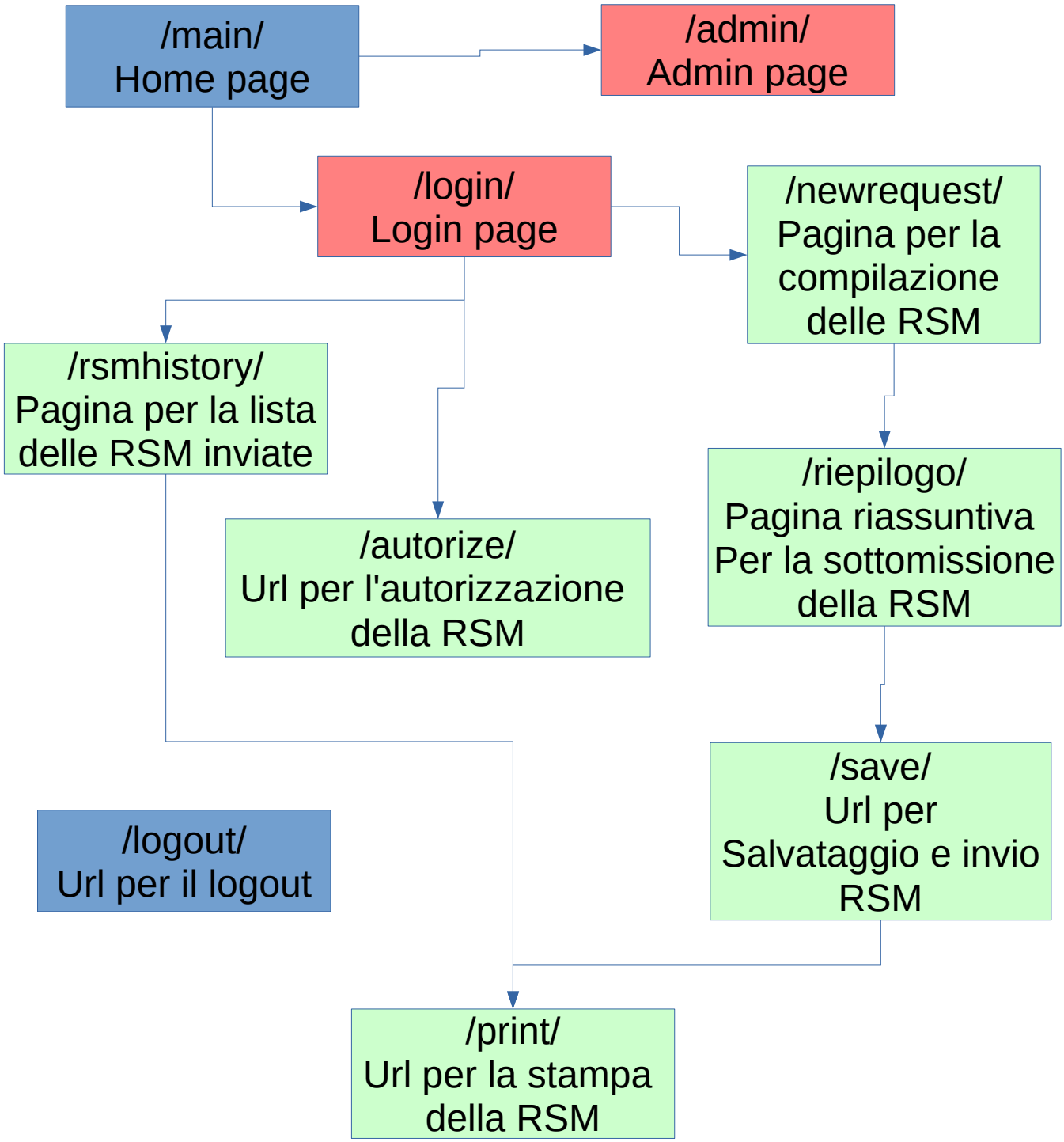
```
├── manage.py
├── moduli
│   ├── admin.py
│   ├── mysettings.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
├── ordini
│   ├── admin.py
│   ├── media
│   ├── modelforms.py
│   ├── models.py
│   ├── pdfmanage
│   │   ├── pdfmanage.py
│   ├── static
│   │   ├── admin -> /usr/lib/python2.6/.../admin/static/admin
│   │   ├── stylefixed2.css
│   │   └── suit
│   │   └── ...
│   ├── templates
│   │   ├── base.html
│   │   ├── dettagliarticoli.html
│   │   ├── doneorder.html
│   │   ├── genericmessage.html
│   │   ├── images
│   │   │   ├── Intestatazione.jpg
│   │   │   └── Intestatazione.png
│   │   ├── index.html
│   │   ├── login.html
│   │   ├── neworder.html
│   │   └── rsmhistory.html
│   ├── tests.py
│   ├── views.py
```

Dove la cartella `moduli` contiene i file necessari al funzionamento del web service mentre la cartella `ordini` contiene i file specifici per l'applicazione web.
Il file `settings.py` contiene le informazioni generali di configurazione del web service quali:

- Le informazioni di connessione al database
- Le applicazioni che sono disponibili per questo web service

- I percorsi in cui si trovano i file necessari ad alcune application come per esempio il path dei templates
- I moduli che django caricherà al suo avvio

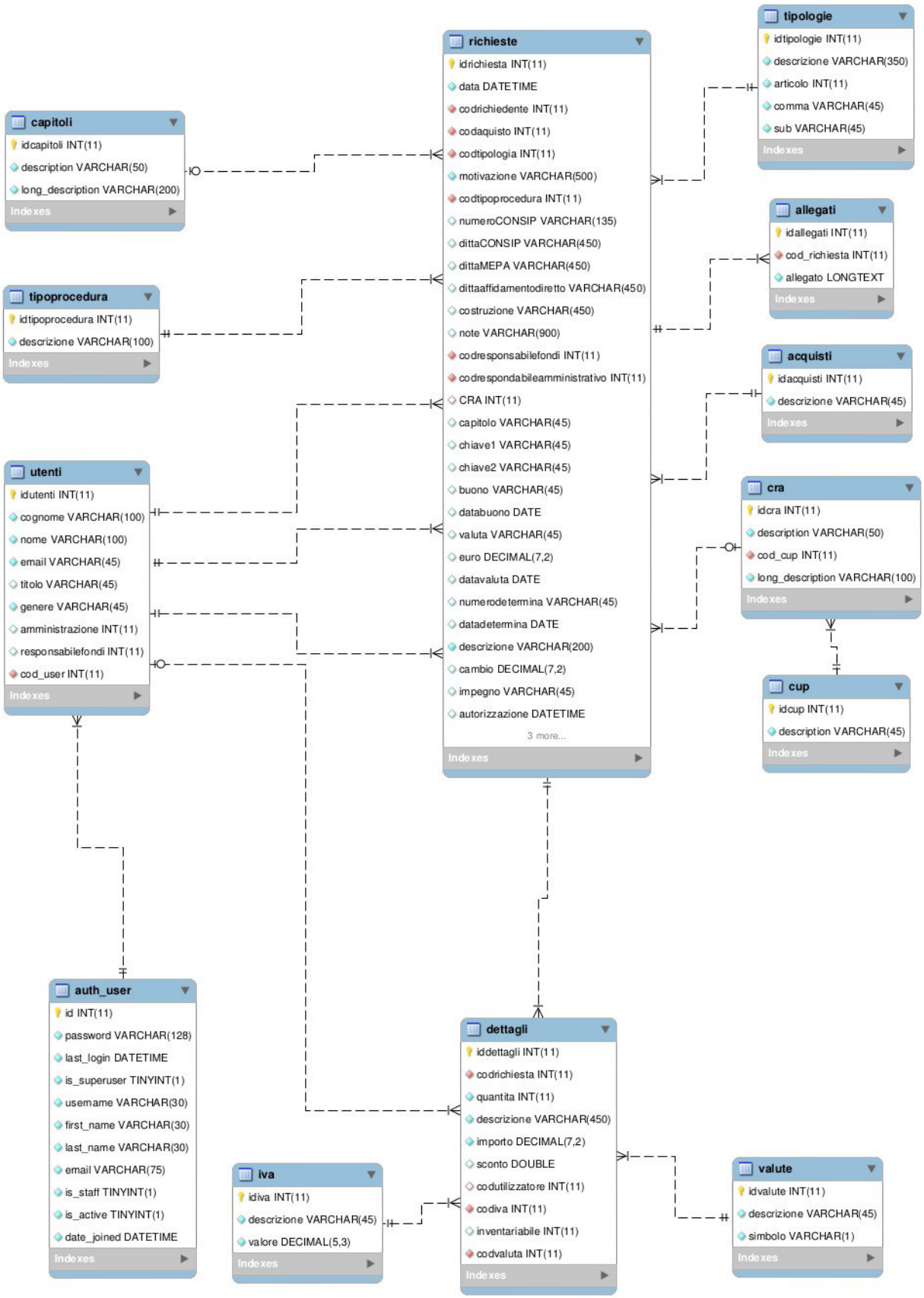
Il file `url.py` contiene i percorsi che verranno serviti dalle web application al posto di essere serviti dal server web che ospita il framework.
Di seguito lo schema dei path serviti dalla web application.



Nello schema, sono indicati in verde gli URLs relativi che sono accessibili solo dopo autenticazione.
Il modulo `admin.py` è inutilizzato.
Il modulo `mysetting.py` è stato creato per scorporare dal file `settings.py` le informazioni specifiche di questo web service. In questo file sono definiti gli indirizzi e-mail dell'Ufficio acquisti e del Direttore.
I file principali dell'applicazione, e cioè quelli che si trovano nella cartella ordini sono:

- `models.py`
  - contiene la definizione degli ORM (object relation mapping) che sono degli oggetti che definiscono un'interfaccia alle tabelle del database. Questo rende, da una parte, trasparente la manipolazione e ricerca dei dati, dall'altra permette l'estensione delle funzionalità associate alle singole tabelle.
- `admin.py`
  - contiene le informazioni relative a quello che deve essere presentato dall'interfaccia amministrativa come per esempio le tabelle del database, le specifiche azioni consentite sui record e i filtri applicabili.
- `modelforms.py`
  - contiene le definizioni delle strutture dei form che ospiteranno i dati restituiti dai From html che faranno parte delle pagine presentate all'utente durante l'interazione con le pagine web dell'applicazione
- `views.py`
  - contiene le funzioni che gestiscono le richieste che provengono dal server web e restituiscono i dati (o costruiscono le pagine web) che saranno la risposta all'interazione con le pagine web dell'applicazione.
- La cartella `templates`
  - contiene i modelli delle pagine web dell'applicazione che verranno completati con le informazioni restituire dalle funzioni definite nel file `view.py`
- La cartella/package `pdfmanage` e il relativo modulo `pdfmanage.py`
  - contiene le funzioni che definiscono il modello della carta intestata e quelle che definiscono la generazione della RSM e della Determina in formato PDF
- La cartella `static`
  - è una cartella direttamente risolta dal browser web che contiene i file di stile (CSS) e i javascript utilizzati dalla pagine web generate dai templates

L'RDBM è Oracle MySQL 5.5.41-0+wheezy1-log (Debian), e lo schema del database relazionale di OAC-RSM è il seguente


capitoli
idcapitoli INT(11)
description VARCHAR(50)
long_description VARCHAR(200)
Indexes
tipoprocedura
idtipoprocedura INT(11)
descrizione VARCHAR(100)
Indexes
utenti
idutenti INT(11)
cognome VARCHAR(100)
nome VARCHAR(100)
email VARCHAR(45)
titolo VARCHAR(45)
genere VARCHAR(45)
amministrazione INT(11)
responsabilefondi INT(11)
cod_user INT(11)
Indexes
auth_user
id INT(11)
password VARCHAR(128)
last_login DATETIME
is_superuser TINYINT(1)
username VARCHAR(30)
first_name VARCHAR(30)
last_name VARCHAR(30)
email VARCHAR(75)
is_staff TINYINT(1)
is_active TINYINT(1)
date_joined DATETIME
Indexes
richieste
idrichiesta INT(11)
data DATETIME
codrichiedente INT(11)
codaquisto INT(11)
codtipologia INT(11)
motivazione VARCHAR(500)
codtipoprocedura INT(11)
numeroCONSIP VARCHAR(135)
dittaCONSIP VARCHAR(450)
dittaMEPA VARCHAR(450)
dittaaffidamentodiretto VARCHAR(450)
costruzione VARCHAR(450)
note VARCHAR(900)
codresponsabilefondi INT(11)
codrespondabileamministrativo INT(11)
CRA INT(11)
capitolo VARCHAR(45)
chiave1 VARCHAR(45)
chiave2 VARCHAR(45)
buono VARCHAR(45)
databuono DATE
valuta VARCHAR(45)
euro DECIMAL(7,2)
datavaluta DATE
numerodetermina VARCHAR(45)
datadetermina DATE
descrizione VARCHAR(200)
cambio DECIMAL(7,2)
impegno VARCHAR(45)
autorizzazione DATETIME
3 more...
Indexes
tipologie
idtipologie INT(11)
descrizione VARCHAR(350)
articolo INT(11)
comma VARCHAR(45)
sub VARCHAR(45)
Indexes
allegati
idallegati INT(11)
cod_richiesta INT(11)
allegato LONGTEXT
Indexes
acquisti
idacquisti INT(11)
descrizione VARCHAR(45)
Indexes
cra
idcra INT(11)
description VARCHAR(50)
cod_cup INT(11)
long_description VARCHAR(100)
Indexes
cup
idcup INT(11)
description VARCHAR(45)
Indexes
dettagli
iddettagli INT(11)
codrichiesta INT(11)
quantita INT(11)
descrizione VARCHAR(450)
importo DECIMAL(7,2)
sconto DOUBLE
codutilizzatore INT(11)
codiva INT(11)
inventariabile INT(11)
codvaluta INT(11)
Indexes
iva
idiva INT(11)
descrizione VARCHAR(45)
valore DECIMAL(5,3)
Indexes
valute
idvalute INT(11)
descrizione VARCHAR(45)
simbolo VARCHAR(1)
Indexes

Il web server è Apache 2.2.22 con il modulo wsgi 3.3-4+deb7u1. Il VirtualHost definitio per gestire OAC-RSM è il seguente:

```
Listen 8090
<VirtualHost *:8090>
	ServerName 192.167.8.115
	ServerAdmin webmaster@jansky.oa-cagliari.inaf.it
	#LimitRequestBody 1004857600

	AddOutputFilterByType DEFLATE application/xml
	AddOutputFilterByType DEFLATE text/xml
	AddOutputFilterByType DEFLATE application/x-votable+xml
	AddOutputFilterByType DEFLATE text/javascript
	AddOutputFilterByType DELFATE text/css

	Alias /robots.txt /var/www/wsgi/static/robots.txt
	Alias /favicon.ico /var/www/wsgi/static/favicon.ico

	AliasMatch   ^/([^/]*\.css)   "/homes/jansky/asaba/Documents/Aptana   Studio   3
Workspace/moduli/ordini/static/$1"

	WSGIScriptAlias     /     "/homes/jansky/asaba/Documents/Aptana     Studio     3
Workspace/moduli/moduli/wsgi.py"
	WSGIDaemonProcess      moduli_oacagliari      display-name=%{GROUP}      python-
path="/homes/jansky/asaba/Documents/Aptana                  Studio                  3
Workspace/moduli:/usr/lib/python2.6/dist-packages"
	WSGIProcessGroup moduli_oacagliari

	<Directory         "/homes/jansky/asaba/Documents/Aptana         Studio         3
Workspace/moduli/moduli/">
		<Files wsgi.py>
			Order deny,allow
			Allow from all
		</Files>
	</Directory>

	Alias      /static/      "/homes/jansky/asaba/Documents/Aptana      Studio      3
Workspace/moduli/ordini/static/"
	<Directory         "/homes/jansky/asaba/Documents/Aptana         Studio         3
Workspace/moduli/ordini/static/">
		Order deny,allow
		Allow from all
	</Directory>

	Alias /media/ "/var/www/wsgi/media/"

	<Directory "/var/www/wsgi/media/">
		Order deny,allow
		Allow from all
	</Directory>
```

```
        ErrorLog ${APACHE_LOG_DIR}/moduli-error.log

        # Possible values include: debug, info, notice, warn, error, crit,
        # alert, emerg.
        LogLevel debug

        CustomLog ${APACHE_LOG_DIR}/moduli.access.log combined

</VirtualHost>
```

dove l'alias "`media`" definisce la posizione in cui verranno salvati gli allegati alle RSM.

Principali file dell'applicazione

Di seguito i file fondamentali per l'applicazione:

models.py

Nel file models.py sono presenti le definizioni degli ORM che sono in relazione con le tabelle del DB; oltre le definizione dei singoli campi delle tabelle, sono stati definiti dei nuovi campi per alcune tabelle che sono calcolati sulla base dei dati del record o dei records delle tabelle correlate. Inoltre sono state definite delle funzioni d'appoggio per delle operazioni specifiche quali:

- `estendinome()`
  - che restituisce l'articolo determinativo e il titolo nella forma grammaticale corretta
- `datatimenow()`
  - che restituisce la data attuale come stringa o come oggetto in base all'argomento passato

```
class Utenti(models.Model):
    idutenti = models.AutoField(primary_key=True, unique=True)
    cognome = models.CharField(max_length=100, verbose_name = "Cognome")
    nome = models.CharField(max_length=100, verbose_name = "Nome")
    email = models.EmailField(verbose_name = "E-mail")
    titolo = models.CharField(max_length=45, verbose_name = "Titolo")
    genere = models.CharField(max_length=1, verbose_name = "Genere")
    amministrazione = models.BooleanField(verbose_name = "Amministrazione")
    responsabilefondi = models.BooleanField(verbose_name = "Responsabile Fondi")
    user = models.ForeignKey(User, db_column='cod_user', verbose_name = "User")
    def __unicode__(self):
        return self.nomecompleto()
    def nomecompleto(self):
        return unicode(self.nome) + u" " + unicode(self.cognome)
    def esteso(self, capitalize = False, articolo = True, legato = False):
        #[articolo] + titolo + Nome + cognome
        articolo_tmp, titolo_tmp = estendinome(self.titolo, self.genere,
capitalize, legato)
        if articolo:
            return articolo_tmp + titolo_tmp + u" " + self.nomecompleto()
        else:
            return titolo_tmp + u" " + self.nomecompleto()
    class Meta:
```

```
        verbose_name_plural = "Utenti"
        verbose_name = "Utente"
        db_table = u'utenti'



class Richieste(models.Model):
    idrichiesta = models.AutoField(primary_key=True, unique=True)
    data = models.DateTimeField(editable = False, verbose_name = "Data della
domanda")
    codrichiedente = models.ForeignKey(Utenti, db_column='codrichiedente',
verbose_name = "Richiedente", related_name = "rel_name_richiedente")
    codaquisto = models.ForeignKey(Acquisti, db_column='codaquisto',
verbose_name = "Tipologia di Acquisto")
    codtipologia = models.ForeignKey(Tipologie, db_column='codtipologia',
verbose_name = "Tipologia di Articoli")
    motivazione = models.CharField(max_length=450, verbose_name = "Motivazione")
    codtipoprocedura = models.ForeignKey(Tipoprocedura,
db_column='codtipoprocedura', verbose_name = "Tipologia di Procedura")
    numeroconsip = models.CharField(max_length=135, db_column='numeroCONSIP',
blank=True, verbose_name = "Numero Convenzione CONSIP") # Field name made
lowercase.
    dittaconsip = models.CharField(max_length=450, db_column='dittaCONSIP',
blank=True, verbose_name = "Ditta CONSIP") # Field name made lowercase.
    dittamepa = models.CharField(max_length=450, db_column='dittaMEPA',
blank=True, verbose_name = "Ditta MEPA") # Field name made lowercase.
    dittaaffidamentodiretto = models.CharField(max_length=450, blank=True,
verbose_name = "Ditta Affidamento Diretto")
    costruzione = models.CharField(max_length=450, blank=True, verbose_name =
"Costruzione in economia")
    note = models.CharField(max_length=900, blank=True, verbose_name = "Note")
    codresponsabilefondi = models.ForeignKey(Utenti,
db_column='codresponsabilefondi', verbose_name = "Responsabile dei Fondi",
related_name = "rel_name_resp_fondi")
    codrespondabileamministrativo = models.ForeignKey(Utenti,
db_column='codrespondabileamministrativo', verbose_name = "Responsabile
Amministrativo", related_name = "rel_name_resp_amm")
    cra = models.ForeignKey(Cra, db_column='cra', null=True, blank=True,
verbose_name = "Cra") # Field name made lowercase.
    impegno = models.CharField(max_length=135, blank=True, verbose_name =
"Impegno") # Field name made lowercase.
    #capitolo = models.CharField(max_length=135, blank=True, verbose_name =
"Capitolo")
    codcapitolo = models.ForeignKey(Capitoli, db_column='codcapitolo',
null=True, blank=True, verbose_name = "Capitolo")
    chiave1 = models.CharField(max_length=135, blank=True, verbose_name =
"Chiave 1")
    chiave2 = models.CharField(max_length=135, blank=True, verbose_name =
"Chiave 2")
    buono = models.CharField(max_length=135, blank=True, verbose_name = "Buono
d'Ordine")
    databuono = models.DateField(null=True, blank=True, verbose_name = "Data del
Buono d'Ordine")
    #cup = models.CharField(max_length=135, db_column='CUP', blank=True) # Field
name made lowercase.
    valuta = models.CharField(max_length=135, blank=True, verbose_name =
```

```
"Valuta")
    euro = models.DecimalField(null=True, max_digits=8, decimal_places=2,
blank=True, verbose_name = "Importo in Euro")
    cambio = models.DecimalField(null=True, max_digits=8, decimal_places=5,
blank=True, verbose_name = "Cambio")
    datavaluta = models.DateField(null=True, blank=True, verbose_name = "Data
della Valuta")
    numerodetermina = models.CharField(max_length=450, blank=True, verbose_name
= "Determina")
    datadetermina = models.DateField(null=True, blank=True, verbose_name = "Data
della Determina")
    descrizione = models.CharField(max_length=450, blank=True, verbose_name =
"Descrizione breve per Determina")
    autorizzazione = models.DateTimeField(null=True, blank=True, verbose_name =
"Data autorizzazione Direttore")
    annullamento = models.DateTimeField(null=True, blank=True, verbose_name =
"Data annullamento")
    noteannullamento = models.CharField(max_length=450, blank=True, verbose_name
= "Note annullamento")

    def __unicode__(self):
        try:
            return u"{} di {}".format(self.data.strftime("%Y-%m-%d %H:%M:%S"),
self.codrichiedente)
        except:
            return u"XXXX"
    def calcolatotale(self, iva = True):
        totale = 0.0
        for d in Dettagli.objects.filter(codrichiesta = self.pk):
            totale+=d.totale(iva)
        return totale
    def totalerichiestavalutaiva(self):
        valuta = ""
        for d in Dettagli.objects.filter(codrichiesta = self.pk):
            valuta=d.codvaluta.simbolo
            break
        return u"{0} {1:.2f}".format(valuta, self.calcolatotale())
    totalerichiesta = property(totalerichiestavalutaiva)
    def autorizzata(self):
        if self.autorizzazione:
            return u"SI"
        else:
            return u"NO"
    autorizzata = property(autorizzata)
    def annullata(self):
        if self.annullamento:
            return u"SI"
        else:
            return u"NO"
    annullata = property(annullata)
    def crashort(self):
        if self.cra:
            return self.cra.description
        else:
            return u"NON DEFINITO"
    crashort = property(crashort)
    def ditta(self):
```

```
        if self.codtipoprocedura.pk == 1:
            #CONSIP
            return self.dittaconsip
        elif  self.codtipoprocedura.pk == 2:
            #MEPA
            return self.dittamepa
        elif self.codtipoprocedura.pk == 3:
            #RDO
            return u"--DA GARA--"
        elif self.codtipoprocedura.pk == 4:
            #Affidamento diretto
            return self.dittaaffidamentodiretto
    ditta = property(ditta)
    def appenddettaglitotals(self, tot_no_iva = False, tot_iva = True):
        for d in self.dettagli_set.all():
            d.appendtotals(tot_no_iva = False, tot_iva = True)
    def calcolatotale_unicode(self, iva = True):
        result = u"{:.2f}".format(self.calcolatotale(iva)).replace(".", ",")
        return result
    def getvalutasimbolo(self):
        d = Dettagli.objects.filter(codrichiesta = self.pk)
        if len(d) > 0:
            return d[0].codvaluta.simbolo
    def returnnewrichiestafromRecord(self, history=False):
        newrichiestavalues = []
        requeststatuscode = 0
        if history:
            …
            …

        return newrichiestavalues, unicode(requeststatuscode)
    class Meta:
        db_table = u'richieste'
        verbose_name_plural = "Richieste di spesa motivata"
        verbose_name = "Richiesta di spesa motivata"



class Dettagli(models.Model):
    iddettagli = models.AutoField(primary_key=True, unique=True)
    codrichiesta = models.ForeignKey(Richieste, db_column='codrichiesta')
    quantita = models.IntegerField(verbose_name = "Quantita'")
    descrizione = models.CharField(max_length=450, verbose_name = "Descrizione")
    inventariabile = models.BooleanField(verbose_name = "Inventariabile")
    importo = models.DecimalField(max_digits=8, decimal_places=2, verbose_name =
"Importo")
    sconto = models.FloatField(null=True, blank=True, verbose_name = "Sconto %")
    codutilizzatore = models.ForeignKey(Utenti, blank = True, null = True,
db_column='codutilizzatore', verbose_name = "Utilizzatore")
    codiva = models.ForeignKey(Iva, db_column='codiva', verbose_name = "IVA %")
    codvaluta = models.ForeignKey(Valute, db_column='codvaluta', verbose_name =
"Valuta")

    def totale(self, iva = False):
        if iva:
            return self.quantita*float(self.importo)*(1.-
float(self.sconto)/100.)*((1.+(float(self.codiva.valore)/100.)))
```

```
        else:
            return self.quantita*float(self.importo)*(1.-
float(self.sconto)/100.)
    def appendtotals(self, tot_no_iva = False, tot_iva = True):
        self.total = round(self.totale(iva=False), 2)
        self.totaliva = round(self.totale(iva=True), 2)

    def returnnewdettaglifromrecord(self):
        newdettaglivalues = []
        …
        …

        return newdettaglivalues

    def returnnewdettaglidictionaryfromrecord(self):
        dettaglidictionary = {}
        …
        …
        return dettaglidictionary

    def returndettaglirow(self):
        dettaglirow = []

        …
        …
        return dettaglirow

    class Meta:
        verbose_name_plural = "Dettagli"
        verbose_name = "Dettaglio"
        db_table = u'dettagli'
```

Nella classe **Utente** è stato definito il metodo **esteso()** che restituisce il il Nome e Cognome dell'utente preceduti dal titolo e dall'articolo indeterminativo corretto.

Nella classe **Richieste** sono state definite le funzioni:

- **calcolatotale**(*self*, iva = True)
- **totalerichiestavalutaiva**(*self*)
  - restituisce la valuta utilizzata nella RSM (presa dalla valuta indicata nei dettagli degli articoli)
- **autorizzata**(*self*), **annullata**(*self*), **crashort**(*self*), **ditta**(*self*)
  - restituiscono lo stato o il valore delle relative voci.
- **appenddettaglitotals**(*self*, tot_no_iva = False, tot_iva = True):
  - aggiunge il totale per articoli ai singoli ORM `articoli`
- **getvalutasimbolo**(*self*):
  - restituisce il simbolo della valuta utilizzata
- **returnnewrichiestafromRecord**(*self*, history=False):
  - restituisce i valori della richiesta in una forma utilizzabile più facilmente nei templates

Nella classe Dettagli sono state definite infine le funzioni:

- **totale**(*self*, iva = False) e **appendtotals**(*self*, tot_no_iva = False, tot_iva = True)
  - gestiscono il totale per articolo

- **returnnewdettaglifromrecord**(*self*), **returnnewdettaglidictionaryfromrecord**(*self*), **returndettaglirow**(*self*)
  - restituiscono i dati dei dettagli in formati specifici utilizzati in diversi punti del codice

admin.py

Nel file, come visto in precedenza, sono presenti le definizioni delle classi che costituiscono l'interfaccia per gli utenti che hanno accesso alla sezione amministrativa. I particolare la classe relativa alla tabella delle RSM è stata estesa per permettere l'esecuzione delle seguenti azioni:

- **annullaRSM()**
  - Annulla la richiesta di spesa selezionata
- **printrequest()**
  - Stampa la richiesta di spesa motivata per la firma del richiedente tramite la funzione `pdfmanage.pdfmanage.create_richiesta()`
  - il nome del file avrà il formato `modulo_richiesta_spesa_motivata_{0}.pdf`
- **printdetermina()**
  - Stampa la Determina tramite la funzione `pdfmanage.pdfmanage.create_determina()`
  - Il nome del file avrà il formato `determina_richiesta_spesa_motivata_000.pdf`
- **sendCIGreminder()**
  - Invia un promemoria per la richiesta del CIG (Codice Identificativo Gara)
- **sendSignreminder()**
  - Invia un promemoria alla firma al Richiedente
- **sendAssegnazioneCRArequest()**
  - Invia un promemoria di completamento della determina al Responsabile Amministrativo
- **downloadRSM()**
  - Scarica il CSV[1], in formato compatibile Microsoft Excel, delle Richieste Selezionate.
  - I campi presenti nel file saranno: Data Richiesta, Motivazione, RUP, Annullata, Determina, Autorizzata, CRA, Capitolo, Ditta, Totale

E per la stessa classe sono stati definiti i seguenti filtri:

- **AnnullatoListFilter()**
  - Filtra in base allo stato di annullamento della RSM
- **DeterminaListFilter()**
  - Filtra in base all'assegnazione della determina
- **DateListFilter()**
  - Filtra in base all'anno della RSM
- **RichiedenteListFilter()**
  - Filtra in base al Richiedente

Infine è stato inibito il comando di cancellazione delle RSM da parte degli utenti amministrativi. Solo gli utenti che hanno privilegi di amministratore di sistema hanno questa facoltà

1 CVS - Il comma-separated values (abbreviato in CSV) è un formato di file basato su file di testo utilizzato per l'importazione ed esportazione (ad esempio da fogli elettronici o database) di una tabella di dati. (http://it.wikipedia.org/wiki/Comma-separated_values)

modelforms.py

Come illustrato in precedenza questo modulo contiene la definizione dei form che vengono presentati all'utente durante la compilazione della RSM.

In particolare, per ogni form, e per ogni campo che lo compone, sono definiti:

- il tipo di campo (testo, booleano, dropdown, …)
- i limiti di input (es. il numero di caratteri massimi delle caselle di testo)
- l'etichetta del campo
- un campo booleano se il campo è obbligatorio o opzionale

Per alcuni form, come per esempio quello definito per l'inserimento dei dettagli degli articoli, è stato necessario definire dei controlli aggiuntivi per garantire la coerenza dei dati inseriti; questo viene fatto tramite l'overriding dei metodi "clean" del form. Nel caso in cui, al momento della sottomissione del form, queste funzioni di verifica non diano esito positivo, viene ripresentato all'utente lo stesso form con l'aggiunta dei messaggi di errore personalizzati.

Di seguito i controlli che vengono effettuati dai singoli metodi:

- `clean_quantita(self)`
  - verifica che la quantità del singolo articolo sia maggiore di zero
- `clean_importo(self)`
  - verifica che l'importo del singolo articolo sia maggiore o uguale a zero (lo zero è utilizzato per articoli in omaggio)
- `clean_sconto(self)`
  - verifica che lo sconto sia compreso tra 0 e 100

Inoltre viene definito il metodo che verifica la coerenza di tutto il form:

- `clean(self)`
  - verifica che se l'articolo è segnato come inventariabile debba essere indicato un utilizzatore e, di contro, se è indicato un utilizzatore l'articolo deve essere segnato come inventariabile.

Per i singoli form è stata inoltre definita una variabile `notecompilazione` che è una lista di stringhe le quali saranno presentate, tramite i templates, come note all'utente per la compilazione.

view.py

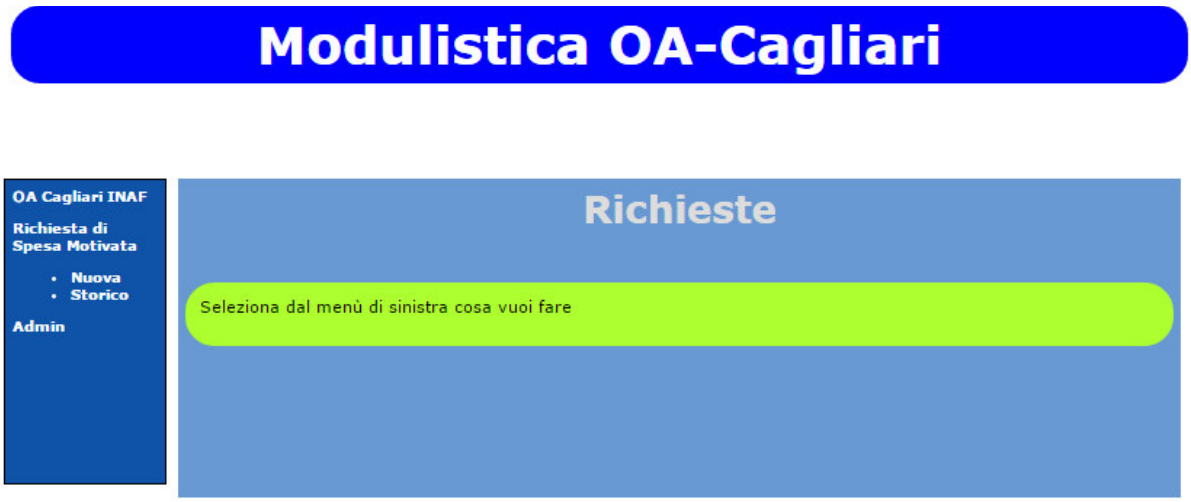
In questo modulo sono definite le funzioni che ricevono le richieste e costruiscono le risposte per il browser del client che si interfaccia con la web application.

Queste funzioni sono chiamate tramite i loro riferimenti nel modulo url.py e costruiscono le risposte.

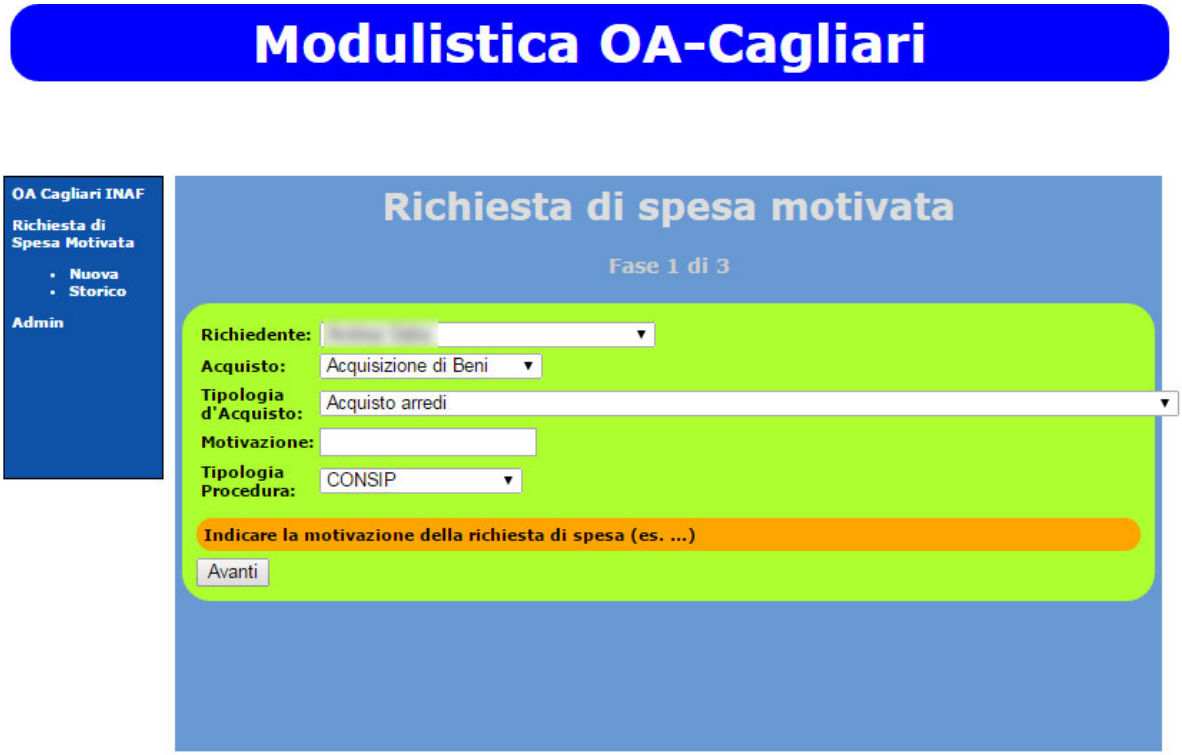
URL	view.py	template
/main/	`global_index()`	`index.html`
/newrequest/	`WizardOrder()`	`neworder.html`

/login/	Gestito da `django.contrib.auth.views.login`	
/rsmhistory/	`rsmhistory()`	`rsmhistory.html`
/print/	`printrequest()`	`genericmessage.html` in caso di errore
/autorize/	`autorizza()`	`genericmessage.html`
/logout/	Gestito da django.contrib.auth.views.logout	
/riepilogo/	`riepilogo()`	doneorder.html
/save/	`savemodule2()`	• `genercimessage.html` in caso di messaggi aggiuntivi da dare all'utente • `stamparichiesta()` per la stampa della RSM

- `global_index()`
  - presenta l'homepage della web application. Non richiede autenticazione.

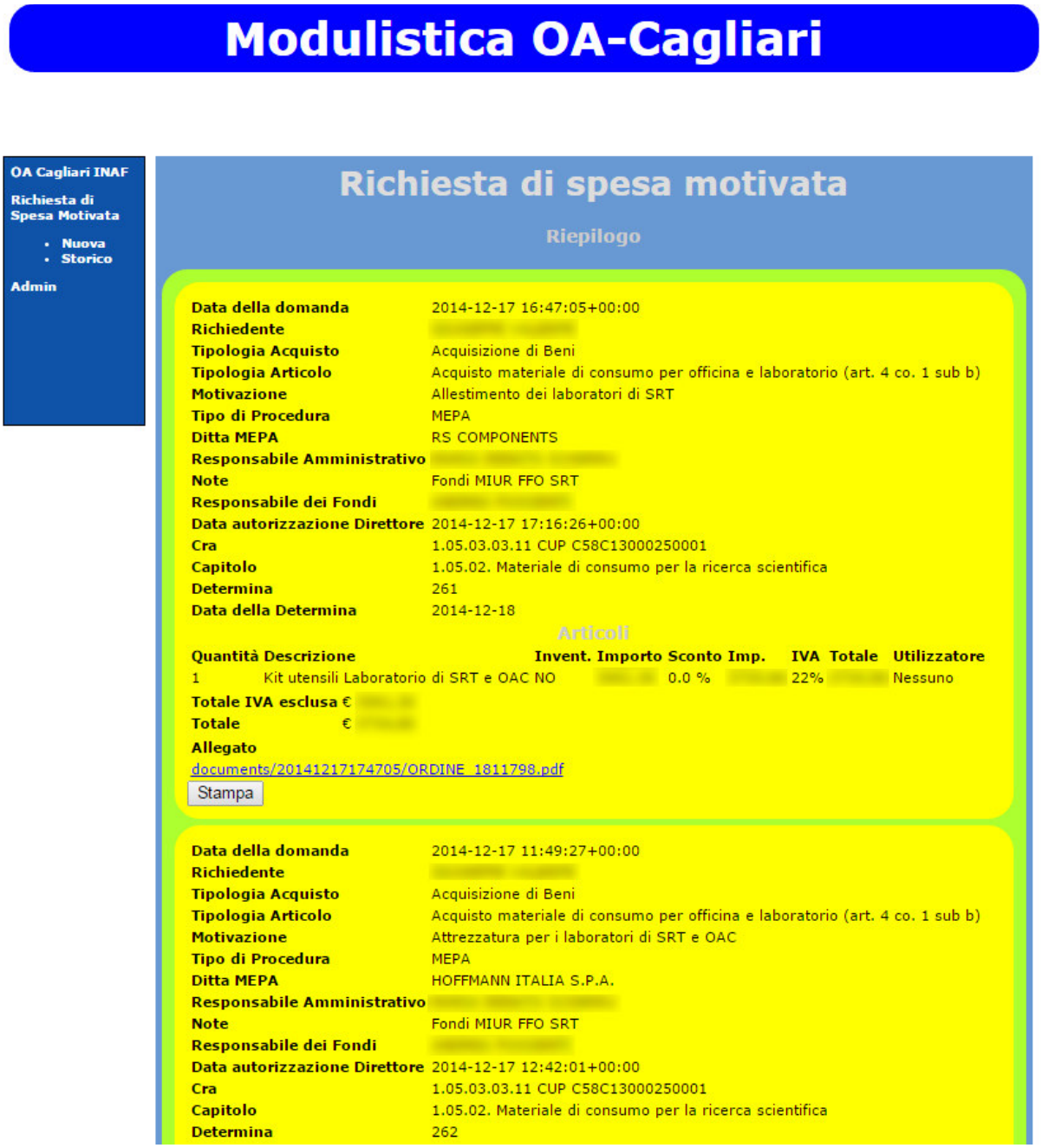


- `WizardOrder()`
  - A seguito dell'autenticazione restituisce la pagina corrispondente all'inizio della procedura di inserimento di una nuova RSM



- Il diagramma di flusso della procedura è rappresentato nell'Illustrazione 1
- Utilizza le credenziali dell'utente per preimpostare il nome del richiedente

- `rsmhistory()`
  - Dopo autenticazione restituisce per questo utente la pagina della lista delle RSM nelle quali egli è indicato come richiedente.
  - Per ogni RSM è utilizzato un colore di sfondo differente a seconda dello stato, che può essere:
    - Completata (status = 5; Giallo – CSS class "`moleculeD`")
    - In attesa dell'assegnazione di numero di Determina (status = 4; Arancione – CSS class "`molecule`")
    - In attesa dell'assegnazione del CRAM (status = 2 or 3; Arancione – CSS class "`molecule`")
    - In attesa dell'autorizzazione del Direttore (status = 1; Arancione – CSS class "`molecule`")
    - Annullata dall'amministrazione (status = 0; Grigio – CSS class "`moleculeA`")
  - Per ogni RSM sono presenti i link agli allegati inseriti durante la compilazione e un pulsante per la generazione del PDF della RSM corredata di tutti gli allegati. Alla generazione corrisponde il download sul browser del client della RSM stessa. La RSM conterrà tutti i dati che sono presenti nel database per quella RSM in quel momento, si ottiene così una RSM sempre aggiornata allo stato attuale della procedura.



- `printrequest()`
  - Come visto poco prima, questa funzione restituisce il download del file PDF generato in quel momento, di una RSM.
  - Nel file PDF generato sono sempre presenti anche tutti gli allegati della RSM.
  - Viene utilizzata al termine della procedura guidata di compilazione della RSM, alla richiesta di stampa tramite la pagina dello storico delle RSM e dall'interfaccia amministrativa.
- `autorizza()`

- l'URL corrispondente a questa chiamata viene inviato esclusivamente al Direttore per e-mail al termine della procedura di compilazione della RSM. L'e-mail contiene le informazioni essenziali della RSM.
- Viene presentata al Direttore una pagina per l'autenticazione. Autenticandosi il Direttore concede l'autenticazione e gli viene presentata una pagina con l'esito dell'operazione che può essere
  - "Hai autorizzato con successo la richiesta del 00/00/0000 00:00:00"
  - "Autorizzazione già concessa il 00/00/000 00:00:00"
  - "Richiesta annullata dall'amministrazione il 00/00/000 00:00:00"
  - Oppure un messaggio di errore.
- All'autorizzazione corrisponde il salvataggio nel relativo campo del record corrispondete della tabella "Richieste".

- `riepilogo()`
  - Restituisce la pagina finale della procedura guidata di compilazione nella quale si trovano le informazioni riassuntive di tutta la procedura guidata di compilazione della RSM
  - La funzione non salva alcun dato sul database.

- `savemodule2()`
  - È la funzione che effettua effettivamente tutti i salvataggi relativi alla RSM sul database tramite una transaction. Quindi se qualcosa nel salvataggio non va a buon fine nessun dato verrà scritto nel database.
  - La funzione si occupa anche di inviare le e-mail di notifica con le informazioni essenziali della RSM ai seguenti attori:
    - Richiedente
    - Direttore (in cui è presente anche il link per l'autorizzazione)
    - Responsabile dei fondi
    - Responsabile amministrativo
    - Ufficio acquisti (in cui è presente anche il link per un accesso diretto – dopo autenticazione – alla RSM per il completamento della procedura)

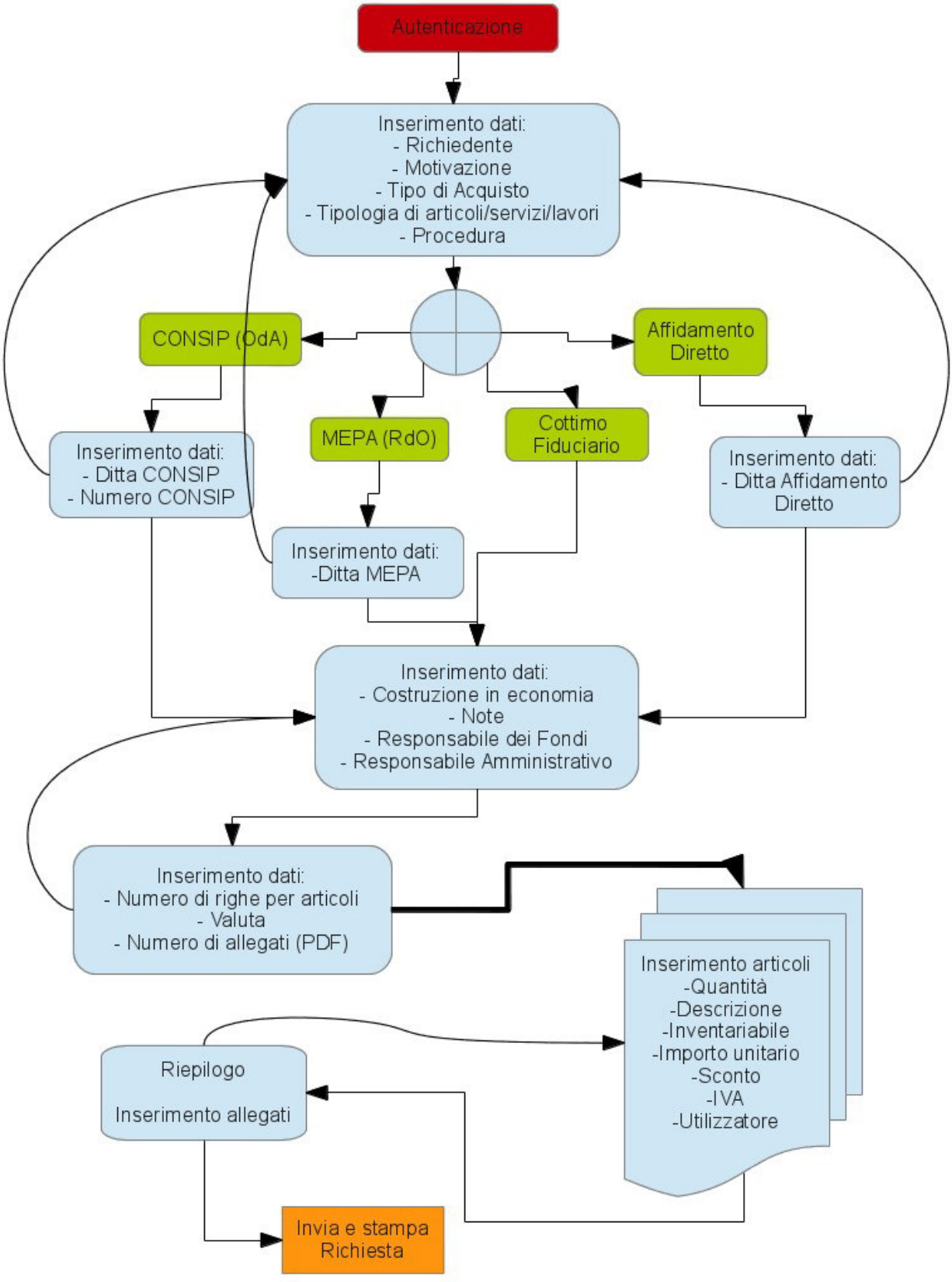


Illustrazione 1: Diagramma di flusso della procedura guidata di inserimento della RSM

I templates

Nella cartella sono presenti i template i quali definiscono uno schema delle pagine HTML (o XHTML o XML) prodotte dalla web application. In questo schema sono definite delle parti statiche e delle parti dinamiche. Le parti statiche seguono le specifiche HTML(o XHTML o XML) e sono utilizzate, per esempio, per definire l'header delle pagine. Le parti dinamiche sono definite tramite dei blocchi di codice che possono essere completati all'occorrenza con delle variabili passate alla pagina dalle funzioni definite nel file `view.py`, oppure tramite l'importazione di altri template. In questo modo è possibile costruire una gerarchia dei file che definisce prima la parte più generica e poi quella più specifica della singola pagina. Il file `base.html` definisce quindi la struttura generica della pagina e i blocchi che devono essere completati, tutte le altre pagine estendono il file `base.html` definendo quello che andrà a sostituire i diversi blocchi.
I file dei template sono:

- `base.html`
- `dettagliarticoli.html`
- `doneorder.html`
- `genericmessage.html`
- `login.html`
- `neworder.html`
- `rsmhistory.html`

Il package pdfmanage

Nel package `pdfmage` è definito l'omonimo modulo che, tramite la versione open source della libreria `reportlab`[2], genera i file pdf.

Le funzioni definite nel modulo sono:

- `returnstylesheet():`
  - funzione che restiuisce la definizione di tutti gli stili dei paragrafi che verranno utilizzati del pdf creati (dimensioni caratteri, allineamento, formato, spaziature righe, ...)
- `drawPage(canvas, doc):`
  - funzione che inserisce nella pagina passata come argomento l'intestazione e il piè di pagina con le immagini.
- `definedoc(buffer_pdf):`
  - funzione che restiusce un documento vuoto con le dimensioni di un foglio A4
- `create_richiesta(richiesta, dettagli):`
  - funzione che costruisce la RSM inserendo le informazioni prese dalla richiesta passata.
- `create_determina(buffer_pdf, richiesta):`
  - funzione che costruisce la Determina a contrarre per gli Affidamenti diretti e gli acquisti MEPA inserendo le informazioni prese dalla richiesta passata.

La costruzione dei documenti PDF segue quindi il principio di utilizzo della funzione `definedoc()` per definire le dimensioni della pagina, la funzione `drawPage()` per creare l'intestazione e il piè di pagina e infine una delle due funzioni `create_richiesta()` o `create_determina()` per definire il

2 http://www.reportlab.com/

contenuto.

Il testo viene inserito come una sequenza di paragrafi formattati uno di seguito all'altro. In alternativa un paragrafo o un oggetto grafico può essere inserito in una posizione specifica della pagina fornendo le coordinare del punto di inserimento sul foglio.

In queste funzioni sono presenti i testi che compongono i documenti.

Le immagini utilizzare nell'intestazione della pagina sono contenute nella cartella indicata nella variabile `moduli.mysettings.IMAGE_PATH`

La cartella static

Nella cartella `static` sono presenti i file per la formattazione delle pagine HTML e in particolare il file CSS specifico della web application e tutti i file dei javascript utilizzati per gli oggetti dinamici delle pagine.

Nel file di stile `stylefixed2.css` sono definiti i seguenti principali stili:

`#side-bar{}`	Menù sinistro
`* html #outer{}`	Contenitori nidificati
`* html #contain-all{}`	
`#inner{}`	
`#main-content{}`	
`.title_blue {}`	Titolo e sottotitolo del `main-content`
`.subtitle_gray {}`	
`#content {}`	Contenitore per le informazioni specifiche della pagina
`#top-bar{}`	Testata della pagina
`#topbar-inner {}`	
`#footer-inner {}`	Piè di pagina
`.mid-content{}`	
`.narrowinteger {}`	Formati numerici
`.narrowimporti {}`	
`.notedicompilazione {}`	Note sulla compilazione dei form
`.hidden {}`	
`.errors {}`	Errori dei dati inseriti nei form
`.molecule {}`	Informazioni riassuntive delle RSM con

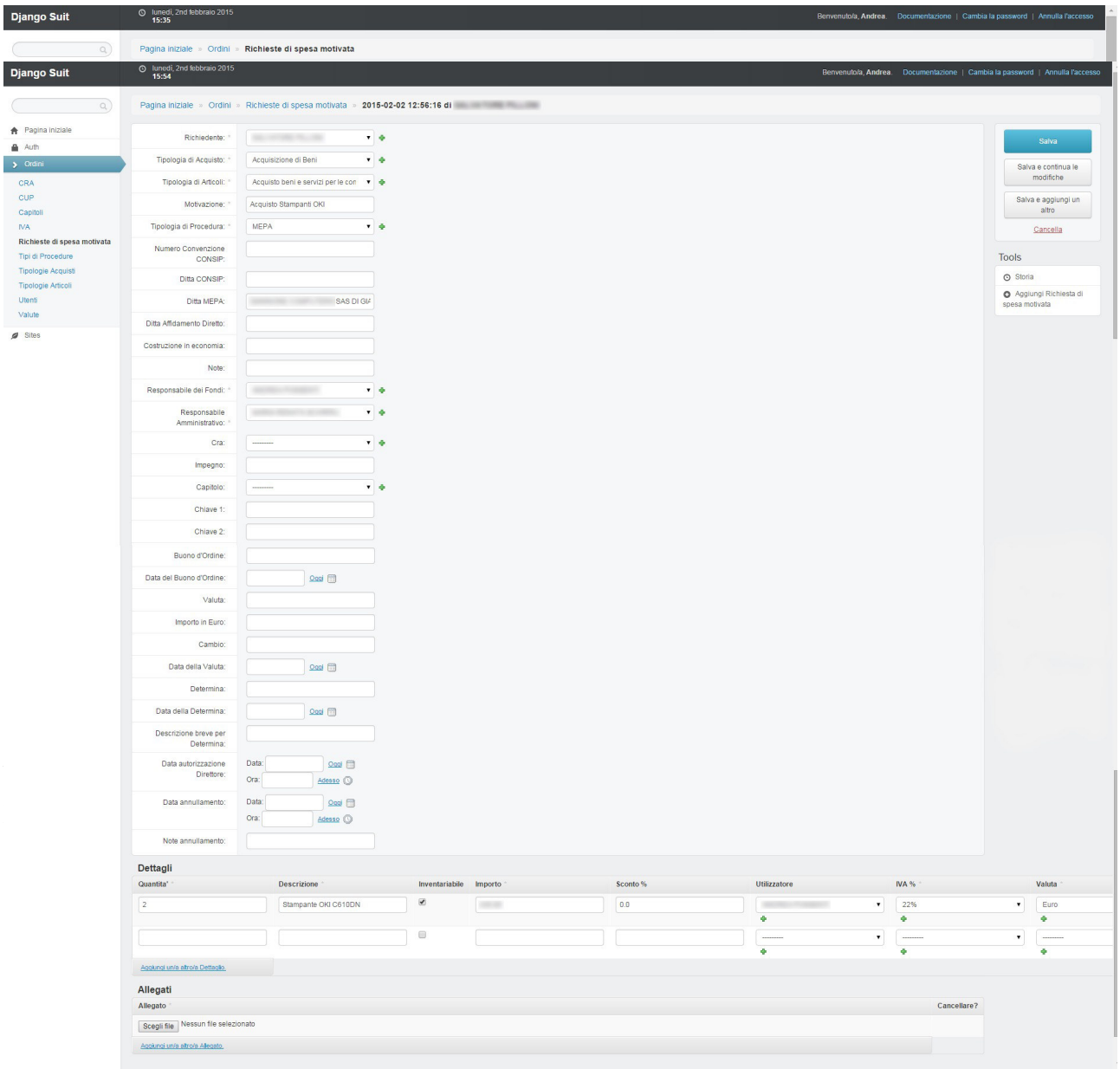
`.moleculeD {}`	colorazione di sfondo in base allo stato.
`.moleculeA {}`	

Interfaccia amministrativa

La procedura amministrativa viene effettuata a seguito di autenticazione, tramite l'interfaccia amministrativa messa a disposizione da Django e adattata per le specifiche esigenze.
Per l'aspetto grafico è stata scelto il tema Suit[3] che, tra l'altro, permette una migliore gestione del posizionamento dei filtri.
Per l'autenticazione sono stati definiti tre livelli di accesso all'interfaccia i possono effettuare le seguenti operazioni:

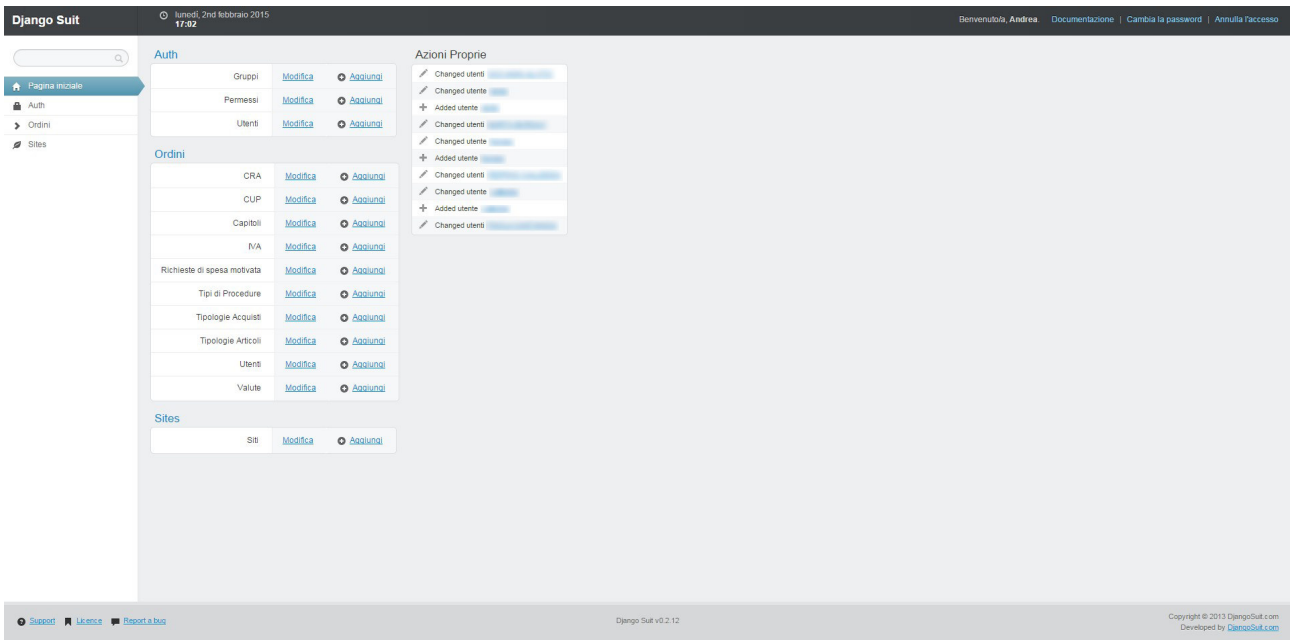
- Credenziali da utente registrato: gestione della password
- Credenziali settore amministrativo: completamento delle Richiesta di Spesa, Stampa delle Richieste, Generazione e Stampa delle Determine a contrarre
- Credenziali da amministratore di sistema: gestione degli utenti, dei permessi di accesso, delle tipologie di articoli, delle aliquote IVA, etc...

Quando all'interfaccia accede un utente registrato senza permessi d'accesso superiori è possibile tramite la voce "cambia password" modificare la propria password d'accesso.
Quando all'interfaccia accede una persona registrata come facente parte del settore amministrativo gli si presenta la tabella del database delle Richieste di Spesa Motivata. Accedendovi appare la seguente pagina con la lista di tutte le Richieste di Spesa Motivata che possono essere modificate facendoci click sopra.



consentite agli utenti dell'amministrazione, la modifica/creazione/cancellazione di tutti i dati di tutte le tabella. In particolare l'amministratore è l'unico utente al quale è consentito cancellare in maniera definitiva una RSM.

Gestione degli utenti



Gli utenti registrati sul sistema sono salvati nella tabella Users del sistema Django. Qui sono presenti oltre il nome utente e la password il livello di accesso gestito con il principio delle policy. È presente una seconda tabella Utenti che contiene le informazioni necessarie alla generazione dei documenti con ad esempio il genere, il titolo e il ruolo. La tabella Utenti è in relazione con la tabella Users.

Per creare un nuovo utente è necessario inserirlo nella tabella Users indicando oltre al nome utente la password, il nome, il cognome, l'indirizzo e-mail e anche lo stato (attivo/non attivo), l'accesso alla sezione di amministrazione, le credenziali di superutente. Per ultimo vanno indicati i gruppi ai quali appartiene l'utente (Amministrativo, Direttore, Richiedente).

Dopo aver inserito l'utente nella tabella Users è necessario creare un utente corrispondente nella tabella Utenti indicando Cognome, Nome, E-mail, Titolo, Genere, Amministrazione, Responsabile Fondi, User.

Ringraziamenti

Un ringraziamento a tutto il personale dell'Osservatorio Astronomico di Cagliari che ha dedicato tempo e pazienza nel dare indicazioni indicazioni per lo sviluppo e che di prestata a fare da beta tester segnalando i diversi bug individuati.