Insegnamento di Laboratorio di Basi di Dati

Lezione 9: Uso di Flask per sviluppo di applicazioni web che usano basi di dati

10/04/18

Roberto Posenato

ver. 1.7, 29/03/2018

Sommario

- Nella lezione 08 è stato introdotto Flask con un semplice esempio di applicazione web basata su Flask.
- In questa lezione si illustrano alcune funzioni avanzate di Flask e un'applicazione web che visualizza valori estratti da una base di dati PostgresSQL.

Accesso ai parametri della query string di una richiesta HTTP GET

- La query string di un URL è la sua sottostringa finale che inizia dopo '?'.
- Una query string è formata da nome=valore [& ...] codificata secondo lo standard HTTP.
  Invoco la risorsa
  Query string
  Esempio: `http://127.0.0.1/login?user=Tim&role=admin` ha come query string 'user=Tim&role=admin'.
- In Flask, l'oggetto `request` contiene tutte le informazioni di una richiesta HTTP.
  Tanti quanti sono nelle query string
- L'attributo `request.args`, di tipo `dict` {nomeParametro: valore,..}, rappresenta la query string di una richiesta. Sia le chiavi sia i valori sono stringhe!
- All'interno di un metodo associato ad un URL, l'oggetto `request` è automaticamente instanziato con la richiesta HTTP corrente.

```
from flask import Flask, request
app = Flask(__name__)
@app.route('/login')
def login():
    user = request.args['user']
    role = request.args['role']
    return ...
```

Devo sapere come è formata la query string

Accesso ai parametri di una richiesta di tipo HTTP POST

- Quando arriva una richiesta di tipo POST, eventuali parametri NON sono nella query string, ma nel corpo della richiesta nel formato 'nome=<valore>'.
- Tipico esempio è l'invio dei dati di una FORM HTML:

Esempio di FORM HTML5

```
<form action="http://localhost:5000/login" method="post">
   <label>Name: <input type="text" name="user"/></label><br>
   <label>Role: <input type="text" name="role"/></label><br>
   <input type="submit" value="Invia">
</form>...
```

Inviati nel corpo e non nella query string

- In Flask, i dati di un POST sono nel `dict` `request.form`:

```
from flask import Flask, request
app = Flask(__name__)
@app.route('/login')
def login():
    user = request.form['user']
    role = request.form['role']
    return ...
```

Arts -> elementi nella query string
Form se il messaggio contiene parametri nel corpo (POST)
Invio i dati in post a flask crea un form request e inserisce i dati di user e role
Poi ho quindi su user il nome e in role il valore role inviati prima

Accesso ai parametri di una richiesta di tipo HTTP POST o GET

- `route()` associa un metodo a un URL in modalità GET. Per usare lo stesso metodo anche in modalità POST, è necessario un'opzione nella associazione con il decorator, come mostrato in esempio.
- La variabile `request.method` indica la modalità della richiesta: i due valori principali sono 'GET' e 'POST'.

Esempio di metodo associato a richieste sia GET sia POST FORM HTML5

```
@app.route('/login', methods = ['POST', 'GET'])
def login():
    user = ''
    if request.method == 'POST':
        user = request.form['user']
    else:
        user = request.args['user']
    return ...
```

Senza method si ha di default get

Codice indipendente dal post o get

Generazione documenti HTML

- Il `return` di ogni metodo mappato da `route()` dovrebbe restituire un documento HTML5.
- Scrivere il documento come stringa multi riga (”’) non è una soluzione pratica.
- Il motore di modelli Jinja2 permette di costruire documenti HTML in modo più semplice e dinamico.
- In un metodo, il motore Jinja2 è attivato mediante il metodo `render_template(nomeTemplate, listaPar)` dove `nomeTemplate` è un template Jinja2 e `listaPar` è una sequenza di parametri con valore (nome=<valore>). Questi parametri possono essere usati dentro il template.

Esempio di metodo che usa Jinja2

```
@app.route('/login')
def login():
    user = request.arg['user']
    role = request.arg['role']
    return render_template('login.html', user = user, role = role)
```

Invoca jinjia e cerca nel file login.html se ci sono tag speciali con user e role sostituisce quindi il valore user con user e role con role

Compila in modo dinamico file html

Motore Jinja2

- Flask invoca Jinja2 via `render_template(nomeTemplate, listaPar)`.
- `nomeTemplate` deve essere un file (HTML con tag Jinja2) presente nella sottodirectory templates:

Esempio di organizzazione file

```
Directory applicazione web/
    Controller.py
    templates/
        login.html
```

Solo nella cartella c on questo nome flask va a trovare i file html

- Template come 'login.html' sono file HTML dove alcuni dati possono essere inseriti dinamicamente mediante tag specifici e dove è possibile anche eseguire del codice Python per generare il documento dinamicamente.
- Motori come Jinja2 analizzano un template sostituendo ciascun tag speciale con il testo determinato *interpretando* il tag e ritornando il documento HTML risultante.

Motore Jinja2

- Jinja2 interpreta i seguenti tag speciali:
  - {{ expr }} = stampa il valore dell'espressione `expr` (che può anche usare i parametri).
    Come fosse una espressione Python (inserisco il valore del taken)
    Il file html conterrà il valore dell'espressione.
  - {% ... %} = esegue il comando (pseudo) Python.
    Esegue codice python ma salva nel file finale ci sarà una riga vuota che comunque è stato eseguito
    Il file html conterrà solo una riga vuota (a meno che il comando non stampi qualcosa con chiamata a funzione specifica).
    Se si inizia con {%- o si termina con -%}, nel file html non sarà prodotto nessuno spazio/caratter di fine riga.
  - Se la funzionalità "Line Statements" è attivata, un comando (pseudo) Python può essere introdotto con il '#'.
    # ... fine linea chiude il contesto.
  - {# ... #} = marca il testo come commento (non viene riportato nel file html).

Motore Jinja2

- Il tag speciale {{ expr }} stampa il valore dell'espressione `expr` (che può anche usare i parametri).
- Se usato con Flask, Jinja decodifica il valore di `expr` in modo da evitare errori di tipo Cross-Site Scripting (XSS). Dettagli: http://flask.pocoo.org/docs/0.12/security/#xss

Esempio di template `login.html`

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>Stampa utente</title>
</head>
<body>
   <p>L'utente è {{ user }} e il suo ruolo è {{ role }}.</p>
</body>
```

Prende i parametri e fa l'escape per rappresentarli in modo corretto html

Motore Jinja2

- I comandi pseudo-Python usabili in Jinja2 sono simili a Python.
- Non potendo usare l'indentazione per chiudere i blocchi, è necessario scrivere la fine di un blocco esplicitamente:
  - {% if expr %} per iniziare un if. {% endif %} per indicare la fine del blocco.
  - {% for expr %} per iniziare un loop. {% endfor %} per fine del blocco.

Esempio di template con codice Python

Graffa % set Pippo = 1 % graffa setto una variabile (nuova)

```
<!doctype html>
<html>
<head> <meta charset="utf-8"> </head>
<body>    Blocco
{% if numero == 100%}{#numero è un parametro passato al template#}
   <p>Indovinato!</p>
{% else %}
   <p>Il valore {{ numero }} non è esatto.</p>
{% endif %} Fine blocco if
</body>
</html>
```

Chiusura for -> endfor

Specifiche

Si deve realizzare un'applicazione web che presenti il seguente comportamento:

1. All'accesso, (URL '/'), l'applicazione deve stampare una pagina di presentazione contenente, come elemento principale, una form dove è possibile selezionare un corso di studi fra quelli della facoltà di Scienze e un anno accademico (aa) in un elenco di aa leciti (base di dati 'did2014').
2. All'invio dei dati della form, l'applicazione deve ritornare una pagina in cui si presentano, in forma tabellare, tutti gli insegnamenti erogati (compreso i moduli) associati al corso di studi selezionato nell'aa indicato (se ci sono).
3. Per ciascun insegnamento ci deve essere nome, numero crediti, se è modulo o no e i docenti associati. Il nome dell'insegnamento erogato deve essere un link a una pagina di dettaglio dell'insegnamento.
4. La pagina di dettaglio dell'insegnamento deve presentare tutte le informazioni possibili associate all'insegnamento ricavabili dallo schema della lezione 3.

Analisi Specifiche

Analizzando in modo top-down le specifiche, emerge che:

- Occorrono tre pagine HTML: uno iniziale contenente la form, una per l'elenco degli insegnamenti erogati (o assenza) e una per il dettaglio di un insegnamento erogato.
- Il Controller deve, per ciascuna pagina (in ordine):
  1. Richiedere al Model lista corsi di studio e lista anni accademici della Facoltà di Scienze. Passarli poi alla view della prima pagina.
  2. Leggere i parametri 'id' corso studi e aa dal request e chiedere al Model lista degli insegnamenti erogati corrispondenti completi di docenti associati. Passare poi la lista alla view della seconda pagina.
  3. Leggere id insegnamento erogato dal request e chiedere al Model oggetto che rappresenta l'insegnamento, l'oggetto che rappresenta i docenti associati e quello che rappresenta il periodo di lezioni associate. Passare poi gli oggetti alla view della terza pagina.

Analisi Specifiche

- Il Model deve:
  1. Fornire un metodo per determinare i corsi di studi di una facoltà, dato il nome o id della stessa.
  2. Fornire un metodo per determinare gli aa validi di associabili ai corsi di studio di una facoltà.
  3. Fornire un metodo per determinare gli insegnamenti erogati dati id corso di studi e anno accademico.
  4. Fornire un metodo per determinare i dettagli, docenti e periodo lezioni dati id di un insegnamento erogato.
  5. Per ciascun metodo, Model registra su una tabella in un altro database il nome del metodo e l'istante in cui è stato invocato.

Anche se non necessario, in questa applicazione Model è diviso in due moduli: uno contiene i metodi sopra descritti, l'altro (DataMapper) contiene i metodi con le query alla base di dati PostgreSQL. Questo per sottolineare che l'interfaccia alla base di dati deve essere isolata il più possibile dalla logica/modello dell'applicazione.

Controller.py 1/3

```
'''Controller dell'applicazione web 'Insegnamenti'
Formattazione salva righe per i lucidi!
@author: posenato'''

import logging
from flask import Flask, request
from flask.templating import render_template
from Model import Model

logging.basicConfig(level=logging.DEBUG)
app = Flask(__name__)  # Applicazione Flask!
app.jinja_env.line_statement_prefix = '#'  # attivo Line \
    statements in JINJA

app.model = Model()
app.facolta = app.model.getFacolta("Scienze Matematiche Fisiche \
    e Naturali")
```

Controller.py 2/3

```
@app.route('/')
def homePage():
    '''Home page deve presentare form per la scelta corso studi e
    anno accademico tra i corsi della facoltà di Scienze MM FF \
        NN.'''
    corsiStudi = app.model.getCorsiStudi(app.facolta['id'])
    aA = app.model.getAnniAccademici(app.facolta['id'])
    return render_template('homepage.html', facolta=app.facolta, \
        corsiStudi=corsiStudi, aa=aA)

@app.route('/insegnamenti', methods=['POST', 'GET'])
def insegnamenti():
    '''Elenco degli insegnamenti di un corso di studi in un a.a.'''
    if request.method == 'POST':
        idCorsoStudi = request.form['idCorsoStudi']
        aA = request.form['aa']
    else:
        idCorsoStudi = request.args['idCorsoStudi']
        aA = request.args['aa']
```

Controller.py 3/3

```
    corsoStudi = app.model.getCorsoStudi(idCorsoStudi)
    insEroConDoc = app.model.getInsEroConDoc(idCorsoStudi, aA)
    return render_template('insegnamenti.html', \
        facolta=app.facolta, corsoStudi=corsoStudi, aa=aA, \
        insErogati=insEroConDoc)

@app.route("/insegnamento", methods=['POST', 'GET'])
def insegnamento():
    '''Dettagli di un insegnamento erogato'''
    pass  # Da completare da parte dello studente.


if __name__ == '__main__':  # Questo if deve essere ultima \
    istruzione.
    app.run(debug=True)  # Debug permette anche di ricaricare i \
    file modificati senza rinizializzare il web server.
```

Model.py 1/5

```
"""Semplice Model applicazione 'Insegnamenti'.
Questo Model mantiene anche un log delle chiamate ai metodi.
La separazione con il database è forzata.
@author: posenato"""

from datetime import date, datetime
from DM_PG import DM_PG

class Model(object):
    """Realizza il modello dei dati da pubblicare."""

    def __init__(self):
        self.id = "Model_" + date.today().isoformat()
        self.dataMapper = DM_PG()  # DataMapper verso PostgreSQL

    def getFacolta(self, name):
        """Ritorna struct della facoltà con nome 'name'"""
        fac = self.dataMapper.getFacolta(name)
        self.dataMapper.log(self.id, datetime.today(), 'getFacolta')
        return fac
```

Model.py 2/5

```
    def getCorsiStudi(self, idF):
        """Ritorna list di {id, nome,codice, durataAnni} dei corsi \
        di studio della facolta 'idFS'"""
        cs = self.dataMapper.getCorsoStudiFacolta(int(idF))
        self.dataMapper.log(self.id, datetime.today(), \
        'getCorsiStudi')
        return cs

    def getAnniAccademici(self, idF):
        """Ritorna list di stringhe con gli anni accademici \
        presenti nei corsi di studio della facoltà 'idFS'"""
        aa = self.dataMapper.getAnniAccademiciFacolta(int(idF))
        self.dataMapper.log(self.id, datetime.today(), \
        'getAnniAccademici')
        return aa
```

Model.py 3/5

```
    def getInsEroConDoc(self, corsoStudi, annoA):
        """Ritorna lista ins. erogati nel 'corsoStudi' nell'anno \
           accademico 'annoA'.

        Ogni elemento della lista è
        {id, nome,discr,hamoduli, modulo,nomeModulo, \
           discriminanteModulo, haunita, nomeUnita, crediti, docente}"""

        listaIns = \
            self.dataMapper.getInsEroConDoc(int(corsoStudi), annoA)

        if not listaIns:  # una lista vuota è sempre false!
            return listaIns
```

Model.py 4/5

```
        # listaIns può avere più righe per uno stesso insegnamento
        # se ci sono più docenti.
        # Tali righe si devono unire unendo i docenti.

        listaInsFinale = []
        rigaPrecedente = listaIns[0]

        for riga in listaIns:
            if riga['id'] == rigaPrecedente['id'] and riga != \
                rigaPrecedente:
                riga['docente'] = riga['docente'] + "\n" + \
                rigaPrecedente['docente']
            else:
                listaInsFinale.append(riga)

        self.dataMapper.log(self.id, datetime.today(), \
            'getInsEroConDoc')

        return listaInsFinale
```

Model.py 5/5

```
    def getCorsoStudi(self, idCS):
        """Ritorna {idCS, nome,codice, durataAnni, annoaccademico, \
        stato} del corso di studi 'idCS' dove stato è lo stato di \
        attivazione."""
        cs = self.dataMapper.getCorsoStudi(int(idCS))
        self.dataMapper.log(self.id, datetime.today(), \
        'getCorsoStudi')
        return cs

    def __del__(self):
        self.dataMapper.close()  # Chiudere sempre il DataMapper
```

DM_PG.py 1/10

```
"""DataMapper verso PostgreSQL. La connessione è un oggetto \
    condiviso da tutte le istanze della classe. La connessione \
    viene chiusa quando non c'è nessuna istanza della classe \
    attiva."""
from datetime import datetime
import logging
import psycopg2.extras

class DM_PG():
    """Data mapper verso PostgreSQL. Per semplicità, i parametri \
    di connessione sono attributi di classe: dovrebbero essere \
    scritti in un file esterno e caricati durante __init__."""
    __server = "localhost"
    __db = "did2014"
    __db4Log = 'posenato'
    __user = 'posenato'
    __pw = '6078'
    __dbCon = None  # La connessione è condivisa!
    __db4LogCon = None  # La connessione è condivisa!
    __nIstanze = 0
```

DM_PG.py 2/10

```
    @classmethod
    def __open(cls):
        if cls.__dbCon is None:
            try:
                cls.__dbCon = psycopg2.connect(host=cls.__server, \
    database=cls.__db, user=cls.__user, password=cls.__pw)
                cls.__dbCon.set_session(readonly=True, \
    autocommit=True)  # Connessione di lettura condivisa
                logging.info("Connection to database " + cls.__db + \
    " created.")
            except psycopg2.OperationalError as err:
                logging.error("Error connecting to PostgreSQL DBMS \
    at %s.\nDetails: %s.", cls.__server, err)
                cls.__dbCon = cls.__db4LogCon = None
                exit()
```

DM_PG.py 3/10

```
        else: #Questo else è del TRY
            try:
                cls.__db4LogCon = \
                psycopg2.connect(host=cls.__server, database=cls.__db4Log, \
                user=cls.__user, password=cls.__pw)
                cls.__db4LogCon.set_session(autocommit=True)  # \
                Connessione di scrittura condivisa
                logging.info("Connection to database " + \
                cls.__db4Log + " created.")
            except psycopg2.OperationalError as err:
                logging.error("Error connecting to PostgreSQL \
                DBMS at %s.\nDetails: %s.", cls.__server, err)
                cls.__dbCon = cls.__db4LogCon = None
                exit()
            return "New connection opened."
    return "Connection already opened."
```

DM_PG.py 4/10

```
    @classmethod
    def __close(cls):
        if cls.__nIstanze == 0 and cls.__dbCon is not None:
            cls.__dbCon.close()
            cls.__db4LogCon.close()
            logging.info("Connection closed.")
            cls.__dbCon = cls.__db4LogCon = None

    @classmethod
    def __cursor(cls):
        """Ritorna un cursore che restituisce dict invece di tuple \
        per ciascuna riga di una select."""
        return \
        cls.__dbCon.cursor(cursor_factory=psycopg2.extras.DictCursor)

    @classmethod
    def __cursor4log(cls):
        """Ritorna un cursore per scrivere nel database \
        cls.__db4Log."""
        return cls.__db4LogCon.cursor()
```

DM_PG.py 5/10

```
    def __init__(self):
        DM_PG.__open()
        DM_PG.__nIstanze += 1

    def close(self):
        """Chiude in modo esplicito la connessione, se non ci sono \
           altre istanze attive"""
        self.__del__()

    def __del__(self):
        DM_PG.__nIstanze -= 1
        DM_PG.__close()
```

DM_PG.py 6/10

```
    def getFacolta(self, name):
        """Ritorna il dict {id, nome, url, dataCreazione} della \
        facoltà di nome "name" se esite, None altrimenti."""
        with DM_PG.__cursor() as cur:
            cur.execute("SELECT id, nome, url, datacreazione FROM \
        Facolta WHERE nome ILIKE %s", (name,))
            facolta = cur.fetchone()
            return facolta

    def getCorsoStudiFacolta(self, idF):
        """Ritorna una list di dict {id, nome,codice, durataAnni} \
        di tutti i corsi di studi associati alla facoltà con id \
        'idF'"""
        with DM_PG.__cursor() as cur:
            cur.execute('SELECT cs.id, cs.nome, cs.codice, \
        cs.durataanni as "durataAnni" FROM corsostudi cs join \
        corsoinfacolta csf on cs.id =csf.id_corsostudi WHERE  \
        csf.id_facolta=%s order by cs.nome', (int(idF),))
            return list(cur)
```

DM_PG.py 7/10

```
    def getAnniAccademiciFacolta(self, idF):
        """Ritorna una list di stringhe rappresentanti gli anni \
        accademici di tutti i corsi di studi associati alla facoltà \
        con id 'idF'"""
        with type(self).__cursor() as cur:
            cur.execute('SELECT DISTINCT ie.annoaccademico FROM \
            inserogato ie JOIN corsostudi cs ON ie.id_corsostudi=cs.id \
            JOIN corsoinfacolta csf ON cs.id=csf.id_corsostudi WHERE \
            csf.id_facolta=%s ORDER BY ie.annoaccademico DESC', \
            (int(idF),))
            lista = list()
            for tupla in cur:
                lista.append(tupla[0])
            return lista
```

DM_PG.py 8/10

```
    def getCorsoStudi(self, idCS):
        """Ritorna dict {idCS, nome,codice, durataAnni, \
           annoaccademico, stato} del corso di studi 'idCS'"""
        with type(self).__cursor() as cur:
            cur.execute('SELECT cs.id, cs.nome, cs.codice, \
           cs.durataanni as "durataAnni", scs.annoaccademico as \
           "annoAccademicoUltimoStato", st.valore as "ultimoStato" \
           FROM corsostudi cs JOIN statodics scs ON \
           cs.id=scs.id0_corsostudi JOIN statocs st ON \
           scs.id1_statocs=st.id WHERE cs.id=%s order by \
           annoaccademico desc', (int(idCS),))
            return cur.fetchone()
```

DM_PG.py 9/10

```
    def getInsEroConDoc(self, idCS, annoA):
        """Ritorna una list di dict {id, nome,discr,hamoduli, \
           modulo,nomeModulo, discriminanteModulo, crediti, docente} \
           di tutti gli insegnamenti erogati del corso di studi 'idCS' \
           nell'anno accademico 'annoA'"""
        with type(self).__cursor() as cur:
            cur.execute('SELECT DISTINCT ie.id,i.nomeins as nome, \
           d.nome as discr, ie.hamoduli, abs(ie.modulo) as modulo, \
           ie.nomemodulo as "nomeModulo", ie.discriminantemodulo as \
           "discriminanteModulo", haunita, nomeunita as "nomeUnita", \
           ie.crediti, p.nome || ' ' || p.cognome as docente FROM \
           inserogato ie JOIN insegn i ON ie.id_insegn=i.id JOIN \
           corsostudi cs ON ie.id_corsostudi=cs.id LEFT JOIN \
           discriminante d ON ie.id_discriminante=d.id LEFT JOIN \
           docenza doc ON doc.id_inserogato=ie.id JOIN persona p ON \
           doc.id_persona=p.id WHERE cs.id=%s AND \
           ie.annoaccademico=%s ORDER BY i.nomeins, modulo', \
           (int(idCS), annoA))
            return list(cur)
```

DM_PG.py 10/10

```
    def log(self, idModel: str, instant: datetime, methodName: \
        str):
        """Scrive il log sulla tabella InsegnamentiLog che deve \
        essere presente nel database cls.__database4Log.
        CREATE TABLE INSEGNAMENTILOG (id SERIAL PRIMARY KEY, \
        idModeApp VARCHAR NOT NULL, instant TIMESTAMP NOT NULL, \
        methodName VARCHAR NOT NULL)"""
        if idModel is None or methodName is None or instant is None:
            return
        with DM_PG.__cursor4log() as cur:
            cur.execute("INSERT INTO InsegnamentiLog (idModeApp, \
            instant, methodName) VALUES (%s,%s,%s)",
                        (idModel, instant, methodName))
            if cur.rowcount != 1:
                logging.error("Log has not been written. Details: " \
                + idModel + ", " + str(instant) + "," + methodName)
                return False
            return True
```

Nelle prossime slide si presentano i template scritti in linguaggio Jinja2 2.9

homepage.html 1/2

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>Pagina principale</title>
  <style type="text/css">
  form {
   border: 1px solid black;
   padding: 2px;
   width: auto;
  }
  </style>
</head>
<body>
<h1>Facoltà di {{ facolta.nome }}</h1>
<p>Selezionare il corso di studi e l'anno accademico di cui si
     vogliono vedere gli insegnamenti erogati.</p>
```

homepage.html 2/2

```
<form action="/insegnamenti" autocomplete="on" method="post">
   <label>Corso di studi:</label>
   <select name="idCorsoStudi">
{% for cs in corsiStudi %}
      <option value="{{ cs.id }}">{{ cs.nome|truncate(120, True)
     }}</option>
{% endfor %}
   </select><br>
   <label>Anno accademico:</label>
   <select name="annoA">
{% for aa in annoA %}
      <option value="{{ aa }}" {% if loop.first %}
     selected="selected" {% endif %}>{{ aa }}</option>
{% endfor %}
   </select><br>
   <input type="submit" value="Invia">
</form></body></html>
```

insegnamenti.html 1/4

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>Pagina insegnamenti</title>
  <style type="text/css">
  table, thead, tbody {
   border: 1px solid black;
   border-collapse: collapse;
   padding: 1px;
   width: auto;
  }
  tr.even {
   background-color: aliceblue;
  }
  tr.odd {
   background-color: inherit;
  }
```

insegnamenti.html 2/4

```
  td,th  {
   border: 1px solid gray;
   margin: 0px;
   padding: 2px;
  }
  td.numero {
   text-align: right;
  }
  th {
  background-color: highlight;
  }
  </style>
</head>
<body>
<h1>Facoltà di {{ facolta.nome }}</h1>
<h2>Corso di studi {{ corsoStudi.nome }}</h2>
<h3>Durata {{ corsoStudi.durataAnni }} anni</h3>
#if corsoStudi.ultimoStato <- è un line statement!
```

insegnamenti.html 3/4

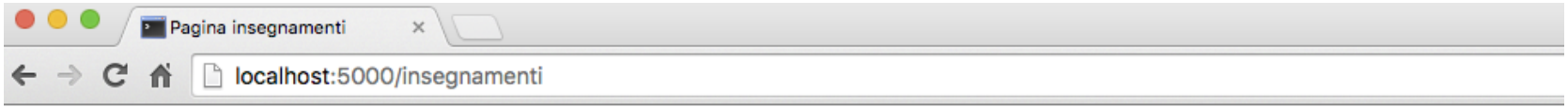
```
<p>L'ultima variazione di stato del corso di studi è stata
     nell'anno accademico {{ corsoStudi.annoAccademicoUltimoStato
     }}. Da tale anno il corso di studi è <strong>{{
     corsoStudi.ultimoStato }}</strong>.</p>
#endif
<p>Elenco degli insegnamenti erogati nell'anno accademico
     <strong>{{ annoA }}</strong>.</p>
#if not insErogati
<p><strong>Non ci sono insegnamenti nell'anno accademico
     <strong>{{ annoA }}</strong>.</strong></p>
#else
<table>
   <thead>
      <tr><th>N.</th> <th>Insegnamento</th> <th>Discr.</th>
     <th>Modulo</th> <th>Nome modulo</th> <th>Discr. modulo</th>
     <th>Nome unità</th> <th>Crediti</th> <th>Docente</th></tr>
   </thead>
   <tfoot><tr><td colspan="8">Fine tabella</td></tr></tfoot>
   <tbody>
{%- macro url(idC) -%}
/insegnamento?id={{idC}}
{%- endmacro %}
{#spazio memoria per creare la variabile numeroIns che vive al di
     fuori del ciclo for#}
#set ns = namespace(numeroIns=1)
```

insegnamenti.html 4/4

```
#for ins in insErogati
      <tr class="{{ loop.cycle('odd', 'even') }}">
         <td class="numero">{%- if ins.modulo == 0 -%}{{
    ns.numeroIns }}
                          {%- set ns.numeroIns = ns.numeroIns+1 -%}
                          {%- endif -%}</td>
         <td><a href="{{ url(ins.id) }}">{{ ins.nome }}</a></td>
         <td>{{ ins.discr }}</td>
         <td class="numero">{{ ins.modulo }}</td>
         <td>{{ ins.nomeModulo }}</td>
         <td>{{ ins.discriminanteModulo }}</td>
         <td>{{ ins.nomeUnita }}</td>
         <td class="numero">{{ ins.crediti }}</td>
         <td>{{ ins.docente|replace('\n','<br>') }}</td>
      </tr>
#endfor
   </tbody>
</table>
#endif
<p><a href="/">Torna alla pagina principale</a></p>
</body></html>
```

Pagina iniziale:

Pagina insegnamenti:



Facoltà di Scienze matematiche fisiche e naturali

Corso di studi Laurea in Informatica

Durata 3 anni

L'ultima variazione di stato del corso di studi è stata nell'anno accademico 2009/2010. Da tale anno il corso di studi è **attivo**.

Elenco degli insegnamenti erogati nell'anno accademico **2013/2014**.

N.	Insegnamento	Discr.	Modulo	Nome modulo	Discr. modulo	nome unità	Crediti	Docente
1	Algebra lineare	-	0				6.00	Enrico Gregorio
2	Algoritmi	-	0				12.00	Roberto Segala
3	Analisi matematica I	-	0				6.00	Federica Briata
4	Analisi matematica II	-	0				6.00	Federica Briata
5	Architettura degli elaboratori	-	0				12.00	Franco Fummi
	Architettura degli elaboratori	-	1		I turno A-l	Laboratorio	2.00	Nicola Bombieri
	Architettura degli elaboratori	-	1		II turno M-Z	Laboratorio	2.00	Nicola Bombieri

Esercitazione

Completare l'applicazione presentata realizzando i metodi e i template necessari per dare le informazioni di un insegnamento.