# Architettura

Descrizione di alto livello

L’architettura del sistema rispetta la specifica JavaEE ed è composta da tre layer:

* Il Presentation Layer genera delle pagine web dinamiche in base ai dati forniti in input dal client che vengono interpretati e passati al Business Layer. I contenuti dinamici sono creati grazie alla realizzazione, a questo livello, di un pattern MVC (Model View Controller) che utilizza Servlet e JSP.

Il Controller ricevendo le richieste HTML dal Client, le inoltra alle Servlet di competenza, che utilizzando il Model accede alla logica di Business dell’applicazione richiamando il Web Service appropriato. La risposta, gestita dal Controller, viene inoltrata alla View che permette la visualizzazione al Client.

Dopo aver valutato varie tecnologie si è scelto di utilizzare il framework opensource Struts per implementare il design pattern Model View Controller. Tale frame work garantisce modularità, riusabilità, manutenibilità e rapidità di sviluppo in quanto l’applicazione, costituita da livelli logici distinti, consente di scrivere codice modulare, riutilizzabile e sviluppabile in parallelo.

In questo layer viene anche gestito il controllo sui dati inseriti e sui permessi di un utente per compiere delle funzionalità.

Il Presentation Layer viene implementato su Web Server Apache Tomcat versione 6.

* Il Business Layer contiene la logica applicativa che interpreta ed elabora i dati forniti in input dal livello superiore. Il Layer è costituito da un’Application Server che mette a disposizione una serie di servizi implementati come dei WebServices utilizzando la tecnologia Apache Axis 2.

Tra i Web Services che verranno implementati nell’architettura, GestioneLogin e GestioneAccount si occupano dell’accesso dell’utente al sistema, della creazione/modifica di nuovi account e dei relativi controlli utilizzando le informazioni nel Data Layer.

Il Web Service GestioneNotizia implementa la logica per l’inserimento, la modifica oppure la cancellazione delle notizie.

Ogni Web Services indicato prevederà un oggetto Validator che si occupa dei controlli sui dati inseriti e sui permessi di un utente per compiere delle funzionalità.

In tale Layer viene inserito un oggetto Service Locator per il caricamento univoco dei file di configurazione relativi ai vari servizi da implementare e per la gestione della lingua dei messaggi di errore allo startup.

È previsto un Business Layer Interface per disaccoppiare il Front-End dell’applicazione con il Business Layer.

* Il Data Layer è il punto d’accesso alle risorse utilizzate. Al suo interno un ruolo importante viene ricoperto dal Data Access Object (DAO) il quale richiamando le Stored Procedures implementate in Oracle PLSQL gestisce la persistenza dei dati.

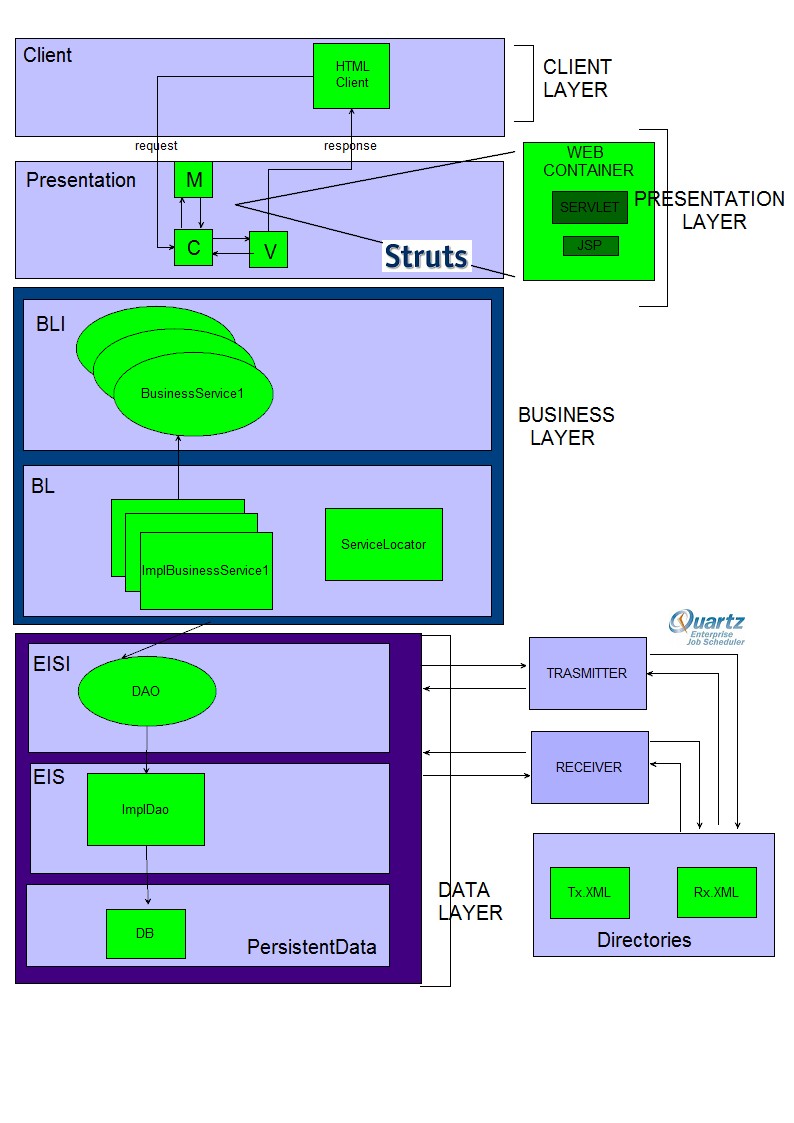
Il Database relazionale è realizzato in Oracle DataBase 11g e fornisce ai Layer superiori i dati necessari per elaborare le risposte da fornire al client.

Anche in tale Layer è prevista una interfaccia per disaccoppiare il livello di Business Logic dalle risorse del sistema.

I processi di Trasmissione e Ricezione per le notizie batch sono esterni al sistema e sono gestite tramite uno scheduler che utilizza il frame work Quartz.

Tale frame work consente di eseguire tali processi con un periodo prefissato, comunicando con il Data Layer dell’applicazione e in particolare con il DAO.

Il processo Trasmissione consente di trasferire le notizie dal DB verso un’apposita directory dopo un’opportuna validazione del formato. L’oggetto Ricezione si occupa invece di prelevare le notizie inviate da un Fornitore esterno al sistema e renderle persistenti dopo la validazione.



**3.1.1 Tecnologie Utilizzate**

Web Server: Apache Tomcat versione 6.0

Web Services con teconologia Apache Axis2 1.6.

Database: Oracle 11g.

Receiver e Transmitter: Framework Quartz.

Log: Log4j logger

Presentation: Apache Struts 2.3 per Web App

Gestione Lingue: Framework Internazionalizzazione e localizzazione (i18n).

Test: JUnit

**3.1.2 Struttura Fisica**

La suddivisione dell’architettura logica in Layer disaccoppiati grazie all’utilizzo di interfacce consente l’eventuale deployment di ogni singolo layer del sistema su web server distinti per garantire maggiore robustezza, modularità e scalabilità. In particolare il Data Layer potrebbe essere deployato su un server diverso rispetto al Presentation Layer e Business Layer affinchè le risorse siano dislocate in posti diversi rispetto a che le utilizza. I Web Services dell’applicazione implementata sono localizzati tutti su un unico server e inglobati in un unico Business Layer per motivi di semplicità.

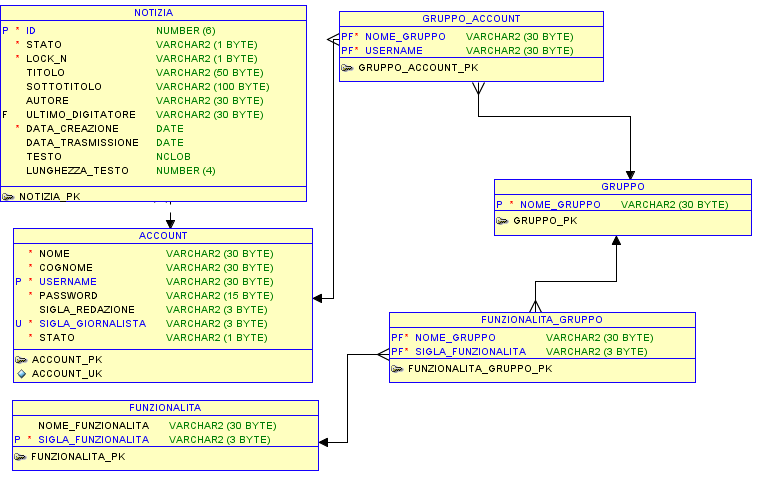
# Modello Dati

## C:\Users\Maurizio\Desktop\ProgFinaleUIIP\Diagrammi\Modello_relazionale.jpgModello Entità-Relazione

* Aggiunta la relazione Gruppo\_Account in quanto è necessario tener traccia di eventuali account appartenenti ad entrambi i gruppi (Giornalista e Amministratore).
* Aggiunta dell’attributo Stato all’entità Account per tener traccia di eventuali account cancellati logicamente.
* Aggiunta dell’attributo Id all’entità Notizia, il quale rappresenta la chiave primaria.

## Modello Logico

Come si può evincere dal modello sottostante le entità Testo\_Notizia ed Anagrafica\_Notizia, con i relativi attributi, sono state incorporate nell’ entità Notizia



## Script DataBase:

**Create User:**

create user editoriale identified by editoriale;

grant all privileges to editoriale;

disconnect;

connect editoriale;

**Create Table gruppo:**

CREATE TABLE gruppo (

nome\_gruppo varchar2(30),

CONSTRAINT gruppo\_pk PRIMARY KEY (nome\_gruppo)

);

**Create Table account:**

CREATE TABLE account (

nome varchar2(30) default '' NOT NULL,

cognome varchar2(30) default '' NOT NULL,

username varchar2(30) default '' NOT NULL,

password varchar2(15) default '' NOT NULL,

sigla\_redazione varchar2(3) default '',

sigla\_giornalista varchar2(3) default '' NOT NULL,

stato varchar2(1) default 'A' NOT NULL,

CONSTRAINT account\_pk PRIMARY KEY (username),

CONSTRAINT account\_uk UNIQUE (sigla\_giornalista)

) ;

**Create Table gruppo\_account:**

CREATE TABLE gruppo\_account (

nome\_gruppo varchar2(30),

username varchar2(30),

CONSTRAINT gruppo\_account\_pk PRIMARY KEY (nome\_gruppo,username),

CONSTRAINT gruppo\_account\_fk1 FOREIGN KEY (nome\_gruppo) REFERENCES gruppo(nome\_gruppo) ON DELETE CASCADE,

CONSTRAINT gruppo\_account\_fk2 FOREIGN KEY (username) REFERENCES account(username) ON DELETE CASCADE

) ;

**Create Table funzionalità:**

CREATE TABLE funzionalita (

nome\_funzionalita varchar2(30),

sigla\_funzionalita varchar2(3),

CONSTRAINT funzionalita\_pk PRIMARY KEY (sigla\_funzionalita)

) ;

**Create Table funzionalità\_gruppo:**

CREATE TABLE funzionalita\_gruppo (

nome\_gruppo varchar2(30),

sigla\_funzionalita varchar(3),

CONSTRAINT funzionalita\_gruppo\_pk PRIMARY KEY (nome\_gruppo, sigla\_funzionalita),

CONSTRAINT funzionalita\_gruppo\_fk1 FOREIGN KEY (nome\_gruppo) REFERENCES gruppo(nome\_gruppo) ON DELETE CASCADE,

CONSTRAINT funzionalita\_gruppo\_fk2 FOREIGN KEY (sigla\_funzionalita) REFERENCES funzionalita(sigla\_funzionalita) ON DELETE CASCADE

) ;

**Create Table notizia:**

CREATE TABLE notizia (

id number(6),

stato varchar2(1) default 'S' NOT NULL,

lock\_n varchar2(1) default 'N' NOT NULL,

titolo varchar2(50),

sottotitolo varchar2(100),

autore varchar2(30),

ultimo\_digitatore varchar2(30),

data\_creazione date default CURRENT\_TIMESTAMP NOT NULL,

data\_trasmissione date,

testo NCLOB,

lunghezza\_testo number(4),

CONSTRAINT notizia\_pk PRIMARY KEY (id),

CONSTRAINT notizia\_fk FOREIGN KEY (ultimo\_digitatore) REFERENCES account(username)

) ;

**Create sequence seq\_notizie:**

genera una sequenza che sarà utilizzata nel trigger successivo

create sequence seq\_notizie INCREMENT BY 1 START WITH 1 NOCACHE NOCYCLE ;

**Create trigger trigger1:**

utilizza seq\_notizie per assegnare un id univoco ed incrementale alla notizia

CREATE or REPLACE

TRIGGER trigger1

BEFORE INSERT ON notizia

FOR EACH ROW

BEGIN

SELECT seq\_notizie.nextval

INTO :new.id FROM dual;

END trigger1;

/

**Create trigger trigger2:**

Simula una foreign key tra le tabelle Account e Notizia in modo da tener tracia di eventuali aggiornamenti sull’attributo autore

CREATE or REPLACE

TRIGGER trigger2

AFTER UPDATE OF username ON account

FOR EACH ROW

BEGIN

UPDATE notizia SET autore= :new.username WHERE autore= :old.username;

END;

/

**Create trigger trigger3:**

Implementa un Update on cascade nella foreign keytra le tabelle Account e Notizia in modo da aggiornare automaticamente il contenuto del campo ultimo\_digitatore della tabella Notizia nel caso di modifiche nel corrispondente campo username della tabella Account

CREATE or REPLACE

TRIGGER trigger3

AFTER UPDATE OF username ON account

FOR EACH ROW

BEGIN

UPDATE notizia SET ultimo\_digitatore= :new.username WHERE ultimo\_digitatore= :old.username;

END;

/

**Create trigger trigger4:**

Implementa un Update on cascade nella foreign keytra le tabelle Account e Gruppo\_Account in modo da aggiornare automaticamente il contenuto del campo username della tabella Gruppo\_Account nel caso di modifiche nel corrispondente campo username della tabella Account

CREATE or REPLACE

TRIGGER trigger4

AFTER UPDATE OF username ON account

FOR EACH ROW

BEGIN

UPDATE gruppo\_account SET username= :new.username WHERE username= :old.username;

END;

/

**Create trigger trigger5:**

Simula il comportamento di una Foreign key nel caso di inserimento di tupla nella tabella Notizia, controllando la reale esistenza dell’autore inserito nella tabella Account, o nel caso in cui la notizia sia stata ricevuta da un sistema esterno (RCV) si controlla la presenza della sigla RCV all’interno del campo autore. Nel caso in cui nessuna delle due condizioni sia verificata viene generato un errore Oracle (ORA-20000 – ‘Autore Non Valido’)

CREATE or REPLACE

TRIGGER trigger5

BEFORE INSERT ON notizia

FOR EACH ROW

DECLARE counter number;

BEGIN

SELECT count(\*) INTO counter from account where username = :new.autore;

IF

:new.autore != 'RCV' and counter = 0

THEN

raise\_application\_error(-20000, 'Autore non valido.');

END IF;

END;

/

**Insert:**

Inserimento di tre account appartenenti al gruppo Amministratore

INSERT INTO account VALUES ('admin', 'admin', 'admin@aa.a', 'admin', 'aaa', 'ad1', 'A');

INSERT INTO account VALUES ('admin1', 'admin1', 'admin1@aa.a', 'admin', 'aaa', 'ad2', 'A');

INSERT INTO account VALUES ('admin2', 'admin2', 'admin2@aa.a', 'admin', 'aaa', 'ad3', 'A');