

Università degli studi di Salerno



CORSO DI LAUREA IN INGEGNERIA INFORMATICA

Anno accademico 2019/2020

REGISTRAZIONE DEI NOMI A DOMINIO NEL CCTLD .IT

Processo di comunicazione tra Registrar e Registro .it

Relatore:

Ch.mo Prof.

Ivan Visconti

Candidato:

Simone Lungarella

Matr: 0612702882

Indice

Indice	3
1. INTRODUZIONE.....	5
1.1 CONTESTO	5
1.2 TARGET	5
2. DNS NEL CCTLD .IT	6
2.1 DNS IN BREVE	6
2.2 DEBOLEZZE DNS.....	7
2.3 DNSSEC	8
2.4 PROCESSO DI ACCREDITAMENTO.....	8
2.5 PROTOCOLLO EPP	9
3. ECOSISTEMA APPLICAZIONE.....	10
3.1 TECNOLOGIE	10
3.2 LIBRERIE	10
3.3 DIPENDENZE	11
4. PROGETTAZIONE.....	13
4.1 DATA LAYER	13
4.2 BUSINESS LAYER	17
4.3 PRESENTATION LAYER	19
5. IMPLEMENTAZIONE.....	25
5.1 APPLICATION CONTEXT	25
5.2 FACES CONFIG	25
5.3 WEB	26
5.4 APPLICATION CONTEXT PROVIDER.....	27
5.5 DATA TRANSFER OBJECTS.....	28
5.6 SERVIZI	30
5.7 SERVIZI DI RECUPERO COMMAND E RESPONSE.....	40
5.8 CLASSI BUILDER PER REQUEST E RESPONSE.....	42
5.9 DATA ACCESS OBJECTS	48
6. TEST	53
6.1 TEST CONNETTIVITÀ	53
6.2 TEST RESPONSIVITÀ.....	54
6.3 TEST FUNZIONALI	55
7. CONCLUSIONI	58
7.1 PRESTAZIONI.....	58
7.2 OTTIMIZZAZIONI POSSIBILI.....	58

1. INTRODUZIONE

1.1 CONTESTO

Nella creazione dei nomi di dominio ci sono molti attori che interagiscono tra di loro per far sì che il nome di dominio rispetti tutte le regole previste dall'ente gestore relativo ad esso. Per i domini .it l'ente responsabile è il **Registro .it**. Per fornire il servizio di creazione domini, il Registro, non agisce direttamente ma delega questo compito a professionisti che prendono il nome di **Registrar**. Il Registro prevede l'utilizzo del protocollo **EPP** per comunicare con i Registrar, un protocollo basato su XML. Il servizio più importante che definisce e fornisce questo protocollo è relativo all'invio di *comandi*.

I comandi che un Registrar può inoltrare possono essere di tre tipi:

- Session management – funzioni per stabilire e gestire la sessione di collegamento fra client e server
- Query – comandi di sola lettura per l'interrogazione del database
- Data transform – comandi di lettura e scrittura per l'aggiornamento dei contenuti di un database

Ogni Registrar deve dimostrare di poter gestire per conto del Registro qualsiasi richiesta legata alla creazione e/o alla modifica dei domini; ciò avviene attraverso il processo di accreditamento, nel quale il Registrar viene sottoposto a vari test. Il superamento dei test è strettamente legato al software creato e utilizzato dal Registrar. Di seguito viene progettato un software che consente di simulare il processo di comunicazione tra un Registrar accreditato e il Registro. In particolare si realizzerà un'interfaccia web che consentirà di svolgere il lavoro del Registrar e un layer di back-end che ci consentirà di simulare il lavoro del Registro.

La corretta esecuzione del processo, per il quale il Registro se ne assume la responsabilità, prevede:

- Acquisizione della richiesta relativa al comando inoltrato;
- Interpretazione della richiesta acquisita;
- Esecuzione del comando definito dalla richiesta;
- Invio di una risposta relativa allo stato dell'esecuzione.

1.2 TARGET

L'obiettivo di questo progetto è fornire un'applicativo che possa superare una parte dei test sopra citati evidenziandone **request** e **response** relativa ad ogni operazione. Sarà possibile analizzare le richieste concorrentemente alle operazioni effettuate e per ogni operazione sarà possibile visualizzare la risposta da parte del Registro (assunta sempre positiva) e tutte le informazioni in essa contenute. Ciò permetterà di acquisire una maggiore consapevolezza del procedimento di accreditamento di un Registrar e, più in generale, del processo di comunicazione tra i due principali attori interessati alla creazione di un dominio.

2. DNS NEL CCTLD .IT

2.1 DNS IN BREVE

Tutti i dispositivi su Internet, dallo smartphone al laptop fino al server che carica contenuti su siti Web, si collegano e comunicano tra loro utilizzando un indirizzo IP. Quando apri una pagina Web ed entri in un sito Web, non è necessario tenere a memoria e digitare una lunga stringa numerica. È sufficiente digitare il nome di dominio, ad esempio example.com, per aprire la pagina.

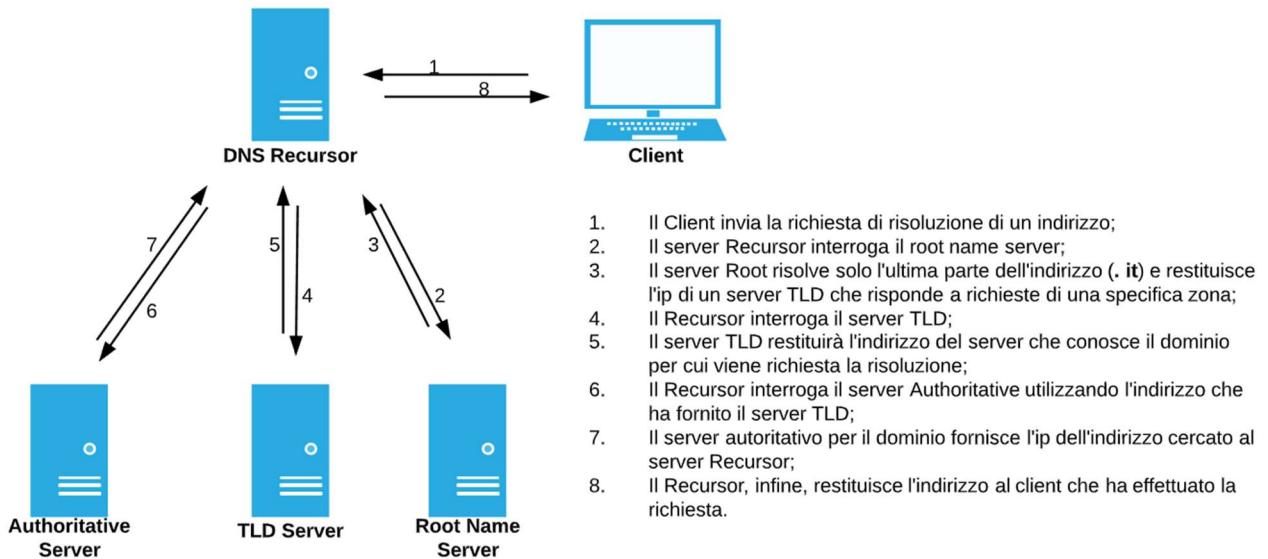
Un servizio DNS è un servizio distribuito globale che traduce nomi comprensibili come www.esempio.com in indirizzi IP numerici come 192.0.2.1, utilizzati dai computer per connettersi tra loro. Il sistema DNS di Internet funziona in modo analogo a una rubrica telefonica, perché gestisce la mappatura tra nomi e numeri. I server DNS traducono le richieste di nomi in indirizzi IP, controllando a quale server si conserverà un utente finale nel momento in cui digiterà un nome di dominio nel suo browser Web.

Data la molteplicità di computer sulla rete internet invece di memorizzare tutti i nomi e i numeri in un solo posto, essi sono stati suddivisi in quelle che vengono chiamate **zone**. Quando occorre risolvere un nome di dominio occorre interrogare il proprio server DNS, questo a sua volta cercherà di recuperare l'informazione interrogando altri server in un processo che viene chiamato **upstream**. La richiesta viene trasmessa da server a server finché non raggiunge il server **autoritativo** per quell'indirizzo che conoscendo l'informazione la trasmette al server che l'ha richiesta.

Esistono 4 tipi di server:

- DNS Recursor: il server che risponde ad una **query DNS** e inoltra la richiesta ricevuta ad altri server a meno che questa informazione non sia stata memorizzata in una memoria cache;
- Root Name Server: un Root Name Server gestisce la *root zone*. Risponde a richieste dirette e può restituire una lista di server autoritativi per il corrispondente dominio di primo livello (top-level domain);
- TLD Name Server: il TLD Name Server (Top Level Domain Name Server) è uno dei più alti livelli, conosce il server autoritativo che può rispondere alla richiesta;
- Authoritative DNS Server: è il server che memorizza l'informazione in maniera corretta e sempre aggiornata. Se nessuno dei server interrogati in precedenza aveva l'informazione memorizzata in memorie cache oppure la versione conservata risultava obsoleta, questo server è quello che deve essere contattato.

Quando un utente accede a un sito web viene inviata una richiesta di risoluzione del nome a dominio al server DNS. Se la corrispondenza tra indirizzo IP non fosse già nota a livello locale, allora il sistema interroga il server DNS impostato dall'utente (nelle impostazioni dell'interfaccia di rete oppure a livello di modem/router/server DHCP). Se il server DNS interpellato non conoscesse l'IP corrispondente al nome a dominio indicato allora si attiva il cosiddetto meccanismo della ricorsione: si parte dalla radice interrogando uno dei server root nel dominio di primo livello, si ottiene il server che lo gestisce, si procede con un'interrogazione nel dominio di secondo livello fino a raggiungere il server autorevole per il nome desiderato.



2.2 DEBOLEZZE DNS

Così come è stato costruito il DNS ha varie debolezze. Alcune caratteristiche del sistema permettono infatti molteplici attacchi. Le caratteristiche principali che comportano rischi per la sicurezza sono le seguenti:

- I server DNS memorizzano, per un periodo, tutti gli indirizzi IP e i nomi dei server per i domini di competenza e li condividono con chiunque li richieda: ciò consente agli hacker di ottenere molte informazioni;
- Le memorie cache dei server possono essere facilmente manipolate: questo consente di effettuare un attacco chiamato **cache poisoning**. Manipolando le cache del server, gli hacker, possono indirizzare gli utenti su un qualsiasi sito;
- I server DNS scambiano informazioni relative alle query con le workstation interne: gli hackers hanno imparato a sfruttare questo comportamento creando canali nascosti con i quali esfiltrare informazioni;
- I server DNS eseguono un controllo molto debole sulle risposte che provengono da altri server, in effetti, basta che sia corretto il numero della transazione (transaction ID);
- I server DNS accettano risposte simultanee alle loro richieste: questo consente agli hacker di fare molteplici tentativi per indovinare il numero di transazione;
- È molto facile effettuare richieste o fornire risposte facendole figurare provenienti da altre sorgenti.

Per questi motivi è nata l'esigenza di creare un protocollo di sicurezza per il DNS. Questa estensione di sicurezza è definita **DNSSEC**.

2.3 DNSSEC

Il protocollo DNSSEC poggia sull'utilizzo della crittografia asimmetrica e, di conseguenza, usa uno schema che prevede l'impiego di due chiavi: una chiave privata e una pubblica. Nella risoluzione di un indirizzo per il soddisfacimento di una qualsiasi richiesta, nel caso di DNSSEC, in caso di disponibilità di **record DS**, viene richiesta la chiave che consente al server di verificare che le informazioni ricevute siano identiche al record presente sul server autoritativo per il nome a dominio indicato.

Se il server ricorsivo determina che il record dell'indirizzo è stato inviato dal server autoritativo e non è stato modificato durante il percorso, il nome a dominio viene risolto e l'utente può accedere al sito (procedura di convalida). Diversamente, se il record fosse stato modificato o non provenisse dall'origine indicata, il server ricorsivo non consente al browser di raggiungere quello che a tutti gli effetti viene considerato un indirizzo fraudolento diverso da quello ufficiale.

2.4 PROCESSO DI ACCREDITAMENTO

Un Registrar, per divenire tale, deve superare il test di accreditamento. Con questo test, il Registrar, deve dimostrare di poter gestire correttamente qualsiasi operazione sui domini per cui egli è responsabile. In particolare il lavoro del Registrar si limita alla raccolta di informazioni per soddisfare una certa operazione e la creazione e l'inoltro di una richiesta in formato *xml* che descriva nei dettagli l'operazione da effettuare. La procedura di accreditamento di un Registrar viene soddisfatta con il superamento di 24 test atti a dimostrare il funzionamento dell'applicativo.

Si riportano, di seguito, quelle operazioni che più hanno significato nella comunicazione e si trascurano operazioni di gestione (ad esempio la modifica della password) e operazioni complesse che esulano dallo scopo dell'applicativo (ad esempio il trasferimento di un dominio da un Registrar ad un altro).

- Handshake;
- Creazione di tre contatti di tipo registrant;
- Creazione di due contatti di tipo tech/admin;
- Aggiornamento di una delle proprietà di un contatto;
- Visualizzazione delle informazioni di un contatto;
- Verifica della disponibilità di due nomi a dominio;
- Visualizzazione delle informazioni di un nome a dominio;
- Modifica del Registrant di un nome a dominio;
- Richiesta di modifica del Registrar di un nome a dominio;
- Nuova richiesta di modifica del Registrar di un nome a dominio;
- Richiesta di modifica del Registrant contestuale ad una modifica del Registrar per un nome a dominio;
- Cancellazione di un nome a dominio;
- Cancellazione di un contatto.

2.5 PROTOCOLLO EPP

Normalmente la comunicazione tra Registro e Registrar avviene tramite lo scambio di **command** e **response** EPP scritte in XML. Il protocollo EPP è stato appositamente creato per garantire la comunicazione sincrona fra il Registro, o più in generale fra un **domain name registry**, e i Registrar. L'EPP è basato su XML, per questo motivo nell'applicativo creato viene usato questo stesso metalinguaggio per descrivere i command e le response.

Questo protocollo fornisce dei servizi base, che sono:

- Service Discovery: supporto all'individuazione dei servizi offerti.
- Command e Response: invio di comandi e risposta all'invio dei comandi.
- Extension: framework di supporto per la definizione degli oggetti gestiti e delle relazioni del protocollo di richiesta e di risposta da utilizzare.

Si riporta, di seguito, un esempio di command e response EPP definito dall'IETF:

Command di **update**, agli elementi standard dell'EPP, per l'inoltro di un command di tipo update, occorre definire anche un elemento *obj:update*. Questo elemento consente di identificare l'oggetto che deve essere aggiornato e gli elementi necessari per modificare l'oggetto.

```
C:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
C:<epp xmlns="urn:ietf:params:xml:ns:epp-1.0">
C:  <command>
C:    <update>
C:      <obj:update xmlns:obj="urn:ietf:params:xml:ns:obj">
C:        <!-- Object-specific elements. -->
C:      </obj:update>
C:    </update>
C:    <clTRID>ABC-12346</clTRID>
C:  </command>
C:</epp>
```

Response di **update**:

```
S:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
S:<epp xmlns="urn:ietf:params:xml:ns:epp-1.0">
S:  <response>
S:    <result code="1000">
S:      <msg>Command completed successfully</msg>
S:    </result>
S:    <trID>
S:      <clTRID>ABC-12346</clTRID>
S:      <svTRID>54322-XYZ</svTRID>
S:    </trID>
S:  </response>
S:</epp>
```

C: Client, colui che invia il *command*

S: Server, colui che invia la *response*

3. ECOSISTEMA APPLICAZIONE

3.1 TECNOLOGIE

L'applicativo viene implementato con **Java Server Faces** su **Eclipse JEE**. Il back-end viene implementato utilizzando **Spring Framework** per la gestione delle dipendenze ed in generale per rendere più semplice la gestione del web context. Per quanto riguarda il front-end viene utilizzato il framework **Primefaces**. Le pagine della web-app sono scritte in **xhtml** e sono supportate da **stylesheet CSS**.

Lo strato di persistenza del Registro viene gestito da un database MySQL, il database viene denominato **dnsEntities** e conserva 3 tabelle principali: **contacts**, **domains** e **contracts**.

È stato scelto **Apache Tomcat** come *Web Container* e per garantire la connessione al database viene definito un *datasource* direttamente sul server. Quando il server è online, l'*applicationContext* gestisce automaticamente la connessione e consente la comunicazione con lo strato di persistenza. La creazione e il popolamento iniziale del database è stato eseguito con MySQL, mentre il recupero del dataSource è stato ottenuto con tramite **JNDI**. L'utilizzo della Java Naming and Directory Interface risulta molto efficace per lo sviluppo di web app poiché consente di nascondere la natura e i dettagli della risorsa che viene utilizzata. Questo consente di creare astrazione e permette un utilizzo di un database differente con poche semplici modifiche.

3.2 LIBRERIE

L'applicativo fa uso di molteplici librerie esterne, queste consentono una corretta implementazione dei meccanismi e un alleggerimento del carico di lavoro.

- **javax.util**
- **javax.xml**: per la scrittura degli xml che compongono command e response;
- **javax.servlet**: per l'acquisizione dell'application context e per la creazione di un ApplicationContextProvider utile all'applicativo;
- **javax.faces**: per la corretta definizione dei bean e per la gestione dello scope dei bean;
- **javax.sql**: per la gestione del Datasource e la gestione della connessione al database;
- **org.w3c.dom**: per la gestione dei document e degli element costituenti il corpo del messaggio del command o della response;
- **org.apache.commons.lang**: per classi di utility nella gestione della scrittura degli xml;
- **org.springframework**: per la gestione del context della web app, per l'utilizzo di keyword che definiscono i componenti e i loro ruoli e per la definizione dei beans;
- **org.primefaces**: per la gestione degli eventi in front-end;

3.3 DIPENDENZE

L'applicativo fa uso di varie dipendenze, viene utilizzato Maven per gestirle quindi esse sono tutte presenti nel file *pom.xml* esistente nel progetto. Di seguito il file:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>Regapp</groupId>
    <artifactId>Regapp</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>war</packaging>

    <dependencies>
        <!-- https://mvnrepository.com/artifact/javax.servlet/servlet-api -->
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>javax.servlet-api</artifactId>
            <version>4.0.0</version>
        </dependency>

        <dependency>
            <groupId>org.webjars</groupId>
            <artifactId>font-awesome</artifactId>
            <version>5.12.0</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/org.springframework/spring-context
-->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>5.2.4.RELEASE</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/org.springframework/spring-web -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-web</artifactId>
            <version>5.2.4.RELEASE</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/org.primefaces/primefaces -->
        <dependency>
            <groupId>org.primefaces</groupId>
            <artifactId>primefaces</artifactId>
            <version>8.0</version>
        </dependency>

        <dependency>
            <groupId>com.sun.faces</groupId>
            <artifactId>jsf-api</artifactId>
            <version>2.2.16</version>
        </dependency>

        <dependency>
            <groupId>com.sun.faces</groupId>
            <artifactId>jsf-impl</artifactId>
```

```

        <version>2.2.16</version>
    </dependency>

    <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>8.0.19</version>
    </dependency>

    <dependency>
        <groupId>org.glassfish.web</groupId>
        <artifactId>el-impl</artifactId>
        <version>2.2</version>
    </dependency>

    <!-- https://mvnrepository.com/artifact/commons-lang/commons-lang -->
    <dependency>
        <groupId>commons-lang</groupId>
        <artifactId>commons-lang</artifactId>
        <version>2.6</version>
    </dependency>

</dependencies>

<build>
    <sourceDirectory>src</sourceDirectory>
    <plugins>
        <plugin>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.8.0</version>
            <configuration>
                <source>1.8</source>
                <target>1.8</target>
            </configuration>
        </plugin>
        <plugin>
            <artifactId>maven-war-plugin</artifactId>
            <version>3.2.3</version>
            <configuration>
                <warSourceDirectory>WebContent</warSourceDirectory>
            </configuration>
        </plugin>
    </plugins>
</build>
</project>
```

4. PROGETTAZIONE

4.1 DATA LAYER

Lo strato di persistenza è il layer più vicino ai dati. Sebbene le informazioni necessarie all'applicativo siano poche, è stato gestito con un modello relazionale per evidenziare quanto le entità gestite e le informazioni conservate siano dipendenti l'un l'altra. Viene utilizzato **MySQL** come DBMS e le tabelle vengono inizializzate in modo da poter effettuare tutte le operazioni con l'applicativo.

Lo schema ER seguente è alla base della progettazione del database e rappresenta le tabelle utilizzate dal database e le loro relazioni, includendo la molteplicità e alcuni dei vincoli di integrità.

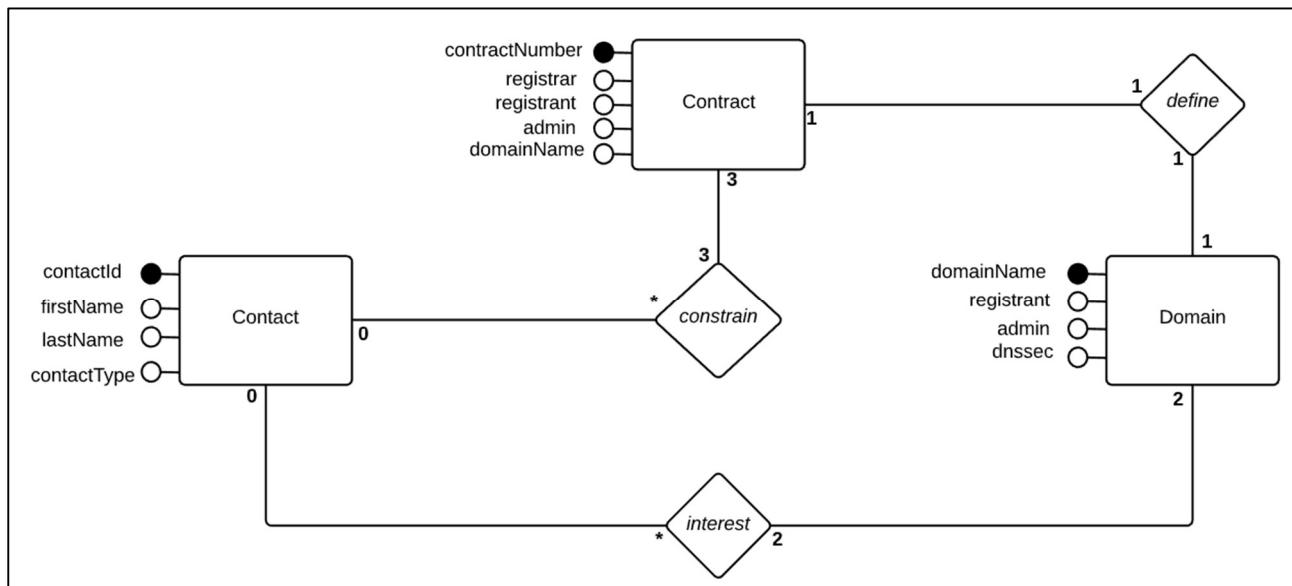


Figura 1: Schema ER del database

Gli script utilizzati per la predisposizione delle relazioni sono i seguenti:

```
-- Query creazione Database, il nome non è vincolato
CREATE DATABASE dnsentities;

USE dnsentities;

-- Tabella contacts che gestisce i contatti, è importante che gli id abbiano 16
caratteri (codice fiscale)
CREATE TABLE contacts (
    idContact CHAR(16) PRIMARY KEY,
    firstName VARCHAR(48) NOT NULL,
    lastName VARCHAR(48) NOT NULL,
    contactType VARCHAR(48) NOT NULL
);

-- Tabella domains per la gestione dei domini
CREATE TABLE domains (
    domainName VARCHAR(64) PRIMARY KEY,
    registrant CHAR(16) NOT NULL,
    admin CHAR(16) NOT NULL,
    dnssec BOOLEAN
);

-- Tabella contracts per la gestione dei contratti
```

```

CREATE TABLE contracts (
    contractNumber INT PRIMARY KEY AUTO_INCREMENT,
    registrar CHAR(16) NOT NULL,
    registrant CHAR(16) NOT NULL,
    admin CHAR(16),
    domainName VARCHAR(64) NOT NULL
);
-- Popolamento con valori validi la tabella contacts
INSERT INTO contacts(idContact, firstName, lastName, contactType)
VALUES
    ('RCCRCC90A01A111A', 'Rocco', 'Rossi', 'Registrar'),
    ('AGGRCC90A01A111A', 'Giovanni', 'Rossi', 'Registrant'),
    ('GGILNN90A01A111A', 'Giuseppe', 'Bianchi', 'Admin'),
    ('FBLAPS90A01A111A', 'Fabiana', 'Di Netta', 'Tech'),
    ('RSARCC90A01A111A', 'Rosa', 'Verdi', 'Admin')
);
-- Popolamento domains
INSERT INTO domains(domainName, registrant, admin, dnssec)
VALUES
    ('www.test.it', 'AGGRCC90A01A111A', 'GGILNN90A01A111A', 1),
    ('www.notSec.it', 'AGGRCC90A01A111A', 'RSARCC90A01A111A', 0),
    ('www.test2.it', 'AGGRCC90A01A111A', 'GGILNN90A01A111A', 0)
);
-- Popolamento contracts
INSERT INTO contracts(registrar, registrant, admin, domainName)
VALUES
    ('RCCRCC90A01A111A', 'AGGRCC90A01A111A', 'GGILNN90A01A111A', 'www.test.it'),
    ('RCCRCC90A01A111A', 'AGGRCC90A01A111A', 'RSARCC90A01A111A', 'www.notSec.it')
);

```

I vincoli di integrità intrarelazionali e interrelazionali sono i seguenti:

1. contacts.idContact **PRIMARY KEY**;
2. domains.domainName **PRIMARY KEY**;
3. contactType='REGISTRAR' **OR** contactType='REGISTRANT' **OR** contactType='ADMIN' **OR** contactType='TECH';
4. **FOREIGN KEY** (domains.registrant) **REFERENCES** contacts(idContact);
5. **FOREIGN KEY** (domains.admin) **REFERENCES** contacts(idContact);
6. **FOREIGN KEY** (contracts.registrar) **REFERENCES** contacts(idContact);
7. **FOREIGN KEY** (contracts.registrant) **REFERENCES** contacts(idContact);
8. **FOREIGN KEY** (contracts.admin) **REFERENCES** contacts(idContact);
9. **FOREIGN KEY** (contracts.domainName) **REFERENCES** domains(domainName);

Perché sia possibile accedere al database per il recupero delle informazioni, l'applicazione utilizza un datasource definito come segue sul web container:

```
<Resource name="jdbc/dnsentities" auth="Container"
    type="javax.sql.DataSource" driverClassName="com.mysql.cj.jdbc.Driver"
    url="jdbc:mysql://localhost:3306/dnsentities?serverTimezone=UTC"
    maxTotal="15" username="root" password="root" maxIdle="3" />
```

Per quanto riguarda il recupero del datasource, per fare uso di JNDI occorre definire, nell'applicationContext, il nome della risorsa:

```
<jee:jndi-lookup id="DataSource" jndi-name="jdbc/dnsentities" expected-
    type="javax.sql.DataSource"/>
```

Per il trasferimento delle entità tra business layer e persistence layer, è stato applicato il pattern DTO (**Data Transfer Object**). Definire una classe specifica per ogni entità rende molto semplice la gestione delle informazioni ottenute in front-end e il salvataggio delle entità sul database.

ContactDTO	DomainDTO	ContractDTO
<pre>-idContact: String -firstName: String -lastName: String -contactType: ContactTypeEnum +ContactDTO(String, String, String, ContactTypeEnum) +ContactDTO() +hashCode(): int >equals(Object): boolean +toString(): String</pre>	<pre>-domainName: String -registrant: String -admin: String -dnssec: boolean +DomainDTO(String, String, String, boolean) +DomainDTO() +hashCode(): int >equals(Object): boolean +toString(): String</pre>	<pre>-domainName: String -registrant: String -admin: String -registrar: String +ContractDTO() +ContractDTO(String, String, String, String)</pre>

Figura 2: DTO per la gestione delle entità di base

Contact: i contatti possono essere di vari tipo: Registrar, Registrant, Tech e Admin. Per gestire la tipologia dei contatti all'interno dell'applicativo viene utilizzata una classe Enum: **ContactTypeEnum**. Le caratteristiche dei contact devono essere utilizzate per la creazione delle request XML per la corretta scrittura della request.

- Registrant: colui che utilizza il dominio per i propri scopi e usufruisce dei servizi del Registro mediante corrispettivo economico. Per ogni operazione si riferisce al proprio Registrar;
- Registrar: colui che opera per conto del Registro per effettuare tutte le operazioni sul dominio. Ha un contratto commerciale attivo con il Registrant;
- Admin e Tech: personale che diritti e doveri diversi, operano per conto del Registrar.

Il DTO che gestisce i contatti è ContactDTO.

Domain: gli oggetti Domain ci consentono di identificare tutte le caratteristiche dei vari domini, in particolare occorre conoscere il nome di dominio, il Registrar che gestisce il dominio, il Registrant che ha diritti su quel dominio, il nome dell'amministratore che può agire per conto del Registrar per effettuare modifiche a tale dominio, ed infine il nome degli 0 o N tecnici che possono effettuare, per conto del Registrar, varie operazioni.

Del dominio occorre conoscere e gestire anche un'informazione relativa alla sicurezza del dns. Un dominio con estensione di sicurezza si dice **DNSSEC** e questa informazione viene gestita nel business con un parametro boolean e nelle request XML vi è una differenza nell'intestazione della request e delle response ad eventuali interrogazioni su questo tipo di domini.

Il DTO che gestisce i domini è DomainDTO.

Contract: un Contract rappresenta la relazione e i vincoli che intercorrono tra un Registrar ed un Registrant. Un contratto è identificato univocamente da un valore intero, deve avere un riferimento ad un Registrant, uno al Registrar ed uno al Domain che quel contratto va a definire.

Il DTO che gestisce i contratti è ContractDTO.

Nel business layer l'accesso alle entità fondamentali dell'applicazione viene facilitato dall'esistenza dei DAO (**Data Access Object**). In particolare vengono usate interfacce per impostare la definizione dei metodi CRUD e una classe astratta per gestire gli oggetti **Statement** e **ResultSet**. Ogni DAO deve implementare la propria interfaccia e può estendere la classe astratta per facilitare la gestione della connessione al database.

Le classi sono definite come segue:

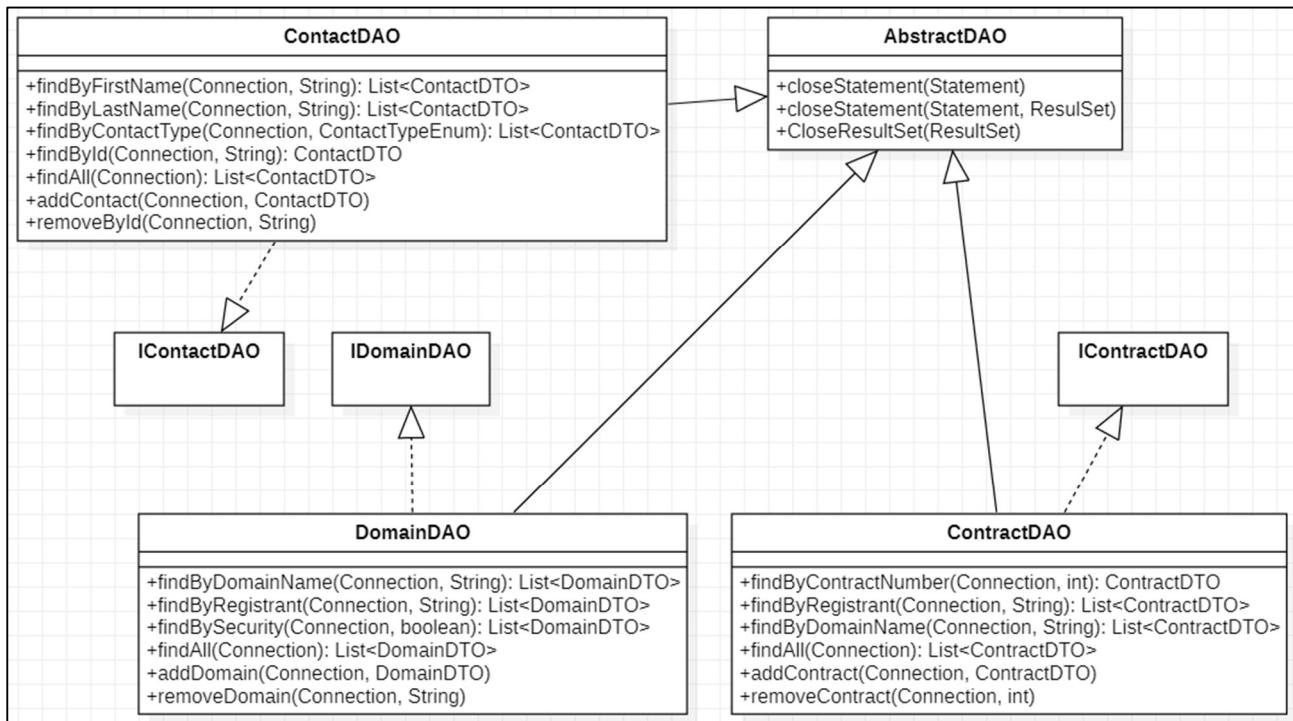


Figura 3: UML dei DAO che consentono l'interfacciamento con il database

4.2 BUSINESS LAYER

Nel business layer vengono definiti una serie di servizi che aiutano a gestire le entità sul database. Questa stratificazione consente di separare i livelli logici e permette una più facile gestione delle classi e il recupero delle entità sullo strato di persistenza. Infatti ciò consente ad ogni strato di interfacciarsi solo con lo strato successivo e quello precedente, consentendo, ad esempio, ai **backing bean** di occuparsi solo del recupero delle informazioni in front-end e utilizzare i service ogni volta occorre utilizzare utility di business. A questo scopo, i servizi sono implementati cercando di limitare la loro responsabilità ad una singola attività per facilitarne la scrittura e migliorarne la gestione. Andando nel merito dei Service, questi sono gestiti da una classe astratta, comune a tutti i service, la quale gestisce la creazione della connessione e il rilascio della stessa consentendo di effettuare il *commit* o il *rollback* a seconda dell'esigenza. Ogni service implementa un'interfaccia e gestisce una sola entità.

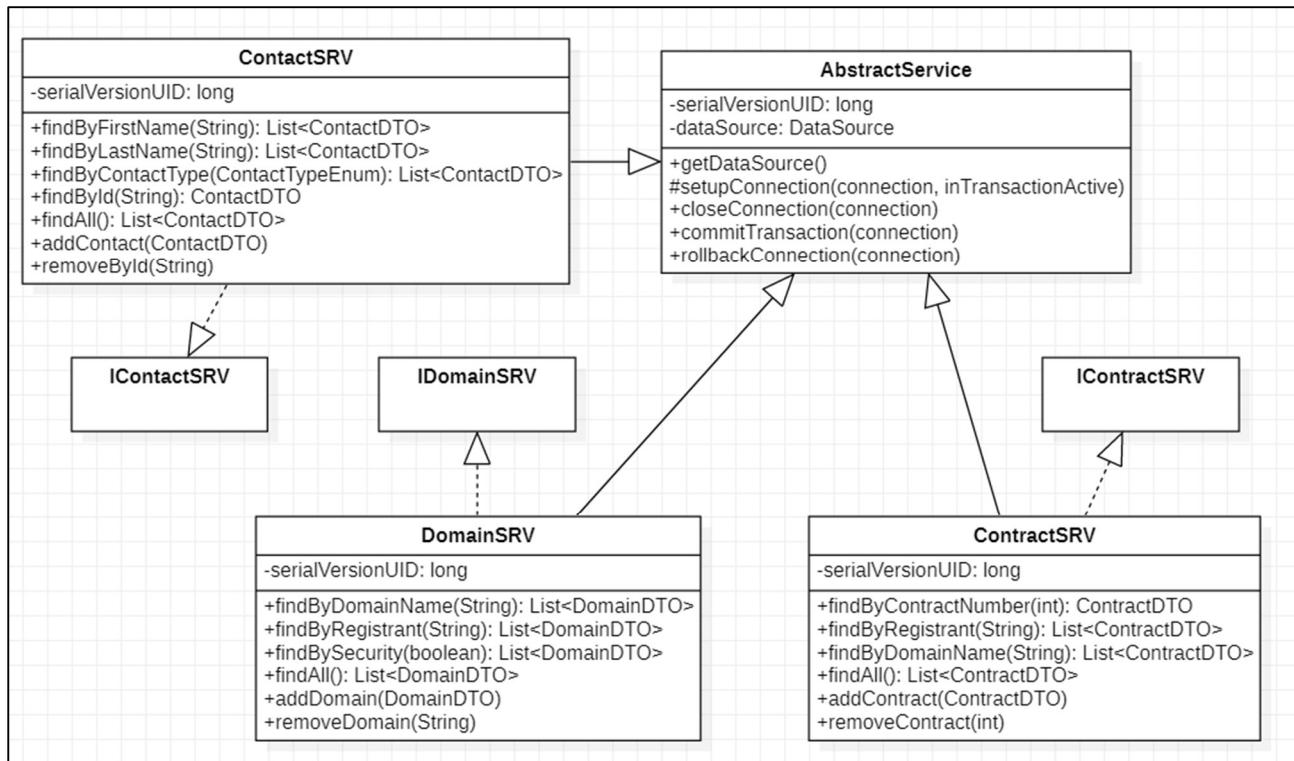


Figura 4: UML delle classi che definiscono i servizi per la comunicazione con il database

I command e le response vengono implementati nell'applicativo con una serie di servizi che, a partire dalle entità basilari, permettono di generare un file di testo, in formato xml, che rappresenta i messaggi che veicolano le request e le response. Per ogni operazione possibile, viene fornita la possibilità di generare il command e visualizzarlo in una finestra in sovrapposizione. Al completamento dell'operazione è possibile visualizzare la response ottenuta a partire dalle informazioni contenute nel command e alle informazioni già possedute grazie allo strato di persistenza. Esiste un service utilizzato per la creazione dei command, esso si basa su varie classi che permettono la facile generazione degli xml, lavorando sui singoli elementi. In particolare esistono due Factory per la creazione dei tag XML necessari alla costruzione dei command, e la classe *RequestBuilder* (risp. *ResponseBuilder*) che permette la creazione della *Request* (risp. *Response*) utilizzando i metodi ereditati dalla Factory.

Il SRV si occuperà di fornire gli oggetti necessari alla creazione dei command a queste classi, e infine veicolerà il messaggio ottenuto in front-end permettendone la visualizzazione all'utente.

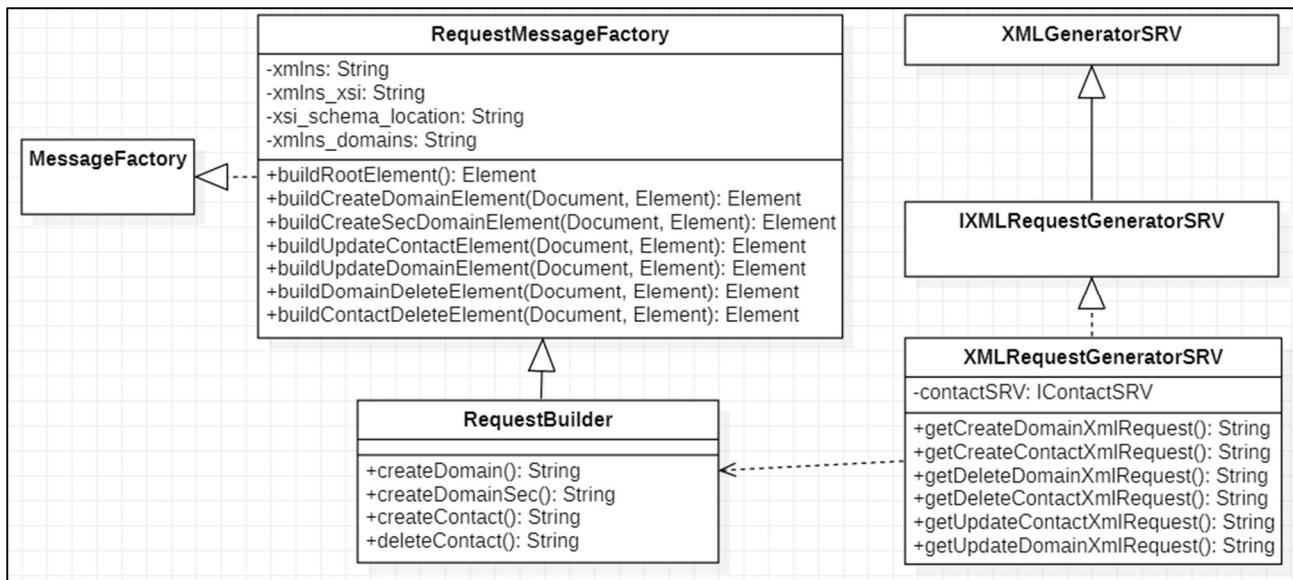


Figura 5: UML delle classi che gestiscono la creazione dei messaggi per la creazione dei command e delle response

Prima di raggiungere il front-end, l'applicativo fa uso di **ManagedBean** che consentono di gestire le view che compongono le pagine della web-app. Ogni bean viene gestito in modalità **ViewScoped**, questo consente un'alleggerimento del carico e una migliore gestione della memoria e in generale di tutte le risorse poiché il bean viene allocato in memoria solo quando viene utilizzata una view che ne fa utilizzo. Ogni bean utilizza i service, appositamente costruiti a grana fine, per fare uso di tutte le utility necessarie al recupero di informazioni che rendono l'applicazione user-friendly e molto interattiva. Esiste un'associazione 1 a 1 tra bean e view, ogni bean viene utilizzato da una specifica view che mostra all'utente tutte le informazioni di back-end lavorate e adattate alle esigenze del caso.

4.3 PRESENTATION LAYER

L'applicazione è costituita da tre pagine scritte in *xhtml*, ognuna delle quali si occupa di gestire *command* in riferimento ad una specifica entità e una pagina principale che permette di effettuare le scelte iniziali. Viene utilizzato il framework Primefaces per una più facile creazione dell'interfaccia e **AJAX** per la gestione degli eventi, molteplici componenti di Primefaces nascondono l'utilizzo di AJAX ma fanno comunque capo a funzioni asincrone scritte in **javascript**, questo velocizza l'applicativo e consente una maggiore efficienza quando non occorre fare operazioni complesse che coinvolgono il back-end.

La pagina principale consente di osservare un diagramma interattivo che descrive sintatticamente le tabelle esistenti in database e le loro relazioni e tre pulsanti che consentono la navigazione tra le pagine.

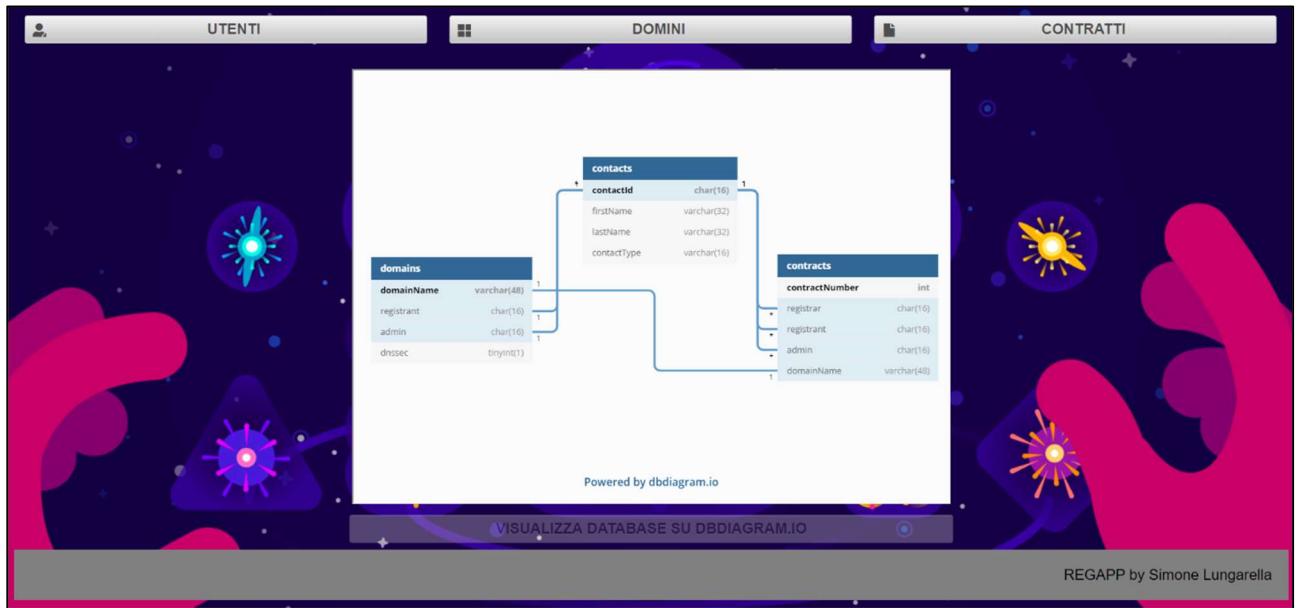


Figura 6: Homepage

Le operazioni *CRUD* sui Contacts sono garantite da una pagina di navigazione composta da tre sezioni. Nella prima viene data la possibilità di ricercare tutti gli utenti esistenti sul database, la seconda ne consente la creazione e infine l'ultima ne consente modifica ed eliminazione. Per ognuna di queste operazioni, oltre a rendere persistenti le entità sul database è possibile visualizzare Request e Response generate. Gestire tutte le operazioni tramite maschera consente, inoltre, di gestire la validazione degli xml che compongono Request e Response garantendo un controllo sui dati rapido e offrendo la possibilità di correzione all'utente. È da sottolineare, infatti, che tutti i command, durante il processo di comunicazione con il Registro, vanno posti a validazione. I controlli che devono essere fatti sono molteplici e riguardano tutti i command. Ad esempio, è richiesto che un nome di dominio non contenga parole di lunghezza superiore a 63 caratteri, questo è un esempio di validazione che può essere fatta *on the fly* durante la creazione della Request, senza necessità di aspettare la Response da parte del Registro per comunicare un errore nel formato del messaggio.

La maschera di ricerca dei contatti è composta da un *datatable* contenente tutte le informazioni sugli utenti ricercati ed una sezione di ricerca da popolare per effettuare la ricerca come viene mostrato in *figura 7*. La maschera di modifica, riportata in *figura 8*, mostra come l'utente può modificare i contatti, all'esecuzione dell'operazione vengono forniti messaggi di supporto all'utente e la possibilità di visualizzare il command generato.

The screenshot shows a user interface for searching contacts. At the top, there are three tabs: 'Ricerca' (Search), 'Creazione' (Creation), and 'Modifica' (Modification). Below the tabs, there are four input fields: 'Nome' (Name) containing 'Katrina', 'Cognome' (Surname) which is empty, 'Codice fiscale' (Tax code) which is empty, and 'Tipo contatto' (Contact type) set to 'Registrar'. A 'Cerca' (Search) button is located below these fields. Below the search bar is a table with columns: Nome, Cognome, Codice fiscale, and Tipo contatto. One row is visible in the table, showing 'Katrina' in the Nome column, 'Maala' in the Cognome column, 'KNTLSI2049SIO30S' in the Codice fiscale column, and 'REGISTRANT' in the Tipo contatto column. Navigation buttons for the table are at the bottom of the table area.

Figura 7: Ricerca contatti

The screenshot shows a user interface for modifying contacts. At the top, there are three tabs: 'Ricerca' (Search), 'Creazione' (Creation), and 'Modifica' (Modification). On the left, there is a sidebar titled 'Utenti esistenti' (Existing users) listing several contact entries. In the main area, there are four input fields: 'Nome' (Name) containing 'Katrina', 'Cognome' (Surname) containing 'Maala', 'Codice fiscale' (Tax code) containing 'KNTLSI2049SIO30S', and 'Tipo del contatto' (Contact type) set to 'Registrant'. Below these fields is a 'Modifica' (Modification) button. Navigation buttons for the sidebar are at the bottom of the sidebar area.

Figura 8: Modifica di contatti

Lo scopo principale dell'applicazione è quello di dimostrare come Registrar e Registro comunichino cercando di simulare al meglio tutti gli aspetti che riguardano la comunicazione. L'applicazione, però, è stata realizzata cercando di non sottovalutare l'importanza di una buona user experience, infatti sono presenti molteplici strumenti per la comprensione dell'argomento, tra tutti, i popup informativi, presenti su ogni pagina principale dell'applicativo, che spiegano brevemente quello su cui si sta operando.



Figura 9: Info popup per informazioni su contatti



Figura 10: Popup informativo

La gestione dei domini viene mostrata in maniera simile a quella dei contatti, c'è un'ulteriore sezione per quanto riguarda i domini ed è la verifica.

Il database WHOIS è la banca dati nella quale vengono raccolte le informazioni relative ai titolari dei nomi di dominio. Nell'applicativo questo servizio è stato simulato per fornire tale servizio ma con uno scope locale legato solo al proprio DB. Il servizio consente di verificare la possibilità di utilizzo di un nome di dominio, nel caso in cui tale nome non sia disponibile, vengono recuperate le informazioni relative al dominio e vengono visualizzate nell'interfaccia grafica. In caso contrario viene mostrato un messaggio che invita l'utente alla creazione del dominio, come riportato di seguito.

The screenshot shows a web-based WHOIS search interface. At the top, there are four tabs: Verifica (selected), Ricerca, Creazione, and Modifica. Below the tabs is a search input field with placeholder text: "Verifica che il dominio che vuoi registrare sia disponibile". The input field contains "www. test .it". A blue "Verifica" button is positioned below the input field. The main content area is titled "RICERCA WHOIS". It displays a message: "Dominio non disponibile, il dominio è già stato utilizzato." Below this message is a table with four columns: Dominio, Registrant, Admin, and DNSSEC. The table row for "www.test.it" shows: Dominio (www.test.it), Registrant (LNGSMN96D07A399N), Admin (RFLOSP29SP20GI2O), and DNSSEC (Si).

Figura 11: Ricerca WHOIS con esito positivo

The screenshot shows a web-based WHOIS search interface. At the top, there are four tabs: Verifica (selected), Ricerca, Creazione, and Modifica. Below the tabs is a search input field with placeholder text: "Verifica che il dominio che vuoi registrare sia disponibile". The input field contains "www. testNonUsato .it". A blue "Verifica" button is positioned below the input field. The main content area is titled "RICERCA WHOIS". It displays a message: "Esegui una ricerca per verificare che il dominio sia disponibile". Below this message is another message: "Dominio disponibile, puoi procedere alla creazione nella sezione di Creazione".

Figura 12: Ricerca WHOIS con esito negativo

Per quanto riguarda la creazione di un dominio, la sezione che viene utilizzata è implementata come mostrato nelle figure 13, 14 e 15. Nella GUI dell'applicazione vengono mostrate tutte le informazioni relative alla creazione e successivamente viene mostrata la Request generata, infine l'entità viene salvata in DB per consentirne la visualizzazione all'utente.

Figura 13: Maschera di creazione Domain



Figura 14: Dialog di selezione Admin

Questa maschera consente la selezione della tipologia del dominio tra DNS o DNSSEC, la selezione dei contatti di tipo Registrant e Admin da scegliere tra i contatti di tale tipo esistenti in DB locale. Viene omessa la scelta del Registrar poiché questa è un'app ad uso di un Registrar pertanto esso non può essere modificato.

Al click di Registrant e Admin vengono mostrati tutti i contatti di tale tipologia esistenti in database e ne viene consentita la selezione. Questi saranno i contatti relativi al dominio e verranno memorizzati in DB e visualizzati nella Request.

Al completamento dei campi del form, l'utente clicca sul pulsante ‘Crea’ per visualizzare la Request che viene immediatamente generata. La Request differisce molto in base al tipo di dominio che viene creato. Un dominio DNSSEC possiede molteplici tag aggiuntivi, essi consentono di fornire tutte le informazioni in merito alla sicurezza.

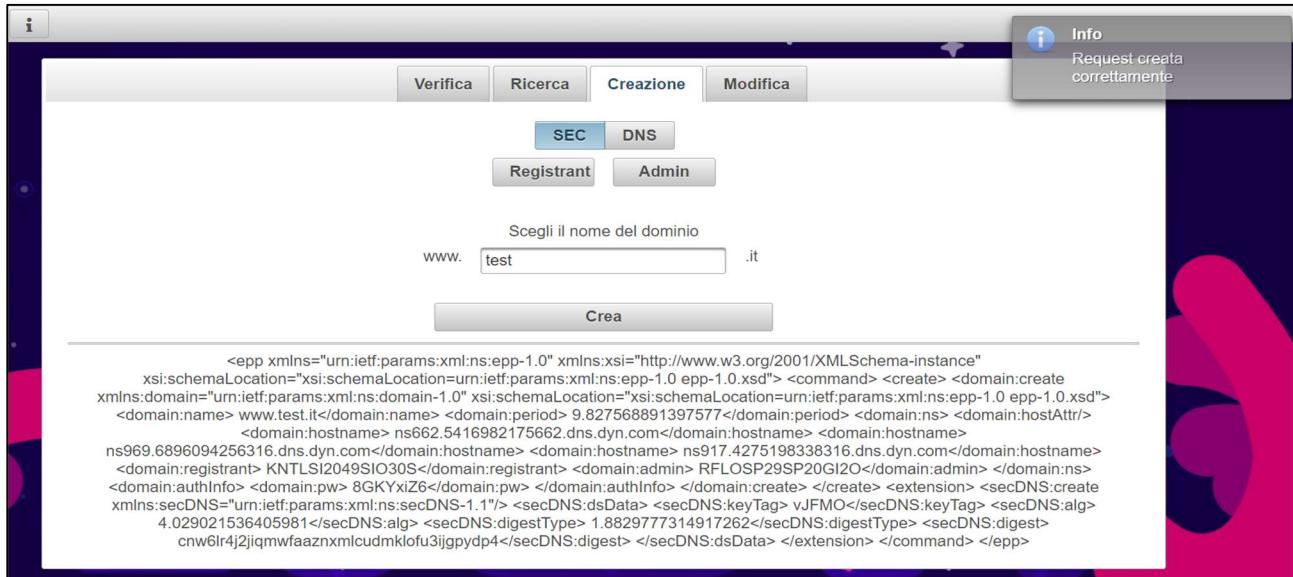


Figura 15: Request generata

5. IMPLEMENTAZIONE

Viene riportato il codice delle classi più importanti e più rappresentative dell'applicativo, per questioni di leggibilità non vengono riportate le classi dei *beans*, i file xhtml che definiscono le *views*, inoltre non vengono riportate classi di utility e classi specifiche per tutti gli oggetti lavorati.

Per la visualizzazione del codice completo si riporta il lettura alla repository pubblica disponibile su GitHub raggiungibile tramite l'indirizzo: <https://github.com/Surveior/Regapp.git>

5.1 APPLICATION CONTEXT

Il file, consente di definire alcuni aspetti dell'applicativo, in particolare viene definito l'indirizzo di lookup del DataSource per il recupero della connessione al database MySQL.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:jee="http://www.springframework.org/schema/jee"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-4.3.xsd
        http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/mvc/spring-mvc-4.3.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-4.3.xsd
        http://www.springframework.org/schema/jee
        http://www.springframework.org/schema/jee/spring-jee-4.3.xsd">

    <jee:jndi-lookup id="DataSource" jndi-name="jdbc/dnsentities" expected-
    type="javax.sql.DataSource"/>
    <context:component-scan base-package="it.business" />
</beans>
```

5.2 FACES CONFIG

Ogni applicazione JSF utilizza questo file come configurazione. Descrive alcune proprietà come le navigation rules tra le pagine, bundle dei messaggi e altre informazioni, ne viene riportato il contenuto in quanto questo file è uno dei più importanti delle applicazioni JSF.

```
<?xml version="1.0" encoding="UTF-8"?>
<faces-config version="2.2" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
        http://xmlns.jcp.org/xml/ns/javaee/web-facesconfig_2_2.xsd">
<application>
    <locale-config>
        <default-locale>it</default-locale>
        <supported-locale>it</supported-locale>
    </locale-config>
    <message-bundle>resources.application</message-bundle>
    <el-resolver>org.springframework.web.jsf.el.SpringBeanFacesELResolver</el-resolver>
</application>

</faces-config>
```

5.3 WEB

Altre informazioni e proprietà dell'applicazione sono definite nel file web.xml.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
    version="3.1">

    <display-name>Regapp</display-name>

    <servlet>
        <servlet-name>JSFServlet</servlet-name>
        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>JSFServlet</servlet-name>
        <url-pattern>*.jsf</url-pattern>
        <url-pattern>*.xhtml</url-pattern>
    </servlet-mapping>

    <listener>
        <listener-class>
            org.springframework.web.context.ContextLoaderListener
        </listener-class>
    </listener>

    <welcome-file-list>
        <welcome-file>homepage.jsf</welcome-file>
    </welcome-file-list>
</web-app>
```

5.4 APPLICATION CONTEXT PROVIDER

Questa classe consente di gestire il contesto dell'applicazione

```
package it.business.utils;

import javax.servlet.ServletContextEvent;

import org.springframework.context.ApplicationContext;
import org.springframework.context.ApplicationContextAware;
import org.springframework.stereotype.Component;
import org.springframework.web.context.support.WebApplicationContextUtils;

@Component
public class ApplicationContextProvider implements ApplicationContextAware {
    private static ApplicationContext context;

    private static Object Lock = new Object();

    @Override
    public final void setApplicationContext(final ApplicationContext ctx) {
        synchronized (Lock) {
            context = ctx;
        }
    }

    public static ApplicationContext getApplicationContext() {
        return context;
    }

    public static void setApplicationContextForTestPurpose(final ApplicationContext
ctx) {
        context = ctx;
    }

    public static void setApplicationContext(final ServletContextEvent event) {
        context =
WebApplicationContextUtils.getWebApplicationContext(event.getServletContext());
    }
}
```

5.5 DATA TRANSFER OBJECTS

Si riporta il DTO dell'oggetto *CONTACT*, tutti i DTO sono implementati seguendo la stessa logica, sono stati dotati di un metodo *toString()* adeguato ed è stata gestita la possibilità di comparazione degli oggetti.

```
package it.business.dto;

import it.business.enums.ContactTypeEnum;

/**
 * @author Simone Lungarella
 *
 * DTO per la gestione dell'entità CONTACT, questo DTO aiuta a gestire l'entità sul DB
 * salvata nella tabella CONTACTS e permette
 * di trasferire le informazioni dallo strato di persistenza al business al front-end
 */

public class ContactDTO {
    private String contactId;
    private String firstName;
    private String lastName;
    private ContactTypeEnum contactType;

    public ContactDTO() {
        this("", "", "", null);
    }

    public ContactDTO(String contactId, String firstName, String lastName,
ContactTypeEnum contactType) {
        setContactId(contactId);
        setFirstName(firstName);
        setLastName(lastName);
        setContactType(contactType);
    }

    public ContactDTO(ContactDTO contactCopy) {
        setContactId(contactCopy.getContactId());
        setFirstName(contactCopy.getFirstName());
        setLastName(contactCopy.getLastName());
        setContactType(contactCopy.getContactType());
    }

    public String getContactId() {
        return contactId;
    }

    public void setContactId(String contactId) {
        this.contactId = contactId;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }
```

```

}

public void setLastName(String lastName) {
    this.lastName = lastName;
}

public ContactTypeEnum getContactType() {
    return contactType;
}

public void setContactType(ContactTypeEnum contactType) {
    this.contactType = contactType;
}

@Override
public int hashCode() {
    final int prime = 31;
    int result = 1;
    result = (contactId == null) ? 0 : prime * result + contactId.hashCode();
    return result;
}

@Override
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    ContactDTO other = (ContactDTO) obj;
    if (hashCode() == 0) {
        if (other.hashCode() != 0)
            return false;
    } else if (this.hashCode() != other.hashCode())
        return false;
    return true;
}

@Override
public String toString() {
    return "Codice fiscale: " + contactId + ", Nome: " + firstName +
        ", Cognome: " + lastName + ", Tipo di contatto: " + contactType.toString();
}
}

```

5.6 SERVIZI

La classe AbstractService gestisce tutto quello che viene usato da tutti i service. Viene estesa ad ogni creazione di un service e permette di gestire il *dataSource* e la connessione al database.

```
package it.business.service;

import java.io.Serializable;

/**
 * @author Simone Lungarella
 *
 *         Service che viene esteso da tutti i service che verranno utilizzati,
 *         implementa tutto ciò di cui hanno bisogno i services
 */

@Service
@Component
public abstract class AbstractService implements Serializable {

    private static final long serialVersionUID = -2017286881829397007L;

    @Autowired
    @Qualifier("DataSource")
    private DataSource dataSource;

    public DataSource getDataSource() {
        return dataSource;
    }

    /**
     * Consente di creare una connessione scegliendo da una tipologia che prevede
     * l'auto commit ad ogni operazione e una strategia che consente di effettuare
     * il commit manuale o il rollback delle azioni a seconda della necessità
     */
    protected final Connection setupConnection(final Connection connection, final
Boolean inTransactionActive) throws SQLException {
        if (inTransactionActive) {

            connection.setTransactionIsolation(Connection.TRANSACTION_READ_COMMITTED);
            connection.setAutoCommit(false);
        }
        return connection;
    }

    public final void closeConnection(final Connection connection) {
        try {
            if (connection != null && !connection.isClosed()) {
                connection.close();
            }
        } catch (SQLException e) {
            System.out.println("Impossibile chiudere la connessione");
        }
    }
    public final void closeTransaction(final Connection connection) {
        try {
            if (connection != null && !connection.isClosed()) {
                connection.commit();
                connection.setAutoCommit(true);
            }
        }
    }
}
```

```

        } catch (SQLException e) {
            System.out.println("Impossibile terminare la transazione");
        }
    }

    /**
     * Se la connessione non è stata gestita con auto commit, occorre utilizzare
     * questo metodo per farlo manualmente
     */
    public final void commitConnection(final Connection connection) throws
SQLException {
    try {
        if (connection != null && !connection.isClosed()) {
            connection.commit();
        }
    } finally {
        closeConnection(connection);
    }
}

    /**
     * Se le modifiche apportate al db non superano una determinata condizione ed
     * occorre annullare tutte le modifiche effettuate
     * dall'ultimo commit, occorre chiamare questa funzione per effettuare il rollback
     * delle operazioni
     */
    public final void rollbackConnection(final Connection connection) {
    try {
        if (connection != null && !connection.isClosed()) {
            connection.rollback();
        }
    } catch (SQLException e) {
        System.out.println("Impossibile effettuare il rollback della
transazione");
    }
}
}

```

Interfaccia che gestisce i contratti

```

package it.business.service;

import java.util.List;

import it.business.dto.ContractDTO;

    /**
     * @author Simone Lungarella
     */

public interface IContractSRV {

    ContractDTO findByContractNumber(int contractNumber);

    List<ContractDTO> findByRegistrar(String registrar);

    List<ContractDTO> findByRegistrant(String registrant);

    ContractDTO findByDomainName(String domainName);
}

```

```

List<ContractDTO> findAll();

void addContract (ContractDTO contract);

void removeContract (int contractNumber);

}

ContractSRV gestisce I contratti e implementa l'interfaccia IContractSRV

package it.business.service;

import java.sql.Connection;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import org.springframework.stereotype.Service;

import it.business.dao.IContractDAO;
import it.business.dto.ContractDTO;

/**
 * @author Simone Lungarella
 */
@Service
@Component
public class ContractSRV extends AbstractService implements IContractSRV{

    private static final long serialVersionUID = 7586912192025747113L;

    @Autowired
    private IContractDAO contractDAO;

    @Override
    public ContractDTO findByContractNumber(int contractNumber) {
        Connection connection = null;
        ContractDTO contract = new ContractDTO();
        try {
            connection = setupConnection(getDataSource().getConnection(),
                false);
            contract = contractDAO.findByContractNumber(connection,
                contractNumber);
        } catch (Exception e) {
            System.out.println("Errore riscontrato durante il setup della
                connessione");
            e.printStackTrace();
        } finally {
            closeConnection(connection);
        }
        return contract;
    }

    @Override
    public List<ContractDTO> findByRegistrar(String registrar) {
        Connection connection = null;
        List<ContractDTO> contracts = new ArrayList<>();
        try {

```

```

        connection = setupConnection(getDataSource().getConnection(),
            false);
        contracts = contractDAO.findByRegistrar(connection, registrar);
    } catch (Exception e) {
        System.out.println("Errore riscontrato durante il setup della
            connessione");
        e.printStackTrace();
    } finally {
        closeConnection(connection);
    }
    return contracts;
}

@Override
public List<ContractDTO> findByRegistrant(String registrant) {
    Connection connection = null;
    List<ContractDTO> contracts = new ArrayList<>();
    try {
        connection = setupConnection(getDataSource().getConnection(),
            false);
        contracts = contractDAO.findByRegistrant(connection, registrant);
    } catch (Exception e) {
        System.out.println("Errore riscontrato durante il setup della
            connessione");
        e.printStackTrace();
    } finally {
        closeConnection(connection);
    }
    return contracts;
}

@Override
public ContractDTO findByDomainName(String domainName) {
    Connection connection = null;
    ContractDTO contract = null;
    try {
        connection = setupConnection(getDataSource().getConnection(),
            false);
        contract = contractDAO.findByDomainName(connection, domainName);
    } catch (Exception e) {
        System.out.println("Errore riscontrato durante il setup della
            connessione");
        e.printStackTrace();
    } finally {
        closeConnection(connection);
    }
    return contract;
}

@Override
public void addContract(ContractDTO contract) {
    Connection connection = null;
    try {
        connection = setupConnection(getDataSource().getConnection(),
            false);
        contractDAO.addContract(connection, contract);
    } catch (SQLException e) {
        System.out.println("Errore riscontrato durante il setup della
            connessione");
        e.printStackTrace();
    }
}

```

```

        } finally {
            closeConnection(connection);
        }
    }

@Override
public void removeContract(int contractNumber) {
    Connection connection = null;
    try {
        connection = setupConnection(getDataSource().getConnection(),
            false);
        contractDAO.removeContractByNumber(connection, contractNumber);
    } catch (Exception e) {
        System.out.println("Errore riscontrato durante il setup della
            connessione");
        e.printStackTrace();
    } finally {
        closeConnection(connection);
    }
}

@Override
public List<ContractDTO> findAll() {
    Connection connection = null;
    List<ContractDTO> contracts = new ArrayList<>();
    try {
        connection = setupConnection(getDataSource().getConnection(),
            false);
        contracts = contractDAO.findAll(connection);
    } catch (Exception e) {
        System.out.println("Errore riscontrato durante il setup della
            connessione");
        e.printStackTrace();
    } finally {
        closeConnection(connection);
    }
    return contracts;
}
}

```

Interfaccia che gestisce i domini

```

package it.business.service;

import java.util.List;

import it.business.dto.DomainDTO;

< /**
 * @author Simone Lungarella
 * */

```

```

public interface IDomainSRV{
    DomainDTO findByDomainName(String domainName);
    List<DomainDTO> findByRegistrant(String registrant);
}

```

```

        List<DomainDTO> findBySecurity(boolean dnssec);

        List<DomainDTO> findAll();

        void addDomain(DomainDTO domainToSave);

        void removeDomain(String domainName);

    }

```

DomainSRV gestisce i domini e implementa l'interfaccia *IDomainSRV*

```

package it.business.service;

import java.sql.Connection;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import org.springframework.stereotype.Service;

import it.business.dao.IDomainDAO;
import it.business.dto.DomainDTO;

/**
 * @author Simone Lungarella
 */
@Service
@Component
public class DomainSRV extends AbstractService implements IDomainSRV{

    private static final long serialVersionUID = -6139923363203091832L;

    @Autowired
    private IDomainDAO domainDAO;

    @Override
    public DomainDTO findByDomainName(String domainName) {
        Connection connection = null;
        DomainDTO domain = null;
        try {
            connection = setupConnection(getDataSource().getConnection(),
                false);
            domain = domainDAO.findByDomainName(connection, domainName);
        } catch (SQLException e) {
            System.out.println("Errore riscontrato durante il setup della
                connessione");
            e.printStackTrace();
        } finally {
            closeConnection(connection);
        }
        return domain;
    }

    @Override
    public List<DomainDTO> findByRegistrant(String registrant) {
        Connection connection = null;
        List<DomainDTO> domains = new ArrayList<>();

```

```

        try {
            connection = setupConnection(getDataSource().getConnection(),
                false);
            domains = domainDAO.findByRegistrant(connection, registrant);
        } catch (SQLException e) {
            System.out.println("Errore riscontrato durante il setup della
                connessione");
            e.printStackTrace();
        } finally {
            closeConnection(connection);
        }
        return domains;
    }

    @Override
    public List<DomainDTO> findBySecurity(boolean dnssec) {
        Connection connection = null;
        List<DomainDTO> domains = new ArrayList<>();
        try {
            connection = setupConnection(getDataSource().getConnection(),
                false);
            domains = domainDAO.findBySecurity(connection, dnssec);
        } catch (SQLException e) {
            System.out.println("Errore riscontrato durante il setup della
                connessione");
            e.printStackTrace();
        } finally {
            closeConnection(connection);
        }
        return domains;
    }

    @Override
    public List<DomainDTO> findAll() {
        Connection connection = null;
        List<DomainDTO> domains = new ArrayList<>();
        try {
            connection = setupConnection(getDataSource().getConnection(),
                false);
            domains = domainDAO.findAll(connection);
        } catch (SQLException e) {
            System.out.println("Errore riscontrato durante il setup della
                connessione");
            e.printStackTrace();
        } finally {
            closeConnection(connection);
        }
        return domains;
    }

    @Override
    public void addDomain(DomainDTO domainToSave) {
        Connection connection = null;
        try {
            connection = setupConnection(getDataSource().getConnection(),
                false);
            domainDAO.addDomain(connection, domainToSave);
        } catch (SQLException e) {

```

```

        System.out.println("Errore riscontrato durante il setup della
                           connessione");
        e.printStackTrace();
    } finally {
        closeConnection(connection);
    }
}

@Override
public void removeDomain(String domainName) {
    Connection connection = null;
    try {
        connection = setupConnection(getDataSource().getConnection(),
                                      false);
        domainDAO.removeByDomainName(connection, domainName);
    } catch (SQLException e) {
        System.out.println("Errore riscontrato durante il setup della
                           connessione");
        e.printStackTrace();
    } finally {
        closeConnection(connection);
    }
}
}

```

Interfaccia che getisce i contatti, viene implementata da ContactSRV

```

package it.business.service;

import java.util.List;

import it.business.dto.ContactDTO;
import it.business.enums.ContactTypeEnum;

/**
 * @author Simone Lungarella
 */
public interface IContactSRV {

    ContactDTO findById(String id);

    List<ContactDTO> findByFirstName(String firstName);

    List<ContactDTO> findByLastName(String lastName);

    List<ContactDTO> findByContactType(ContactTypeEnum contactType);

    List<ContactDTO> findAll();

    void addContact(ContactDTO contactToSave);

    void removeContact(String id);
}

```

ContactSRV gestisce le operazioni sui contatti e predisponi tutti i service utili al recupero e al salvataggio delle informazioni sui contatti.

```
package it.business.service;

import java.sql.Connection;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import org.springframework.stereotype.Service;

import it.business.dao.IContactDAO;
import it.business.dto.ContactDTO;
import it.business.enums.ContactTypeEnum;

/**
 * @author Simone Lungarella
 * Questo service permette di gestire tutte le operazioni sui contatti
 */

@Service
@Component
public class ContactSRV extends AbstractService implements IContactSRV {

    private static final long serialVersionUID = -1374170442695067665L;

    @Autowired
    private IContactDAO contactDAO;

    @Override
    public void addContact(ContactDTO contactToSave) {
        Connection connection = null;
        try {
            connection = setupConnection(getDataSource().getConnection(),
                false);
            contactDAO.addContact(connection, contactToSave);
        } catch (SQLException e) {
            System.out.println("Errore riscontrato durante il setup della
                connessione");
            e.printStackTrace();
        } finally {
            closeConnection(connection);
        }
    }

    @Override
    public List<ContactDTO> findByFirstName(String firstName) {
        Connection connection = null;
        List<ContactDTO> contacts = new ArrayList<>();
        try {
            connection = setupConnection(getDataSource().getConnection(),
                false);
            contacts = contactDAO.findByFirstName(connection, firstName);
        } catch (Exception e) {
            System.out.println("Errore riscontrato durante il setup della
                connessione");
            e.printStackTrace();
        }
    }
}
```

```

        } finally {
            closeConnection(connection);
        }

        return contacts;
    }

    @Override
    public List<ContactDTO> findByLastName(String lastName) {
        Connection connection = null;
        List<ContactDTO> contacts = new ArrayList<>();
        try {
            connection = setupConnection(getDataSource().getConnection(),
                false);
            contacts = contactDAO.findByLastName(connection, lastName);
        } catch (Exception e) {
            System.out.println("Errore riscontrato durante il setup della
                connessione");
            e.printStackTrace();
        } finally {
            closeConnection(connection);
        }
        return contacts;
    }

    @Override
    public List<ContactDTO> findByContactType(ContactTypeEnum contactType) {
        Connection connection = null;
        List<ContactDTO> contacts = new ArrayList<>();
        try {
            connection = setupConnection(getDataSource().getConnection(),
                false);
            contacts = contactDAO.findByContactType(connection, contactType);
        } catch (Exception e) {
            System.out.println("Errore riscontrato durante il setup della
                connessione");
            e.printStackTrace();
        } finally {
            closeConnection(connection);
        }
        return contacts;
    }

    @Override
    public ContactDTO findById(String id) {
        Connection connection = null;
        ContactDTO contact = new ContactDTO();
        try {
            connection = setupConnection(getDataSource().getConnection(),
                false);
            contact = contactDAO.findById(connection, id);
        } catch (Exception e) {
            System.out.println("Errore riscontrato durante il setup della
                connessione");
            e.printStackTrace();
        } finally {
            closeConnection(connection);
        }
        return contact;
    }
}

```

```

@Override
public List<ContactDTO> findAll() {
    Connection connection = null;
    List<ContactDTO> contacts = new ArrayList<>();
    try {
        connection = setupConnection(getDataSource().getConnection(),
            false);
        contacts = contactDAO.findAll(connection);
    } catch (Exception e) {
        System.out.println("Errore riscontrato durante il setup della
            connessione");
        e.printStackTrace();
    } finally {
        closeConnection(connection);
    }
    return contacts;
}
@Override
public void removeContact(String id) {
    Connection connection = null;
    try {
        connection = setupConnection(getDataSource().getConnection(),
            false);
        contactDAO.removeById(connection, id);
    } catch (Exception e) {
        System.out.println("Errore riscontrato durante il setup della
            connessione");
        e.printStackTrace();
    } finally {
        closeConnection(connection);
    }
}
}

```

5.7 SERVIZI DI RECUPERO COMMAND E RESPONSE

La classe *IXmlAttributeGeneratorSRV* è l'interfaccia che va a definire le funzionalità necessarie per il recupero di tutte le request, viene riportata solo in parte:

```

package it.business.service.messaging;

import java.util.List;

import it.business.dto.DomainDTO;

/**
 * @author Simone Lungarella
 */

public interface IXmlAttributeGeneratorSRV extends IXMLGeneratorSRV{

    String getCreateDomainXmlRequest(DomainDTO domain, List<String> contactIds);

    String getCreateDomainXmlRequest(DomainDTO domain, String... contactIds);

}

```

L'interfaccia appena descritta estende un'interfaccia più generica che definisce le caratteristiche di una classe generator, questa interfaccia viene estesa anche da IXmlResponseGeneratorSRV.

```
package it.business.service.messaging;

/**
 * @author Simone Lungarella
 */

public interface IXMLGeneratorSRV {
```

```
}
```

La classe concreta che va ad implementare i metodi per il recupero delle request è *XmlRequestGeneratorSRV*:

```
package it.business.service.messaging;

import java.util.ArrayList;
import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import org.springframework.stereotype.Service;

import it.business.dto.ContactDTO;
import it.business.dto.DomainDTO;
import it.business.service.AbstractService;
import it.business.service.IContactSRV;
import it.business.service.messaging.utils.RequestBuilder;

/**
 * @author Simone Lungarella
 */

@Service
@Component
public class XmlRequestGeneratorSRV extends AbstractService implements
IXMLGeneratorSRV{

    private static final long serialVersionUID = 8931171031798205976L;

    @Autowired
    private IContactSRV contactSRV;

    @Override
    public String getCreateDomainXmlRequest(DomainDTO domain, List<String>
contactIds) {
        String xml = "";
        List<ContactDTO> contacts = new ArrayList<>();

        try {
            for (String s : contactIds) {
                contacts.add(contactSRV.findById(s));
            }

            RequestBuilder reqBuilder = new RequestBuilder();
            xml = reqBuilder.createDomain(domain, contacts);
        } catch (Exception e) {
```

```

        System.out.print("Errore riscontrato nel service XmlGeneratorSRV:
    ");
    e.printStackTrace();
}

return xml;
}

@Override
public String getCreateDomainXmlRequest(DomainDTO domain, String... contactIds) {
    List<String> ids = new ArrayList<>();
    for(String id : contactIds) {
        ids.add(id);
    }

    return getCreateDomainXmlRequest(domain, ids);
}

}

```

5.8 CLASSI BUILDER PER REQUEST E RESPONSE

La classe *MessageFactory* è l'interfaccia che definisce i metodi necessari alla creazione delle request e delle response.

```

package it.business.service.messaging.utils;

import org.w3c.dom.Document;
import org.w3c.dom.Element;

/**
 * @author Simone Lungarella
 */

public interface MessageFactory {

    /**
     * Genera l'elemento root per una qualsiasi richiesta epp
     * @param document
     */
    Element buildRootElement(Document document);

    /**
     * Genera l'elemento createDomain, elemento standard di ogni request epp del
     Registro .it
     * @param document
     */
    Element buildCreateDomainElement(Document document);

    /**
     * Genera un elemento generico della request
     * @param document
     * @param entity    Nome del tag dell'elemento xml da creare
     * @param property Affiancato al nome (e.g entity:property), consente di
     costruire un tag di tipo property.
     *
     * Se property è una stringa vuota allora viene
     creato un tag di default
     * @param value      Il valore che acquisisce il determinato tag in creazione
     */
}

```

```

        Element buildGenericElementWithValue(Document document, String entity, String
property, String value);

    /**
     * Genera il tag per gestire le operazioni con estensione di sicurezza. Si occupa
di definire tutte le caratteristiche relative alla sicurezza.
    *
     * L'estensione di sicurezza prevede il seguente formato
     * <extension>
     *   <secDNS:create xmlns:secDNS="urn:ietf:params:xml:ns:secDNS-1.1">
     *     <secDNS:dsData>
     *       <secDNS:keyTag>12345</secDNS:keyTag>
     *       <secDNS:alg>3</secDNS:alg>
     *       <secDNS:digestType>1</secDNS:digestType>
     *
     <secDNS:digest>4347d0f8ba661234a8eadc005e2e1d1b646c9682</secDNS:digest>
     *       </secDNS:dsData>
     *     </secDNS:create>
     *   </extension>
     *
     * @param document
     */
    Element buildDnsSecExtension(Document document);
}

```

RequestMessageFactory è una classe astratta che si occupa di implementare tutto ciò che è essenziale alla creazione di una generica request, consente di creare gli elementi e gli attributi dei file xml che andranno a definire i command.

```

package it.business.service.messaging.utils;

import java.util.List;

import org.apache.commons.lang.StringEscapeUtils;
import org.w3c.dom.Attr;
import org.w3c.dom.Document;
import org.w3c.dom.Element;

import it.business.dto.ContactDTO;
import it.business.dto.DomainDTO;
import it.business.utils.GenericUtils;

/**
 * @author Simone Lungarella
 */

public abstract class RequestMessageFactory implements MessageFactory{
    /**
     * Le stringhe utilizzate negli attributi degli elementi principali delle request
     */
    private final String xmlns="urn:ietf:params:xml:ns:epp-1.0";
    private final String xmlns_xsi="http://www.w3.org/2001/XMLSchema-instance";
    private final String
xsi_schema_location="xsi:schemaLocation=urn:ietf:params:xml:ns:epp-1.0 epp-1.0.xsd";

    private final String xmlns_domain="urn:ietf:params:xml:ns:domain-1.0";
    private final String xmlns_secDNS="urn:ietf:params:xml:ns:secDNS-1.1";
//    private final String xmlns_extsecDNS="http://www.nic.it/ITNIC-EPP/extsecDNS-1.0";

```

```

abstract String createDomain(DomainDTO domain, List<ContactDTO> contacts);

@Override
public Element buildRootElement(Document document) {
    Element root = document.createElement("epp");
    Attr attr = document.createAttribute("xmlns");
    attr.setValue(StringEscapeUtils.escapeXml(xmlns));
    root.setAttributeNode(attr);
    Attr attr2 = document.createAttribute("xmlns:xsi");
    attr2.setValue(StringEscapeUtils.escapeXml(xmlns_xsi));
    root.setAttributeNode(attr2);
    Attr attr3 = document.createAttribute("xsi:schemaLocation");
    attr3.setValue(StringEscapeUtils.escapeXml(xsi_schema_location));
    root.setAttributeNode(attr3);
    return root;
}

@Override
public Element buildCreateDomainElement(Document doc) {
    Element domainElement = doc.createElement("domain:create");

    Attr attr = doc.createAttribute("xmlns:domain");
    attr.setValue(StringEscapeUtils.escapeXml(xmlns_domain));
    domainElement.setAttributeNode(attr);
    Attr attr2 = doc.createAttribute("xsi:schemaLocation");
    attr2.setValue(StringEscapeUtils.escapeXml(xsi_schema_location));
    domainElement.setAttributeNode(attr2);

    return domainElement;
}

@Override
public Element buildGenericElementWithValue(Document doc, String entity, String
property, String value) {
    // Se property è una stringa vuota allora il tag non sarà costituito da
    // due parole ma solo dall'entità. Questo permette di avere un solo metodo
    // per gestire tutti i tipi generici di nodi
    Element genericElement = doc.createElement(property.isEmpty() ? entity :
(entity+":"+property));
    genericElement.appendChild(doc.createTextNode(value));

    return genericElement;
}

@Override
public Element buildDnsSecExtension(Document doc) {
    Element extension = doc.createElement("extension");
    Element secDNS_create = doc.createElement("secDNS:create");
    Attr attr = doc.createAttribute("xmlns:secDNS");
    attr.setValue(StringEscapeUtils.escapeXml(xmlns_secDNS));
    secDNS_create.setAttributeNode(attr);
    extension.appendChild(secDNS_create);
    Element secDNS_dsData = doc.createElement("secDNS:dsData");
    extension.appendChild(secDNS_dsData);
    Element keyTag = buildGenericElementWithValue(doc, "secDNS", "keyTag",
GenericUtils.randomAlphaNumeric(5));
    secDNS_dsData.appendChild(keyTag);
    Element alg = buildGenericElementWithValue(doc, "secDNS", "alg",
String.valueOf((Math.random()*10/3)+1));
    secDNS_dsData.appendChild(alg);
}

```

```

        Element digestType = buildGenericElementWithValue(doc, "secDNS",
"digestType", String.valueOf((Math.random()*10/3)+1));
        secDNS_dsData.appendChild(digestType);
        Element digest = buildGenericElementWithValue(doc, "secDNS", "digest",
GenericUtils.getDigest());
        secDNS_dsData.appendChild(digest);

        return extension;
    }

}

```

La classe *RequestBuilder* è la classe concreta che, facendo uso di tutti i metodi già implementati nella classe astratta, implementa le funzionalità previste dall'interfaccia.

```

package it.business.service.messaging.utils;

import java.io.StringWriter;
import java.util.ArrayList;
import java.util.List;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.OutputKeys;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerConfigurationException;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;

import org.w3c.dom.Document;
import org.w3c.dom.Element;

import it.business.dto.ContactDTO;
import it.business.dto.DomainDTO;
import it.business.enums.ContactTypeEnum;
import it.business.utils.GenericUtils;

/**
 * @author Simone Lungarella
 */
public class RequestBuilder extends RequestMessageFactory{

    /*
     * Il formato della request per la creazione di un dominio è il seguente
     *
     *      <?xml version="1.0" encoding="UTF-8" standalone="no"?>
     *      <epp xmlns="urn:ietf:params:xml:ns:epp-1.0"
     *           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     *           xsi:schemaLocation="urn:ietf:params:xml:ns:epp-1.0 epp-1.0.xsd">
     *      <command>
     *          <create>
     *              <domain:create
     *                  xmlns:domain="urn:ietf:params:xml:ns:domain-1.0"
     */
}

```

```

        *
        xsi:schemaLocation="urn:ietf:params:xml:ns:domain-1.0
domain-1.0.xsd">
        *
        <domain:name>esempio.it</domain:name>
        *
        <domain:period unit="y">1</domain:period>
        *
        <domain:ns>
            <domain:hostAttr>
        *
        <domain:hostName>ns3174.dns.dyn.com</domain:hostName>
        *
            </domain:hostAttr>
        *
            <domain:hostAttr>
        *
        <domain:hostName>ns3174.dns.dyn.com</domain:hostName>
        *
            </domain:hostAttr>
        *
            </domain:ns>
        *
            <domain:registrant>mm001</domain:registrant>
        *
            <domain:contact type="admin">mm001</domain:contact>
        *
            <domain:contact type="tech">mb001</domain:contact>
        *
            <domain:authInfo>
                <domain:pw>22fooBAR</domain:pw>
            </domain:authInfo>
            </domain:create>
        *
            </create>
        *
            <cLTRID>ABC-12345</cLTRID>
        *
    </command>
    </epp>
*
*/

```

@Override

```

public String createDomain(DomainDTO domain, List<ContactDTO> contacts) {
    String request = "";
    DocumentBuilderFactory documentFactory =
DocumentBuilderFactory.newInstance();

    // Inizializzazione dei contatti utili alla creazione della request
    ContactDTO registrant = new ContactDTO();
    List<ContactDTO> otherContacts = new ArrayList<>();
    for(ContactDTO c : contacts) {
        if (c.getContactType().equals(ContactTypeEnum.REGISTRANT)){
            registrant = new ContactDTO(c);
        } else {
            otherContacts.add(c);
        }
    }

    DocumentBuilder documentBuilder;

    try {
        documentBuilder = documentFactory.newDocumentBuilder();
        Document document = documentBuilder.newDocument();
        Element root = buildRootElement(document);
        document.appendChild(root);
        Element command = document.createElement("command");
        root.appendChild(command);
        Element create = document.createElement("create");
        command.appendChild(create);
        Element cmdCreate = buildCreateDomainElement(document);
        create.appendChild(cmdCreate); // Fin qui tutto ok
        Element name = buildGenericElementWithValue(document, "domain",
"name", domain.getDomainName());

```

```

        cmdCreate.appendChild(name);
        Element period = buildGenericElementWithValue(document, "domain",
"period", String.valueOf((Math.random()*10)+1));
        cmdCreate.appendChild(period);
        Element domain_ns = document.createElement("domain:ns");
        cmdCreate.appendChild(domain_ns);
        Element host_attr = document.createElement("domain:hostAttr");
        domain_ns.appendChild(host_attr);

        // Genero alcuni hostname per la gestione del dominio
        int randomBound = (int) (Math.round(Math.random()*10/3 + 1));
        for(int i = 1; i <= randomBound; i++) {
            domain_ns.appendChild(buildGenericElementWithValue(document,
"domain", "hostname", "ns" + String.valueOf((Math.random()*1000)+1) + ".dns.dyn.com"));
        }
        Element hostname = buildGenericElementWithValue(document, "domain",
"registrant", registrant.getContactId());
        domain_ns.appendChild(hostname);

        // Popolo i tag contenenti le informazioni sugli admin e sui tech
        del dominio
        for(ContactDTO c : otherContacts) {
            domain_ns.appendChild(buildGenericElementWithValue(document,
"domain", c.getContactType().toString().toLowerCase(), c.getContactId()));
        }
        Element authInfo = document.createElement("domain:authInfo");
        cmdCreate.appendChild(authInfo);
        Element pw = buildGenericElementWithValue(document, "domain", "pw",
GenericUtils.randomAlphaNumeric(8));
        authInfo.appendChild(pw);

        // Se il dominio prevede l'estensione di sicurezza, la request deve
        prevedere un tag di estensione
        if(domain.isDnssec()) {
            Element dnssecExtension = buildDnsSecExtension(document);
            command.appendChild(dnssecExtension);
        }

        // Trasformazione in stringa dell'XML generato
        TransformerFactory tf = TransformerFactory.newInstance();
        Transformer transformer = tf.newTransformer();
        transformer.setOutputProperty(OutputKeys.OMIT_XML_DECLARATION,
"yes");
        StringWriter writer = new StringWriter();
        transformer.transform(new DOMSource(document), new
StreamResult(writer));
        request = writer.getBuffer().toString().replaceAll(">", ">\n");

    } catch (ParserConfigurationException e) {
        e.printStackTrace();
    } catch (TransformerConfigurationException e) {
        e.printStackTrace();
    } catch (TransformerException e) {
        e.printStackTrace();
    }

    return request;
}
}

```

5.9 DATA ACCESS OBJECTS

I SRV fanno uso di svariate classi, queste classi si occupano di creare le query e gestire i resultSet in modo da recuperare le informazioni dal database e trasformarle in DTO utilizzabili dal business. La logica con cui sono stati costruiti i DAO segue quella dei SRV pertanto esisterà un AbstractDAO che gestisce gli statement e i resultSet, occupandosi, in particolar modo di liberare le risorse.

```
package it.business.dao;

import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

/**
 * @author Simone Lungarella
 * Questo DAO gestisce tutte le operazioni comuni a tutti i DAO
 */

public abstract class AbstractDAO {

    public final void closeStatement(final Statement ps) {
        closeStatement(ps, null);
    }

    public final void closeStatement(final Statement ps, final ResultSet rs) {
        if (rs != null) {
            try {
                if (!rs.isClosed()) {
                    rs.close();
                }
            } catch (SQLException e) {
                System.out.println("Errore riscontrato durante la chiusura del resultSet");
            }
        }
        if (ps != null) {
            try {
                if (!ps.isClosed()) {
                    ps.close();
                }
            } catch (SQLException e) {
                System.out.println("Errore riscontrato durante la chiusura dello statement");
            }
        }
    }

    public void closeResultSet(final ResultSet rs) {
        if (rs != null) {
            try {
                rs.close();
            } catch (SQLException e) {
                System.out.println("Errore riscontrato durante la chiusura del resultSet");
            }
        }
    }
}
```

Di seguito si riporta *ContactDAO*, la classe che gestisce l'implementazione dell'interfaccia che definisce le operazioni sui contatti.

```
package it.business.dao.imp;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

import org.springframework.stereotype.Repository;
import it.business.dao.AbstractDAO;
import it.business.dao.IContactDAO;
import it.business.dto.ContactDTO;
import it.business.enums.ContactTypeEnum;

/**
 * @author Simone Lungarella
 *
 */
@Repository
public class ContactDAO extends AbstractDAO implements IContactDAO {

    private static final long serialVersionUID = -9110222349730022857L;

    @Override
    public List<ContactDTO> findByFirstName(Connection connection, String firstName)
    {
        List<ContactDTO> contacts = new ArrayList<>();
        PreparedStatement ps = null;
        ResultSet rs = null;
        ContactDTO contact = new ContactDTO();
        try {
            String query = "SELECT * FROM contacts WHERE firstName = ?";
            ps = connection.prepareStatement(query);
            ps.setString(1, firstName);
            rs = ps.executeQuery();
            while (rs.next()) {
                contact.setContactId(rs.getString("contactId"));

                contact.setContactType(ContactTypeEnum.valueOf(rs.getString("contactType")));
                contact.setFirstName(rs.getString("firstName"));
                contact.setLastName(rs.getString("lastName"));
                contacts.add(contact);
            }
        } catch (SQLException e) {
            System.out.println("Errore riscontrato nell'esecuzione della query nel metodo findByFirstName");
            e.printStackTrace();
        } finally {
            closeStatement(ps, rs);
        }
        return contacts;
    }

    @Override
```

```

public List<ContactDTO> findByLastName(Connection connection, String lastName) {
    List<ContactDTO> contacts = new ArrayList<>();
    PreparedStatement ps = null;
    ResultSet rs = null;
    ContactDTO contact = new ContactDTO();
    try {
        String query = "SELECT * FROM contacts WHERE lastName = ?";
        ps = connection.prepareStatement(query);
        ps.setString(1, lastName);
        rs = ps.executeQuery();
        while (rs.next()) {
            contact.setContactId(rs.getString("contactId"));

            contact.setContactType(ContactTypeEnum.valueOf(rs.getString("contactType")));
            contact.setFirstName(rs.getString("firstName"));
            contact.setLastName(rs.getString("lastName"));
            contacts.add(contact);
        }
    } catch (SQLException e) {
        System.out.println("Errore riscontrato nell'esecuzione della query
nel metodo findByLastName");
        e.printStackTrace();
    } finally {
        closeStatement(ps, rs);
    }
    return contacts;
}

@Override
public List<ContactDTO> findByContactType(Connection connection, ContactTypeEnum
contactType) {
    List<ContactDTO> contacts = new ArrayList<>();
    PreparedStatement ps = null;
    ResultSet rs = null;
    ContactDTO contact = new ContactDTO();
    try {
        String query = "SELECT * FROM contacts WHERE contactType = ?";
        ps = connection.prepareStatement(query);
        ps.setString(1, contactType.toString());
        rs = ps.executeQuery();
        while (rs.next()) {
            contact.setContactId(rs.getString("contactId"));

            contact.setContactType(ContactTypeEnum.valueOf(rs.getString("contactType")));
            contact.setFirstName(rs.getString("firstName"));
            contact.setLastName(rs.getString("lastName"));
            contacts.add(contact);
        }
    } catch (SQLException e) {
        System.out.println("Errore riscontrato nell'esecuzione della query
nel metodo findByContactType");
        e.printStackTrace();
    } finally {
        closeStatement(ps, rs);
    }
    return contacts;
}

@Override

```

```

public ContactDTO findById(Connection connection, String id) {
    PreparedStatement ps = null;
    ResultSet rs = null;
    ContactDTO contact = new ContactDTO();
    try {
        String query = "SELECT * FROM contacts WHERE contactId = ?";
        ps = connection.prepareStatement(query);
        ps.setString(1, id);
        rs = ps.executeQuery();
        if (rs.next()) {
            contact.setContactId(rs.getString("contactId"));

            contact.setContactType(ContactTypeEnum.valueOf(rs.getString("contactType")));
                contact.setFirstName(rs.getString("firstName"));
                contact.setLastName(rs.getString("lastName"));
        }
    } catch (SQLException e) {
        System.out.println("Errore riscontrato nell'esecuzione della query
            nel metodo findById");
        e.printStackTrace();
    } finally {
        closeStatement(ps, rs);
    }

    return contact;
}

@Override
public List<ContactDTO> findAll(Connection connection){
    List<ContactDTO> contacts = new ArrayList<>();
    PreparedStatement ps = null;
    ResultSet rs = null;
    try {
        String query = "SELECT * From contacts";
        ps = connection.prepareStatement(query);
        rs = ps.executeQuery();
        while (rs.next()) {
            ContactDTO contact = new ContactDTO();
            contact.setContactId(rs.getString("contactId"));
            contact.setContactType(ContactTypeEnum.valueOf(rs.getString("contactType")));
                contact.setFirstName(rs.getString("firstName"));
                contact.setLastName(rs.getString("lastName"));
            contacts.add(contact);
        }
    } catch (SQLException e) {
        System.out.println("Errore riscontrato nell'esecuzione della query
            nel metodo findByContactType");
        e.printStackTrace();
    } finally {
        closeStatement(ps, rs);
    }
    return contacts;
}

@Override
public void addContact(Connection connection, ContactDTO contact) {
    PreparedStatement ps = null;

```

```

int index = 1;
try {
    String query = "INSERT INTO contacts (contactId, firstName,
lastName, contactType) VALUES (?,?,?,?,?)";
    ps = connection.prepareStatement(query);
    ps.setString(index++, contact.getContactId());
    ps.setString(index++, contact.getFirstName());
    ps.setString(index++, contact.getLastName());
    ps.setString(index++, contact.getContactType().toString());
    ps.executeUpdate();
} catch (SQLException e) {
    System.out.println("Errore riscontrato durante l'inserimento del
nuovo utente");
    e.printStackTrace();
} finally {
    closeStatement(ps);
}
}

@Override
public void removeById(Connection connection, String id) {
    PreparedStatement ps = null;

    try {
        String query = "DELETE FROM contacts WHERE contactId = ?";
        ps = connection.prepareStatement(query);
        ps.setString(1, id);
        ps.executeUpdate();
    } catch (Exception e) {
        System.out.println("Errore riscontrato durante l'eliminazione dell'
utente");
        e.printStackTrace();
    } finally {
        closeStatement(ps);
    }
}

@Override
public void update(Connection connection, ContactDTO newContact) {
    PreparedStatement ps = null;
    int index = 1;
    try {
        String query = "UPDATE contacts SET firstName = ?, lastName = ?,
contactType = ? WHERE contactId = ?";
        ps = connection.prepareStatement(query);
        ps.setString(index++, newContact.getFirstName());
        ps.setString(index++, newContact.getLastName());
        ps.setString(index++, newContact.getContactType().toString());
        ps.setString(index, newContact.getContactId());
        ps.executeUpdate();
    } catch (Exception e) {
        System.out.println("Errore durante l'update dell'utente");
    } finally {
        closeStatement(ps);
    }
}
}

```

6. TEST

Per garantire il corretto funzionamento dell'applicativo sono stati effettuati molteplici test in locale utilizzando windows 10. I test effettuati durante l'implementazione dell'applicazione, però, non bastano a garantire il corretto funzionamento su dispositivi generici. Viene utilizzato **BrowserStack** per effettuare molteplici test andando a selezionare combinazioni di dispositivi e sistemi operativi differenti.

The screenshot shows the BrowserStack build interface for a project named 'Regapp_build02'. It displays session details: 4 UNMARKED sessions, last updated at 14:38 UTC 22 Aug 2020, by user Simone Lungarella, with a completed build ID: e4ffc0d9653d67469374dc323f146630764f035d. Below this, the 'SESSIONS OVERVIEW' section shows 4 completed sessions, 0 timed out, and 0 errors. A search bar for sessions is also present.

Figura 11: Build dell'applicativo attraverso BroswerStack

6.1 TEST CONNETTIVITÀ

Predisposto l'ambiente per effettuare i test tramite questo tool di supporto, è stato effettuato dapprima un test di connessione dell'applicativo e successivamente sono stati eseguiti alcuni test atti a testare la stabilità e la responsività dell'applicazione.

The screenshot shows the 'Debug Connectivity' tool. It prompts for an URL, which is entered as 'http://localhost:8080/Regapp/'. A blue 'Debug' button is visible. Below the URL input, the 'Response:' section shows a status of 200 and a test info message: 'Connecting to debugURL - without proxy'. The 'Description:' field shows the URL being accessed. The 'Debug Info:' section displays a JSON object representing the response headers and content, including the status code 200, server Apache-Coyote/1.1, set-cookie JSESSIONID=63341F2BA35EA5760EDA234BAA1ED297; Path=/Regapp/; HttpOnly, content-type text/html; charset=UTF-8, content-length 6126, date Sat, 22 Aug 2020 14:19:26 GMT, and connection close.

Figura 12: Test connessione con risposta positiva

6.2 TEST RESPONSIVITÀ

Per verificare il livello di responsività dell'applicazione vengono effettuati 14 test scegliendo casualmente dispositivi e sistemi operativi. In particolare viene utilizzato il tool che consente di ottenere screenshot direttamente dai dispositivi in modo da avere la possibilità di visualizzare l'app su diverse tecnologie. Di seguito i risultati del test:

The screenshot shows the BrowserStack local testing interface. At the top, there's a URL bar with 'http://localhost:8080/Regapp/' and a 'Generate' button. Below that is a navigation bar with 'Local Testing' selected, 'Learn more', 'Email results', and 'Settings'. A message says 'Select browsers and device combinations (14/25)'. On the right, there's a 'BACK TO DEVICES' link. The main area displays 14 screenshots of the application interface on various devices. A summary at the top says '14 of 14 done.' with a green checkmark. Below the screenshots, each one is labeled with its configuration: Windows XP ie 7, Windows 8.1 firefox 30, Windows 8 ie 10, Windows XP firefox 3.6, Windows 8.1 ie 11, Windows 8 firefox 62, Windows 7 ie 9, Windows 7 ie 8, OS X Yosemite safari 8, OS X Mavericks chrome 36, OS X Yosemite opera 12.15, Google Nexus 5 (Portrait), Samsung Galaxy S7 (Portrait), and Google Nexus 6 (Portrait). There are download icons next to each screenshot.

Figura 11: Overview dei test su 14 dispositivi

L'applicazione su smartphone non presenta notevoli problemi, il funzionamento è garantito anche su dispositivi mobile ma in questo caso gli elementi utilizzati per la costruzione della pagina non sono correttamente renderizzati. A livello funzionale non sono stati individuati problemi, in quanto resta possibile navigare tra le pagine e svolgere le attività come da progettazione.

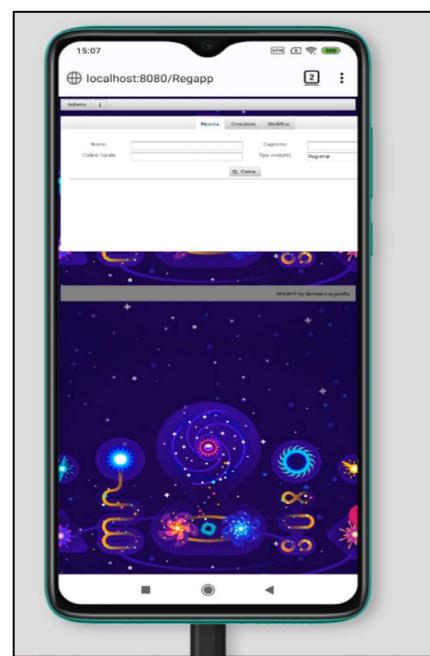


Figura 12: Test failure_01

I risultati ottenuti dimostrano che l'applicazione ha un buon grado di responsività su diverse combinazioni e presenta problemi di renderizzazione quasi solo esclusivamente sulla finestra di visualizzazione dello strato di persistenza. Dato che la finestra è stata renderizzata utilizzando un *iFrame* questo potrebbe essere un problema legato non tanto alla corretta implementazione quanto al metodo di test, dovendo effettuare una connessione al browser partendo da una connessione già simulata.

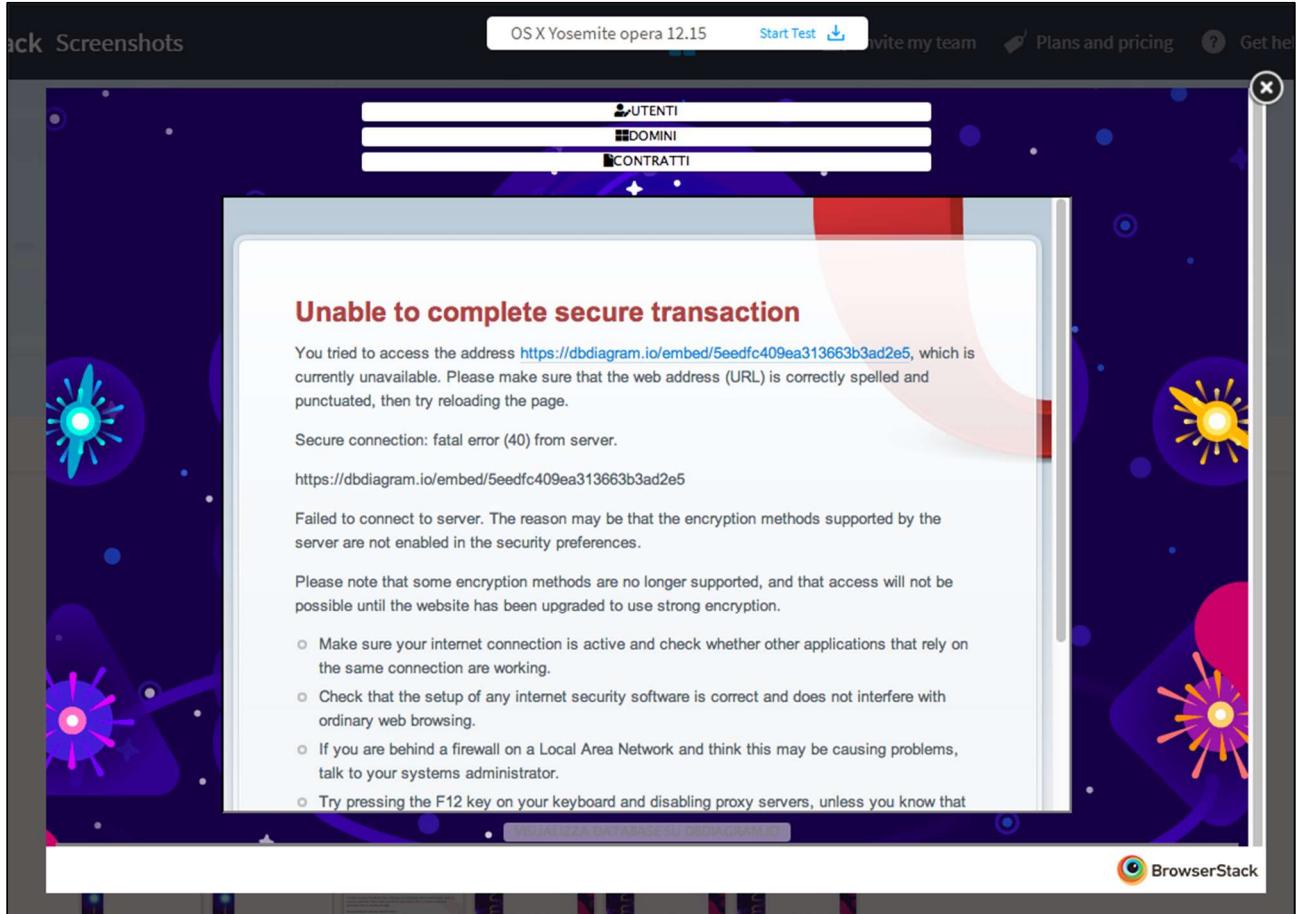


Figura 13: Test failure_02

6.3 TEST FUNZIONALI

Per quanto riguarda le funzionalità implementate, durante il periodo di test dell'applicativo non sono stati riscontrati problemi diversi dalla renderizzazione dell'iFrame, già citato in precedenza. È stato possibile creare nuove entità e generare xml dei command senza problemi di alcun genere. È stato utilizzato un tool di SpeedTest per verificare la velocità di esecuzione sui diversi dispositivi. I test suddetti hanno mostrato notevole velocità di esecuzione ma non sono stati presi in considerazione poiché non è stato possibile utilizzare un database su server esterno, e questo comporta una notevole riduzione di tempo necessario a svolgere le funzionalità in quanto il tempo di comunicazione con il database viene ridotto al minimo.

Per effettuare il test di connessione con il tool di supporto che è stato usato per effettuare i test è stato necessaria la scrittura di una classe di test unitario e l'utilizzo di due ulteriori dipendenze.

L'implementazione della classe **BrowserStackSampleTest**:

```
import java.net.URL;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.remote.DesiredCapabilities;
import org.openqa.selenium.remote.RemoteWebDriver;

public class BrowserStackSampleTest {
    public static final String USERNAME = "simonelungarella1";
    public static final String AUTOMATE_KEY = "Tuhsvga1P7Ksy1mmYUkK";
    public static final String URL = "https://" + USERNAME + ":" + AUTOMATE_KEY + "@hub-
cloud.browserstack.com/wd/hub";

    @Before
    public final void beforeTest() {
        // Non occorre fare niente
    }

    @After
    public final void afterTest() {
        // Non occorre fare niente
    }

    @Test
    public void test() throws Exception {
        DesiredCapabilities caps = new DesiredCapabilities();

        caps.setCapability("os", "Windows");
        caps.setCapability("os_version", "10");
        caps.setCapability("browser", "Chrome");
        caps.setCapability("browser_version", "80");

        caps.setCapability("name", "simonelungarella1's First Test");

        WebDriver driver = new RemoteWebDriver(new URL(URL), caps);
        driver.get("http://www.google.com");
        WebElement element = driver.findElement(By.name("q"));

        element.sendKeys("BrowserStack");
        element.submit();

        System.out.println(driver.getTitle());
        driver.quit();
    }
}
```

Dipendenze aggiunte al file *pom.xml*:

```
<dependencies>
    [...]
    <dependency>
        <groupId>org.seleniumhq.selenium</groupId>
        <artifactId>selenium-java</artifactId>
        <version>3.141.59</version>
    </dependency>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.12</version>
        <scope>test</scope>
    </dependency>
    [...]
</dependencies>
```

7. CONCLUSIONI

Il processo di comunicazione tra Registro e Registrar per la creazione di nuovi domini nel ccTLD .it è un processo complicato che può essere svolto solo utilizzando un client solido e affidabile. L'applicativo realizzato fornisce una dimostrazione di tale processo, evidenziandole gli aspetti principali e fornisce una possibile realizzazione di un client di comunicazione utilizzabile da un Registrar in prima persona. L'applicazione rispetta i vincoli stabiliti in fase di analisi e presenta un'interfaccia utente affidabile e intuitiva, questo consente di godere di un'esperienza di utilizzo molto fluida e piacevole.

7.1 PRESTAZIONI

L'utilizzo di Spring ha permesso, grazie al principio di **Inversion of Control**, di alleggerire il carico dell'applicativo e ridurre al minimo le risorse utilizzate durante il normale funzionamento. I test effettuati hanno dimostrato un buon grado di responsività su svariati dispositivi e alcuni problemi legati all'utilizzo di iFrame. Come detto precedentemente l'applicativo risulta avere tempi di esecuzione molto bassi ma questo è dovuto soprattutto al fatto che il database non comporta quasi alcun rallentamento all'esecuzione poiché non è necessario effettuare nessuna connessione in quanto in esecuzione locale.

7.2 OTTIMIZZAZIONI POSSIBILI

L'applicazione realizzata resterà pubblica su GitHub.com, presenta una guida per agevolare la preparazione del database e il modo di connessione adottato dall'applicativo. Grazie all'utilizzo di Maven e Spring, risulta molto semplice configurare l'applicazione sul proprio dispositivo, questo consente di ripartire dallo stato attuale dell'applicativo e apportarne modifiche e ottimizzazioni, a questo riguardo credo sia opportuno utilizzare un **Database as a Service** in modo da esternalizzare la gestione del database e consentire l'utilizzo dell'applicazione da chiunque senza necessità di riconfigurare lo strato di persistenza. Sarebbe opportuno, inoltre, implementare una pagina di login che consentirebbe l'accesso solo ad uno specifico Registrar, questo consentirebbe, oltre che a migliorare la sicurezza dell'applicazione, a renderne possibile il salvataggio di informazioni sensibili legate all'utente. Un client creato ad hoc per un Registrar, infatti, dovrebbe permettere allo stesso di visualizzare tutte le operazione che comportano un guadagno monetario e tutte quelle che non lo fanno ed eventualmente offrire funzionalità legate alla gestione del proprio saldo. Si ricorda inoltre, che non è stata fornita la possibilità di effettuare tutte le operazioni possibili di un Registrar ma solo quelle principali o di maggior valore informativo, quindi per rendere quest' app completa sarebbe opportuno implementare le rimanenti funzionalità.