# Università degli Studi di Napoli Federico II


Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione

*Classe delle Lauree in Scienze e Tecnologie Informatiche*
Corso di Laurea in Informatica

Elaborato di Laurea

*GAM# - Gestione Accoglienza Minori*

Tutor accademico:
Prof. Anna Corazza

Tutor aziendale:
Ing. Saverio De Vito

Candidato:
Michele Maione (566002580)

Anno Accademico
2017/2018

# Ringraziamenti

Ringrazio:

L’Ing. Saverio De Vito per il supporto alla progettazione del software;

I dirigenti e gli impiegati del Centro “La Tenda” ONLUS per la loro disponibilità.

# Sintesi

Il Centro “La Tenda” ONLUS (NA) nasce nel 1981 per volontà di un giovane sacerdote Don Antonio Vitiello e di un gruppo di volontari impegnati nella ricerca di risposte di accoglienza ad un disagio, allora quasi sconosciuto ma sempre più avvertito, rappresentato da persone e famiglie svantaggiate, in via prioritaria ma non esclusiva a causa di una tossicodipendenza. Da sempre è attiva sul fronte della prevenzione del disagio e della devianza giovanile; negli ultimi anni ha moltiplicato gli interventi a favore dei minori e degli adolescenti, anche immigrati, realizzando un Centro diurno polivalente per le famiglie e i minori, numerosi progetti mirati con le scuole e con altre associazioni sull’intera area metropolitana e provinciale.

L’associazione chiede una soluzione informatica per gestire le interazioni del minore con il centro e con le agenzie educative collaboranti.

Le informazioni raccolte nella fase di analisi provengono dai colloqui effettuati presso il centro, con i dirigenti, alcune psicologhe e due “Informatici senza frontiere”: Saverio De Vito e Paolo Lubrano Lavadera, e me.

Verrà esposto ed analizzato il problema, discusse le scelte progettuali, esposte le metodologie di sviluppo ed illustrati gli strumenti utilizzati per lo sviluppo del gestionale GAM#.

# Indice generale

# I. Requisiti

## 1. Raccolta dei requisiti

A luglio 2015 ci fu il primo e il secondo colloquio su cui si è basata la raccolta dei requisiti, l'analisi e lo sviluppo di questo progetto. Ai colloqui erano presenti il direttore del centro, tre psicologhe, i due "Informatici senza frontiere" Saverio De Vito e Paolo Lubrano Lavadera, ed io. Il cliente chiese una consulenza informatica ad "Informatici senza frontiere" per risolvere il problema di centralizzare i documenti elettronici, la maggior parte file Word, su un unico computer (all'epoca sparsi tra i vari computer della sede). Poiché io avevo a disposizione 4 mesi di lavoro da poter utilizzare, gli ho proposto la creazione di un software gestionale per gestire tutto il centro. Purtroppo ci furono dei requisiti un po' stringenti circa l'infrastruttura e il tempo (rispetto alla dimensione del progetto).

All'inizio avrei voluto proporre un portale web fatto con il web application framework Portofino (Stripes, Hibernate, Liquibase, Groovy, Apache Shiro), ma noleggiare un server costava troppo. Poi proposi una web application ASP.NET, ma non si poteva nemmeno noleggiare uno spazio web. Anzi in realtà non dovevo dare nemmeno per scontato che ci fosse una connessione internet. La maggior parte dei computer aveva Windows XP sp3, alcuni Windows 7, erano tutti senza gruppo di continuità, ed il server era un Windows NT 3.5 Server (del 1995). Proposi un gestionale desktop client-server in .NET 2.0, che poteva essere eseguito su tutti i computer, poiché gli XP avevano tutti il service pack 3. La [s]fortuna volle che a settembre 2015 il server fu attaccato da un Crypto Locker, e fu formattato come Windows 7.

### Richiesta

Il Centro "La Tenda" ONLUS, chiede una soluzione informatica per gestire le interazioni del minore con il centro e con le agenzie educative collaboranti.

### Situazione attuale

Il centro non dispone di nessun software per la gestione delle sue attività. Attualmente vengono salvati sui singoli computer, dei documenti redatti con Microsoft Word, senza una gestione centralizzata dei documenti.

## Obiettivi generali del progetto

Il workflow che il programma dovrà seguire, in linee generali, è quello mostrato nell'illustrazione 1:

- All'arrivo al centro si dovranno memorizzare i dati del minore e dei tutori.
- Il minore verrà aggiunto ad una lista di attesa, verrà generata una "scheda primo contatto" simile a quella fornita dal Comune di Napoli.
- Il software dovrà dare la possibilità di ricercare per parametri nella lista d'attesa.
- Una volta accettato verrà generata la scheda d'iscrizione, e il minore risulterà iscritto.
- Il minore verrà inserito in un gruppo. I gruppi sono divisi per scaglioni di età.
- Minore e tutori, in accordo, sceglieranno dei laboratori e delle materie (le offerta sono stagionali e variano a seconda dello scaglione d'età).
- Il software dovrà memorizzare anche eventuali attività di recupero esterne svolte dal minore.
- Effettuata la scelta verrà generato il documento di "patto educativo" e verrà stampato un calendario per i tutori (calendario stampabile anche durante l'anno)
- Per ogni minore il software dovrà gestire, i laboratori a cui è iscritto, le materie che segue, le attività esterne, i colloqui con i tutori. Dovrà essere sempre possibile stampare sia il calendario sia il modello "PEI"[1].
- Il software dovrà tenere traccia delle presenze dei ragazzi ai laboratori.
- Il software dovrà dare la possibilità di creare un calendario generico delle attività annuali.

1 È il documento nel quale vengono descritti gli interventi integrati ed equilibrati tra di loro, predisposti per l'alunno in situazione di handicap, in un determinato periodo di tempo, ai fini della realizzazione del diritto all'educazione e all'istruzione, di cui ai primi quattro commi dell'art. 12 della legge n. 104 del 1992. (D.P.R. 24/02/1994). Individua gli obiettivi di sviluppo, le attività, le metodologie, le facilitazioni, le risorse umane e materiali coinvolte, i tempi e gli strumenti per la verifica; tiene presenti i progetti didattico-educativi, riabilitativi e di socializzazione individualizzati, nonché le forme di integrazione tra attività scolastiche ed extrascolastiche. Va redatto entro il primo bimestre di scuola, cioè entro il 30 novembre di ogni anno scolastico, si verifica periodicamente.

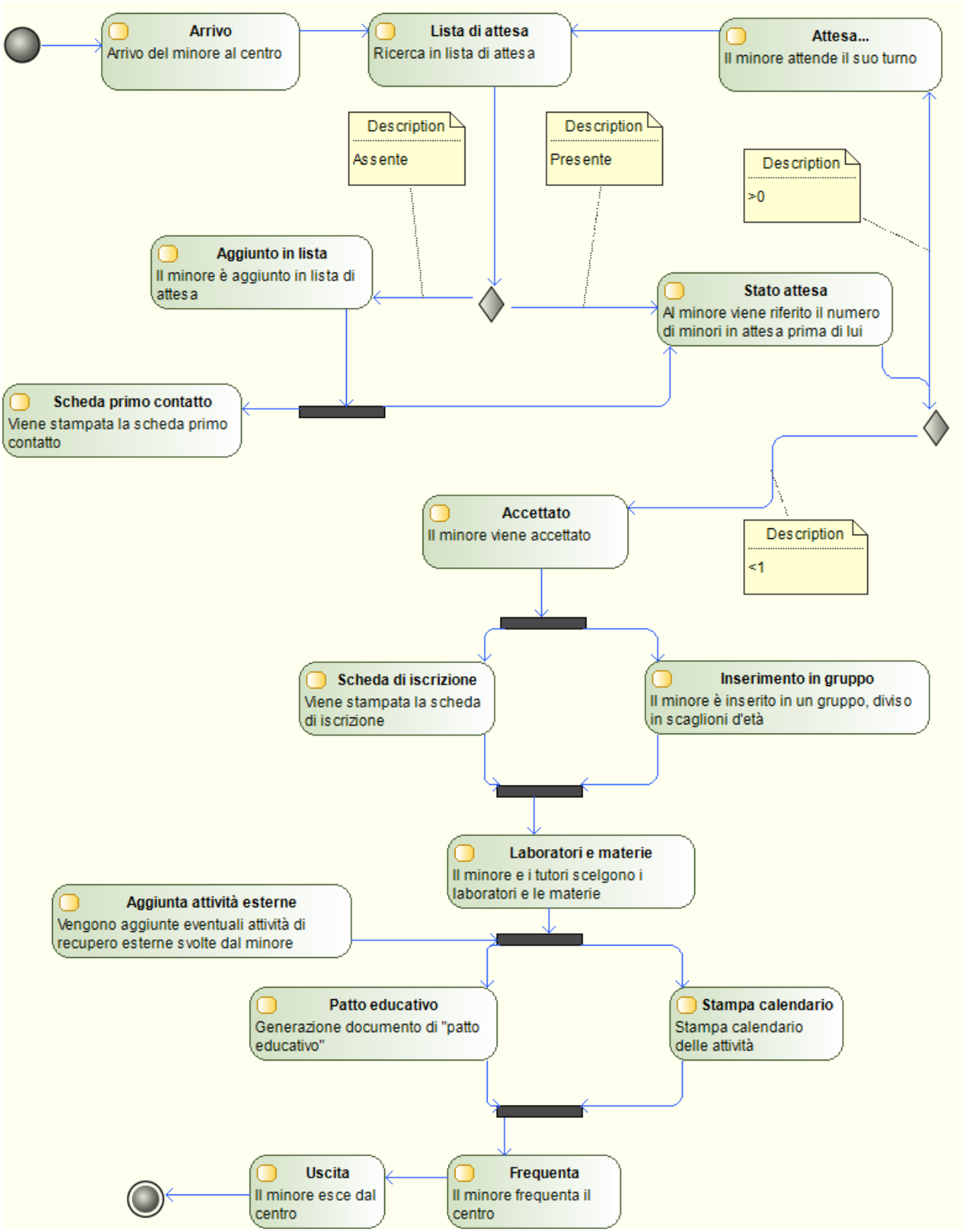


*Illustrazione 1: Workflow - arrivo del minore*

## Utenti

Gli utenti che useranno il software saranno le educatrici, le assistenti e le psicologhe che lavorano al centro.

## Scenari d'uso

Seguono tutti i possibili scenari d'uso del software per i quali sono state sviluppate le funzionalità atte a svolgerli:

1. Creazione laboratori e materie: il personale addetto crea nuovi laboratori e nuove materie, e stampa il calendario generico annuale;
2. Reception, arriva un minore con i suoi tutori per l'iscrizione: il personale addetto alla reception inserisce i dati del minore e dei tutori, salva, e genera il documento "primo contatto";
3. Gestione liste d'attesa: sono disponibili N nuovi posti: il personale addetto alla gestione delle liste d'attesa, dopo aver eseguito alcune ricerche parametriche sulla lista sceglie N ragazzi e chiama i tutori;
4. Gestione liste d'attesa: minore chiamato per scorrimento: il minore con i tutori giunge al centro perché ha ricevuto la telefonata di accettazione, viene stampata la scheda d'iscrizione e risulta iscritto;
5. Il minore e la famiglia sono tenuti a scegliere laboratori e materie: l'addetto all'aiuto per la scelta dei laboratori, dà alcune delucidazioni, e iscrive il minore ad alcuni laboratori ed ad alcune materie. Aggiunge anche eventuali attività di recupero esterne. Stampa il "patto educativo" e il calendario;
6. Gestione presenze ai laboratori: dopo alcune settimane, l'insegnante di un laboratorio, decide di registrare nel programma le presenze dei ragazzi ai laboratori, che ha segnato su carta;
7. Colloquio periodico con i tutori: l'insegnante o lo psicologo, tengono periodicamente dei colloqui, quindi hanno la necessità di inserire nel programma un sunto e le loro valutazioni del colloquio effettuato.

# 2. Requisiti del progetto

## Requisiti di architettura

### Architettura informativa

Il centro dispone di computer con prestazioni piuttosto basse con sistema operativo Microsoft Windows XP sp3 o Microsoft Windows 7.

## Requisiti di comunicazione

### Grafica e multimedialità

Il progetto verrà realizzato per essere funzionale su una risoluzione minima di 800px × 600px.

### Lingua e localizzazione

Il progetto sarà solamente in italiano.

## Requisiti funzionali

### Casi d'uso

- Computer server: su questo computer verrà installato il programma e il database.
- Computer client: su questi computer, tramite un collegamento sul desktop e nel menù start, generato automaticamente al primo avvio di GAM#, verrà eseguito il programma risiedente sul server.

### Base di dati

GAM# non è vincolato a nessun RDBMS poiché rispetta lo standard SQL:2011 e sono presenti nel setup i connettori per Firebird, MySQL, PostgreSQL e Microsoft SQL Server.

La scelta, per motivi hardware, è ricaduta su Firebird[2] (versione Superserver 2.5.4 a 32bit), della Firebird Project, rilasciato sotto licenza IDPL (Initial Developer's Public License). Firebird richiede per funzionare un processore 486DX2 66MHz & 64MB di RAM contro MySQL che ha bisogno di 2 CPU Cores & 2GB di RAM (l'argomento in dettaglio a pagina Errore: sorgente del riferimento non trovata).

2 http://www.firebirdsql.org

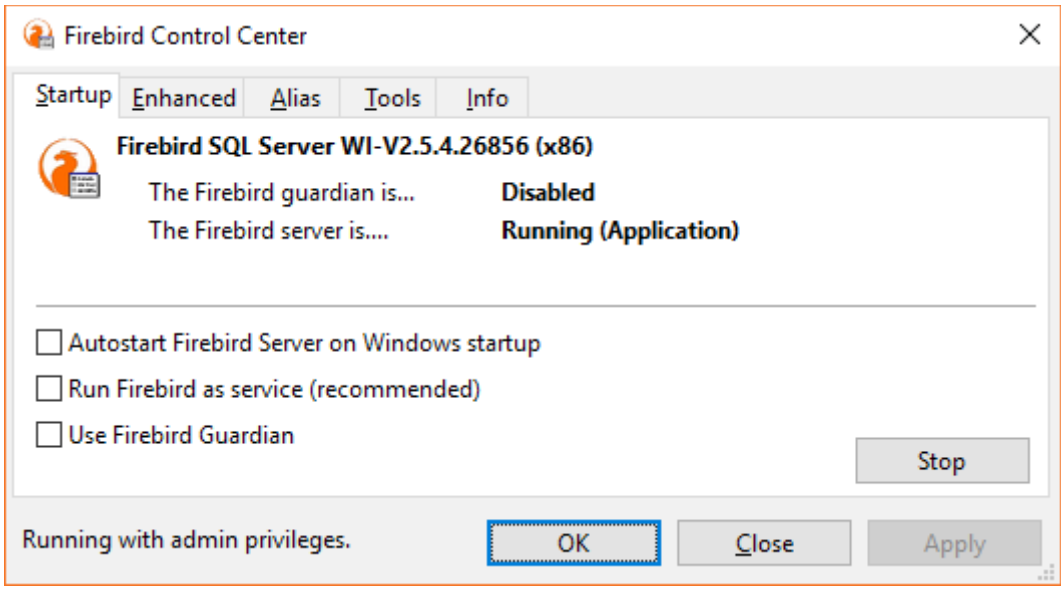


*Illustrazione 2: Impostazioni del server Firebird SQL*

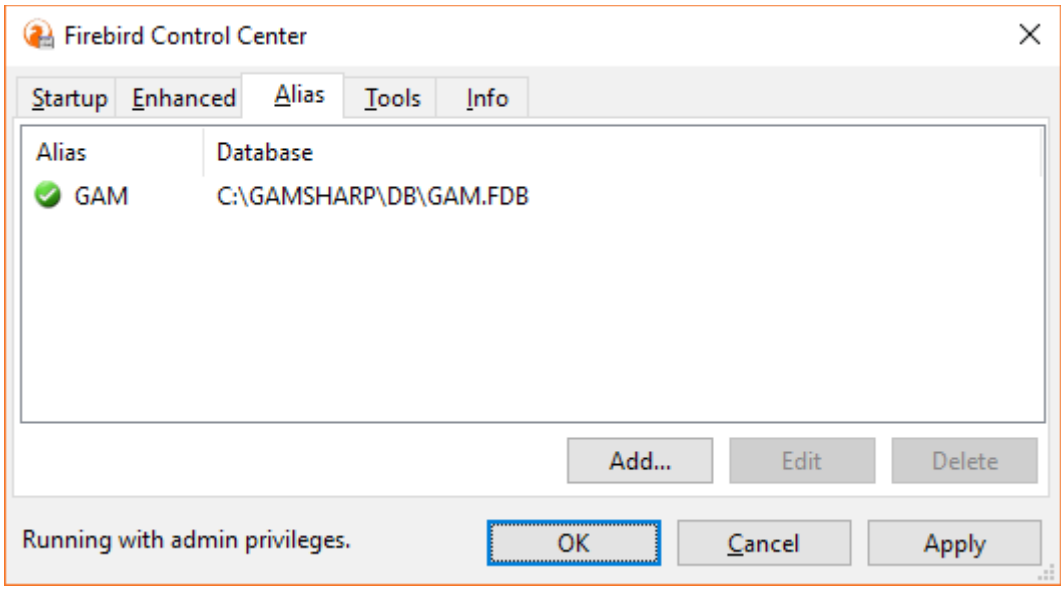


*Illustrazione 3: Alias per il nostro DB*

### Sicurezza e privacy

Al programma è necessario accedere inserendo nome utente e password.

Profili utente:

- Standard
- Amministratore: l'amministratore può eliminare i record creati da qualsiasi utente e può gestire la tabella degli utenti.

## Requisiti di gestione

Questa sezione è relativa alla gestione del progetto in esercizio, dopo la sua installazione.

### Infrastruttura di esercizio del progetto

- Server: il computer più performante connesso in rete, presente nel centro, con sistema operativo Microsoft Windows, che dovrà rimanere perennemente acceso.
- Client: computer connesso in rete, presente nel centro, con sistema operativo Microsoft Windows.

### Gestione dei sistemi

- Prima installazione: Maione Michele.
- Manutenzione sistemi: Informatici senza frontiere.

### Gestione del programma

- Gestione futura: Informatici senza frontiere.

### Gestione degli utenti

- Formazione personale: Maione Michele.
- HelpDesk: Informatici senza frontiere.

## Requisiti di gestione del progetto

### Tempi e risorse

Il progetto verrà consegnato entro e non oltre la fine del 2015.

### Gruppo di progetto

Il progetto verrà sviluppato da “Informatici senza frontiere”, in particolare da Maione sotto la supervisione di Lubrano e De Vito.

### Ambiente di sviluppo

Il progetto verrà realizzato in C# (.NET 2.0) su Microsoft Windows 10[3], tramite Microsoft Visual Studio Community 2015[4].

3 http://www.microsoft.com/windows
4 http://www.visualstudio.com

# II. Struttura dell'applicazione

In questo capitolo parleremo di come è strutturato GAM#, di quali tecnologie sono state utilizzate e quali metodologie di progettazione scelte.

L'illustrazione 4 mostra GAM# in esecuzione, sono stati eseguiti i seguenti passaggi: *Menù Centro* → *Laboratori* → doppio click su un *laboratorio* → click sul pulsante di dettaglio del *Proprietario* → click sul pulsante di ricerca del *Padre*.

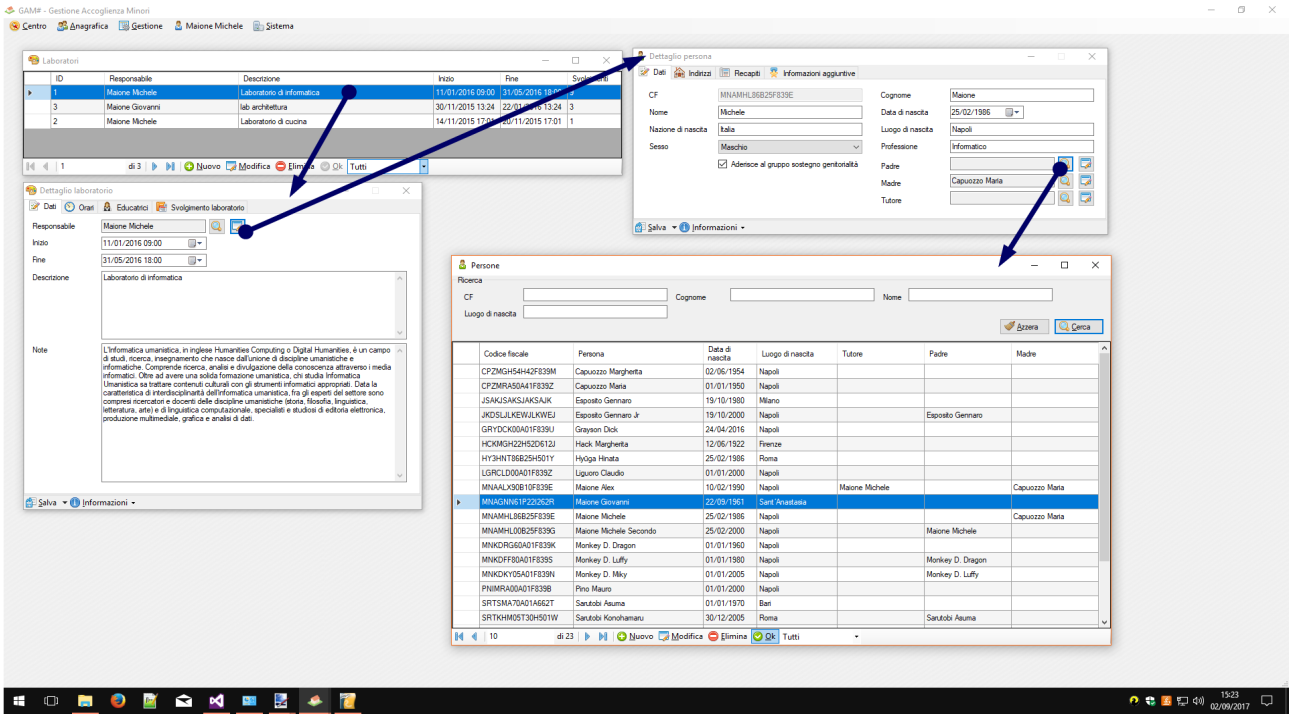

*Illustrazione 4: GAM#, apertura di varie finestre modali*

Ogni finestra di GAM# è di tipo modale, e cioè non è possibile utilizzare le altre finestre del programma finché non si è chiusa l'ultima finestra aperta. In questo modo si obbliga l'utente a fare un task alla volta; esse implementano il modello CRUD.

## CRUD

Il modello CRUD (Create, Read, Update, Delete) si riferisce alle quattro funzionalità che devono essere implementate per poter considerare completa un'applicazione RESTful; Ad ogni operazione corrisponde un'equivalente istruzione in linguaggio SQL:

| Operazione | SQL |
|---|---|
| Create | `insert` |
| Read | `select` |
| Update | `update` |
| Delete | `delete` |

*Tabella 1: Corrispondenza operazioni CRUD con comando SQL*

# 1. Analisi e architettura dell'applicazione

GAM# è una applicazione sviluppata secondo l'architettura Client-server a 2 livelli.

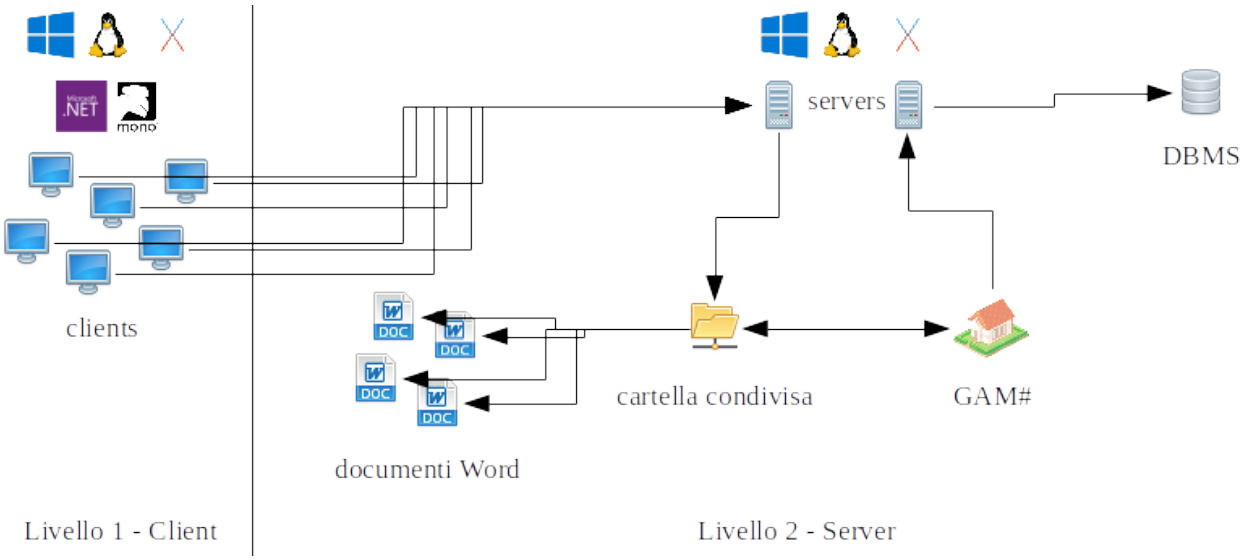


*Illustrazione 5: Pattern architetturale Client-server*

Come si può notare dall'illustrazione 5, l'architettura dell'applicazione è composta da 2 livelli:

- Server:
  - computer sul quale è stato installato un RDBMS (nel nostro caso è stato usato Firebird SQL);
  - computer sul quale è stato installato GAM#. Al momento dell'installazione GAM# rende la cartella in cui risiede condivisa.
- Client:
  - computer su cui è presente o il .NET framework (l'argomento in dettaglio a pagina 17) o il Mono framework, che ha accesso alla cartella condivisa di GAM# sul server, da cui si può eseguire il programma ed accedere ai documenti Word.

Un’architettura Client-server a 2 livelli, per un gestionale desktop, era, francamente, già nel 2015 “antica”, ma era l’unica che rispecchiasse i 3 requisiti che mi erano stati imposti: gratis, integrazione con Microsoft Office e consegna in 4 mesi di un progetto molto grande.

Per rispondere al primo requisito fu subito scartata la possibilità di noleggiare un server per ospitare una qualsiasi applicazione web, per rispondere al secondo requisito fu scelto il C#, e per rispondere al terzo requisito scelsi di riutilizzare i sorgenti C# del mio software di contabilità open-source, RationesCurare[5]. In questo modo avevo una base solida e testata su cui costruire GAM#.

## .NET Framework

È l'ambiente per la creazione, la distribuzione e l'esecuzione di tutti gli applicativi che supportano .NET, siano essi servizi web o altre applicazioni. Si compone di:

- Compilatori per i principali linguaggi supportati da Microsoft
- Ambiente di esecuzione Common Language Runtime o CLR
- Libreria di classi

Il Common Language Runtime è il motore d'esecuzione della piattaforma .NET esegue cioè codice IL (Intermediate Language) compilato con compilatori che possono avere come target il CLR. Tale componente si occupa di compilare just-in-time (al volo) il codice IL in linguaggio macchina, direttamente eseguibile dalla CPU.

5 http://www.maionemiky.it

# 2. Progetto del database

Come anticipato precedentemente il RDBMS scelto è Firebird SQL. Il database è formato da 35 tabelle.

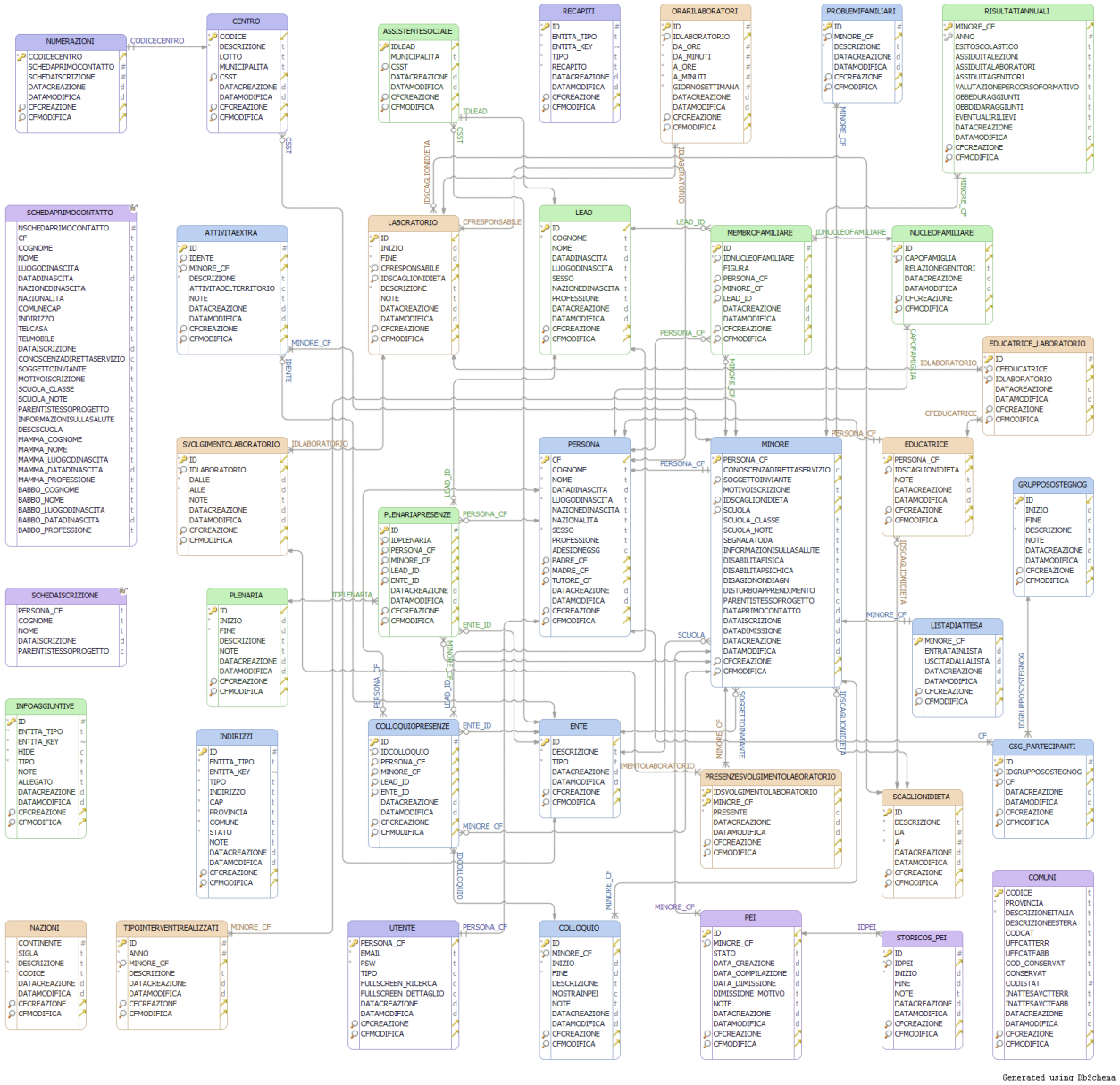


*Illustrazione 6: Layout completo del database*

## Firebird

Firebird SQL è un database management system relazionale (RDBMS), open source distribuito sotto licenza IPL.

Supporta numerosi sistemi operativi tra i quali: Linux, Windows, FreeBSD, macOS.

Le principali caratteristiche di questo Database sono l'alto livello di conformità con gli standard SQL, la completa integrazione con molti linguaggi di programmazione e la facile installazione e manutenzione del software.

Il database utilizzato da Firebird è normalmente un file con estensione *.fdb*, la massima dimensione consentita è pari alla dimensione massima di un file supportata da uno specifico sistema operativo, Firebird, però, permette di suddividere un singolo database in più file (massimo 65536) e quindi sarà possibile gestire un database che abbia una dimensione limitata solo dalla capienza fisica dell'hard disk.

Il numero di client che possono contemporaneamente collegarsi al server è teoricamente illimitato, ma tale numero dipende strettamente dal sistema operativo e dall'hardware in uso.

## Entità

### Anagrafiche

Le entità anagrafiche presenti nel sistema derivano prevalentemente da Lead e Persona. La differenza tra queste due entità è che la seconda ha come chiave primaria e univoca il codice fiscale, mentre la prima utilizza un semplice auto-incrementale, questo perché tutte le entità che derivano da Persona hanno rapporti diretti con il Centro e devono essere censite in modo preciso.

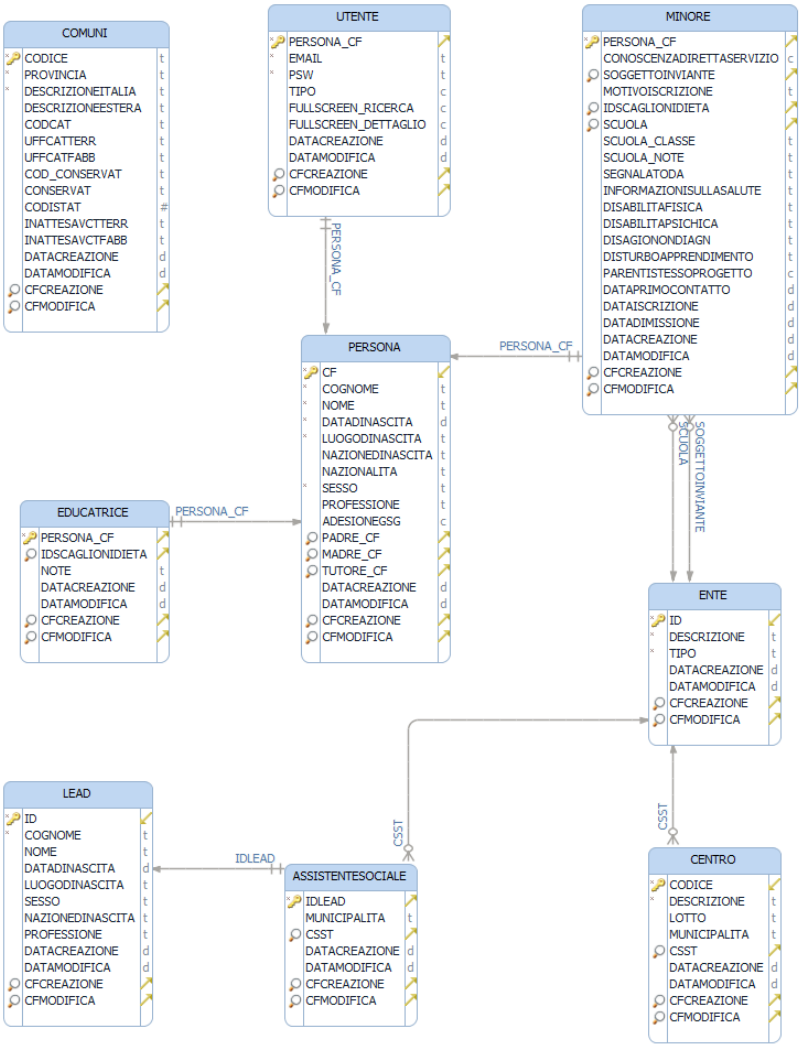


*Illustrazione 7: Tabelle anagrafiche*

# Minore

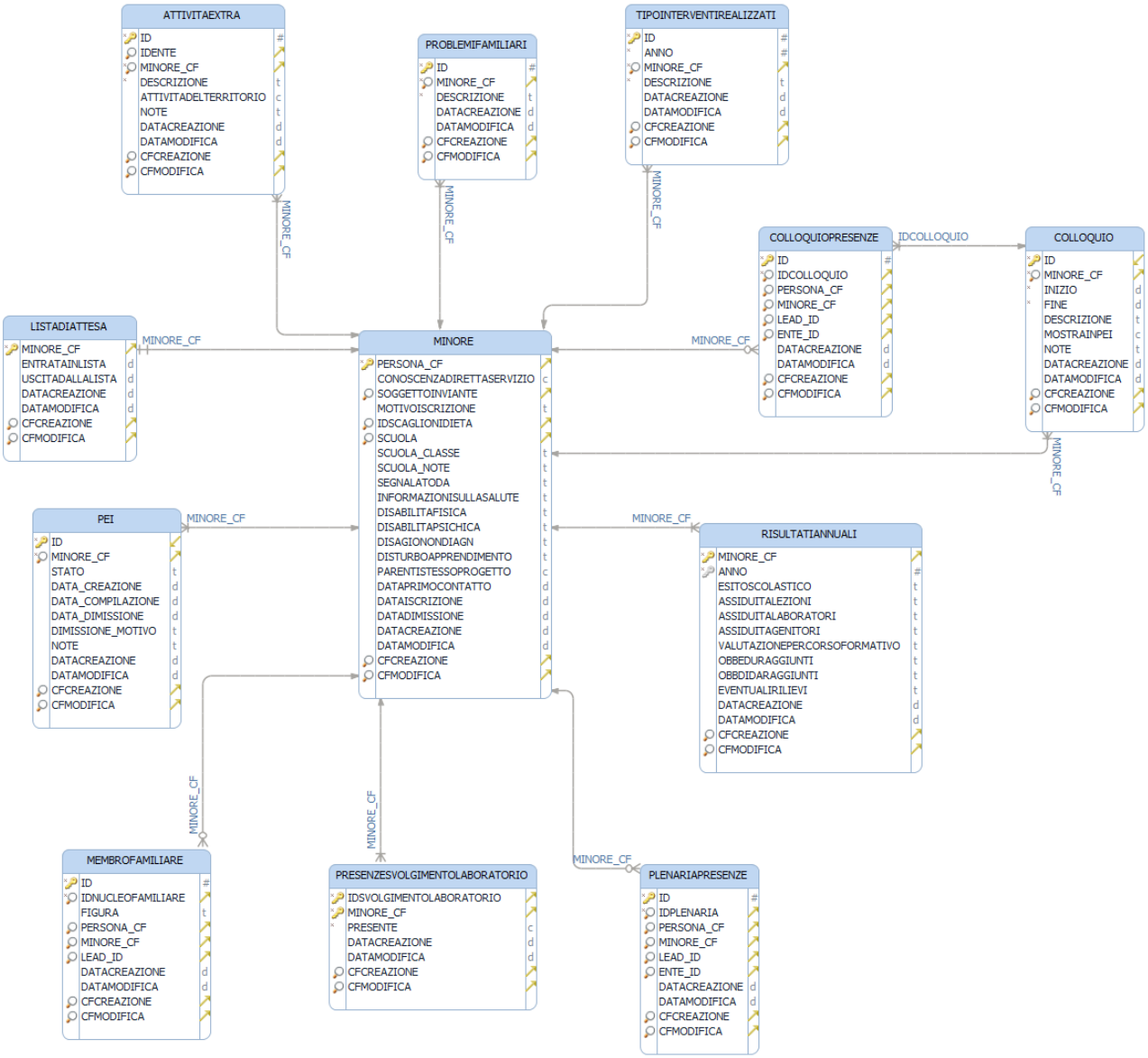


*Illustrazione 8: Tabelle che interagiscono con il minore*

# Attività

## Laboratorio

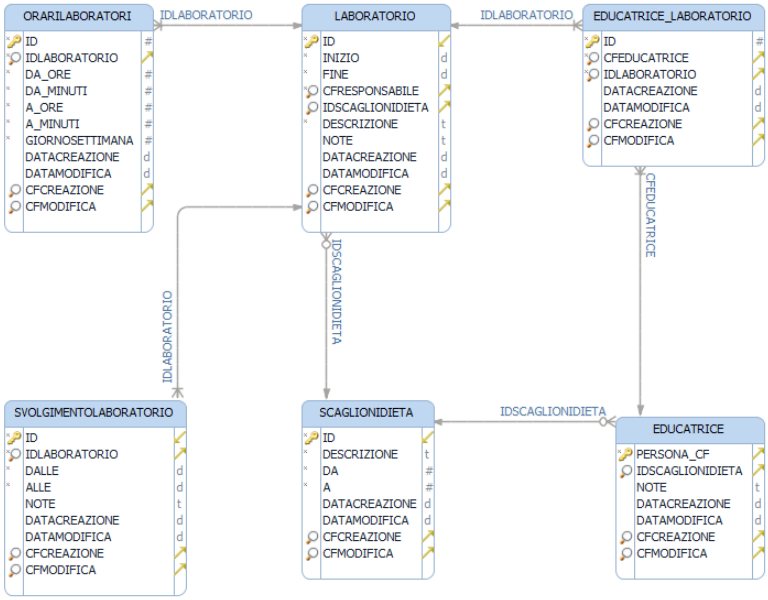


*Illustrazione 9: Tabelle che interagiscono con il laboratorio*

## Plenaria

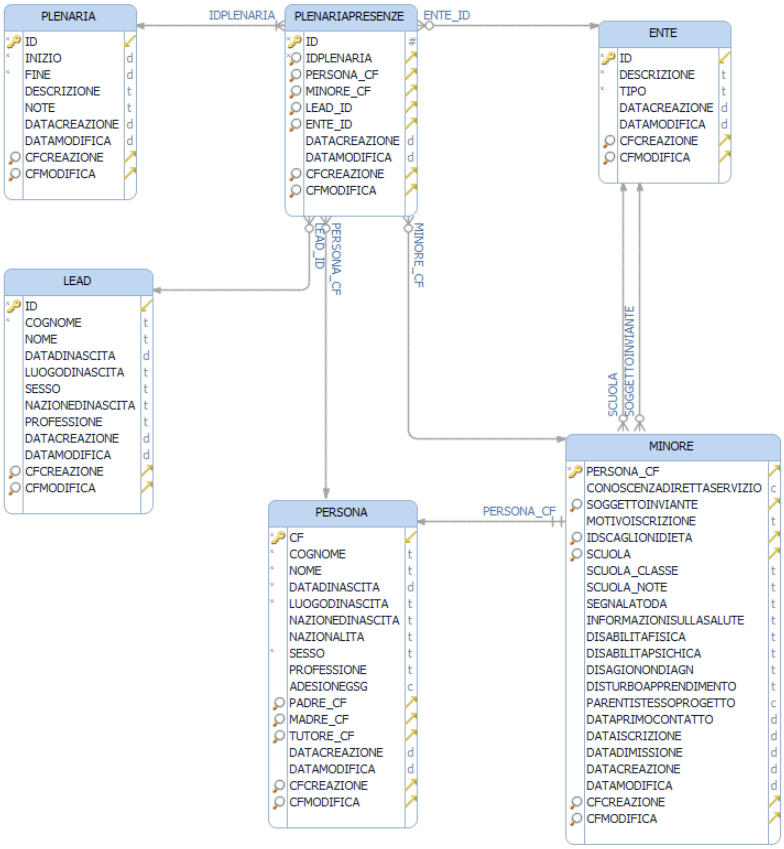


*Illustrazione 10: Tabelle che interagiscono con la plenaria*

## Colloquio

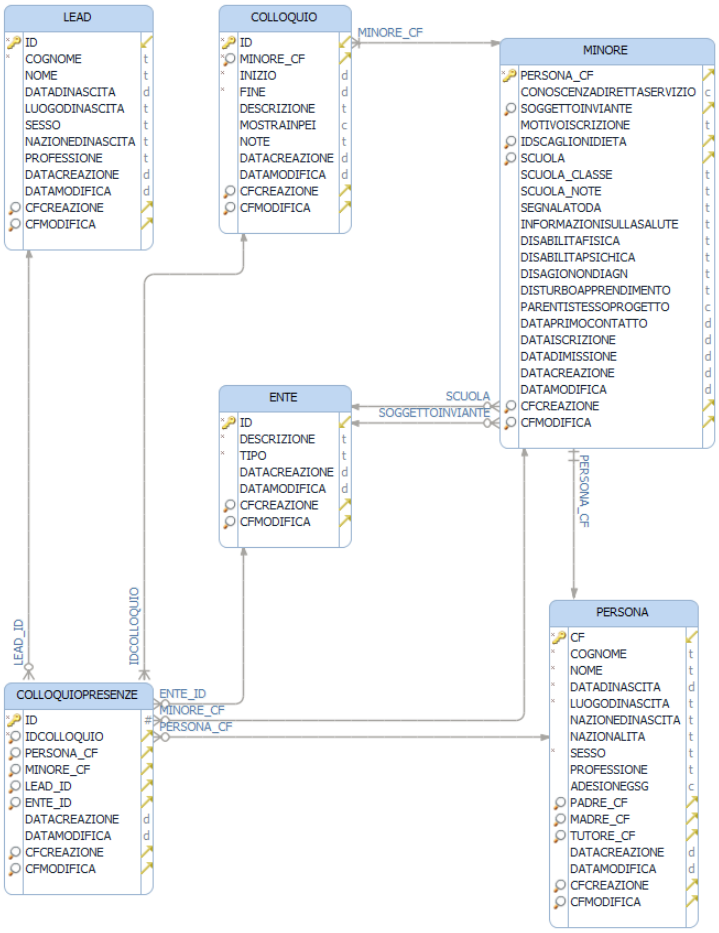


*Illustrazione 11: Tabelle che interagiscono con il colloquio*

# 3. Progetto dell'applicazione

## Design pattern

Nell'ingegneria del software, un design pattern può essere definito "una soluzione progettuale generale ad un problema ricorrente".

Un design pattern è una descrizione o un modello da applicare per risolvere un problema nel contesto della realizzazione di sistemi software orientati agli oggetti. Inoltre, un design pattern è indipendente dal linguaggio e dall'implementazione, è una micro-architettura e non è meccanicamente applicabile.

Un design pattern è costituito da quattro elementi essenziali, che ne mettono in evidenza le caratteristiche e lo rendono univoco:

- Nome: costituito da una o più parole che rappresentino il pattern stesso e ne descrivano il problema;
- Problema: descrizione della situazione alla quale si può applicare il pattern, ad esempio una lista di condizioni che spieghi la necessità dell'utilizzo del pattern;
- Soluzione: descrizione delle modalità di utilizzo degli elementi che costituiscono il progetto con le relazioni e le relative implicazioni, al fine di implementare i requisiti individuati dal problema;
- Conseguenze: risultati e vincoli scaturiti dall'applicazione del pattern, importanti per valutare modelli alternativi e per stimare i vantaggi apportati dall'applicazione del pattern.

## Il pattern MVP

Dopo aver spiegato cosa sia e a cosa serva un design pattern, vediamo ora in dettaglio le caratteristiche del più importante pattern da noi utilizzato in questo sistema: il pattern Model-View-Presenter (MVP).

Il pattern MVP, risale all'anno 1979 ed il suo nome originario era "Thing Model View Editor" (coniato dalla Taligent), nel tempo poi evoluto grazie soprattutto a Martin Fowler che ne ha sviluppato due diverse implementazioni Supervising Controller e Passive View. La soluzione proposta prevede il disaccoppiamento totale della View dalle logiche di manipolazione del Model tramite l'introduzione di un componente, il Presenter, che funga da intermediario e da coordinatore in risposta alle interazioni con l'utente. Ma vediamo nel dettaglio gli attori coinvolti:

- View (visualizza i dati contenuti nel model e raccoglie gli input dell'utente);

- Model (rappresenta il dominio di business ed incapsula lo stato dell'applicazione);
- Presenter (interagisce con il model in base alle richieste ricevute dalla view).

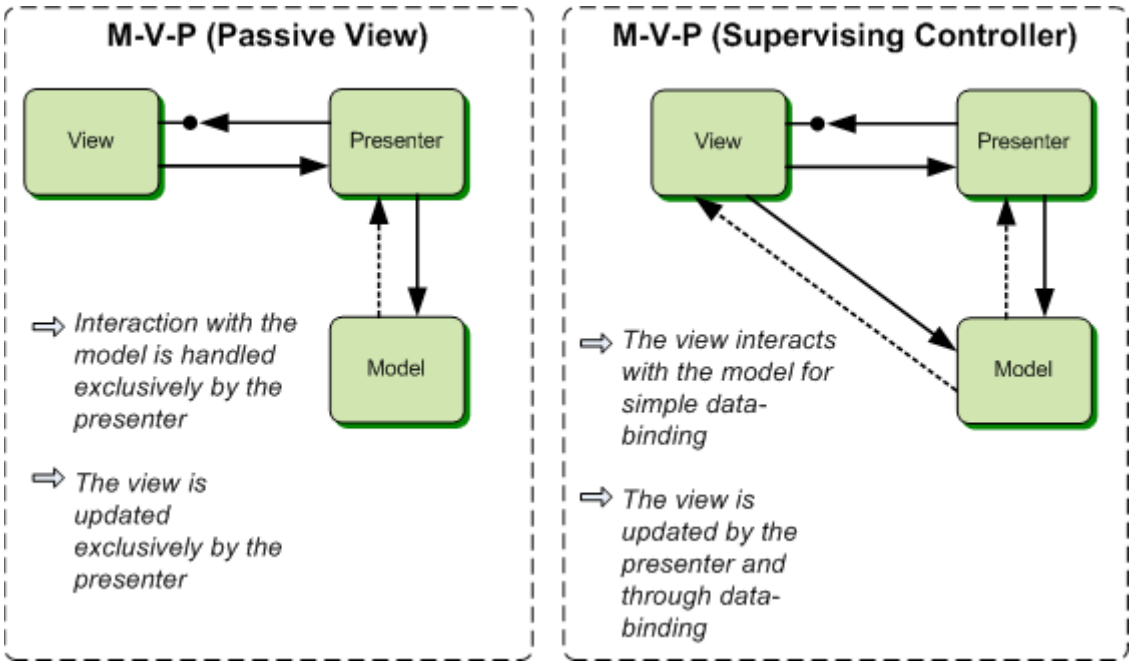


*Illustrazione 12: I due schemi di interazione del pattern MVP*

## Schema di interazione – MVP Passive View

In questo caso la view non conosce il modello (non vi è quindi alcuna dipendenza tra la View ed il Model) ed il presenter è l'unico responsabile dell'aggiornamento dei dati visualizzati.

- L'utente interagisce in qualche modo con la View;
- La View notifica al Presenter dell'interazione;
- Il Presenter opera sul Model eventualmente modificandone lo stato in rispetto all'interazione notificata dalla View;
- Il Presenter aggiorna la View in base ai nuovi dati esposti dal Model.

## Schema di interazione – MVP Supervising Controller

In questo tipo di implementazione la view conosce il modello e generalmente è connessa ad esso tramite un meccanismo di data binding dichiarativo che permette l'aggiornamento dei dati da visualizzare (in situazioni complesse dove il data binding dichiarativo non è sufficiente a garantire tale l'aggiornamento è necessario l'intervento del presenter).

- L'utente interagisce in qualche modo con la View;
- La View notifica al Presenter dell'interazione;

- Il Presenter opera sul Model eventualmente modificandone lo stato in rispetto all'interazione notificata dalla View;
- La View viene aggiornata in base ai nuovi dati esposti dal Model tramite un meccanismo di data binding;
- Il Presenter aggiorna i dati della view che non possono essere agganciati a quest'ultima tramite data binding dichiarativo.

Nel nostro caso si è scelto di utilizzare MVP – Supervising Controller, per sfruttare le funzionalità di data-binding statico.

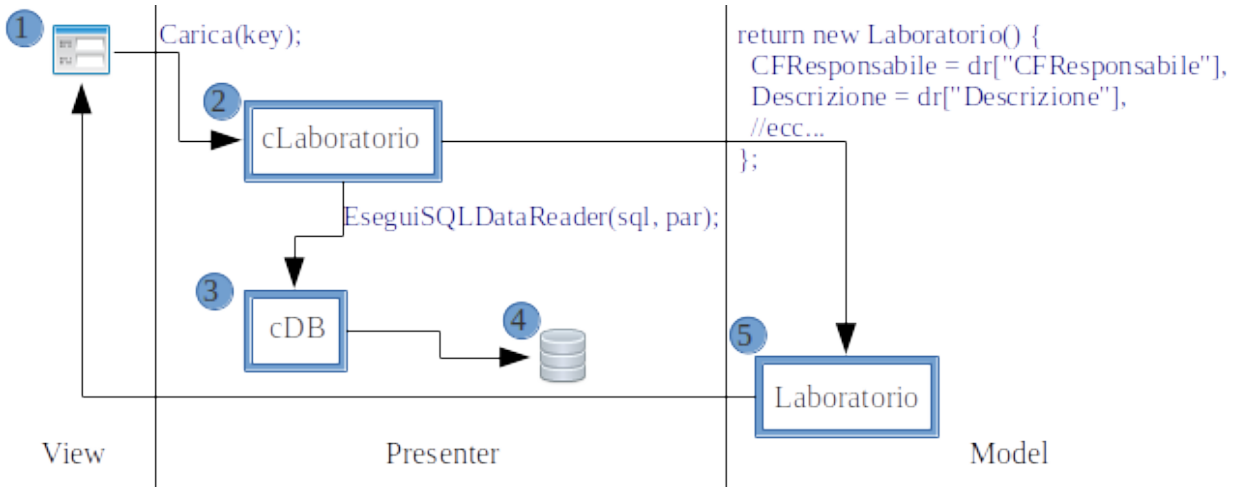


*Illustrazione 13: MVP - Supervising Controller dell'apertura del dettaglio di un Laboratorio*

Nell'illustrazione 13, e in modo dettagliato nella 14, si vede come GAM# utilizzi il pattern MVP Supervising Controller.

Adesso scenderemo nei dettagli progettuali passando dalle classi generiche fino alle classi dettagliate, attraversando 3 livelli di dettaglio.

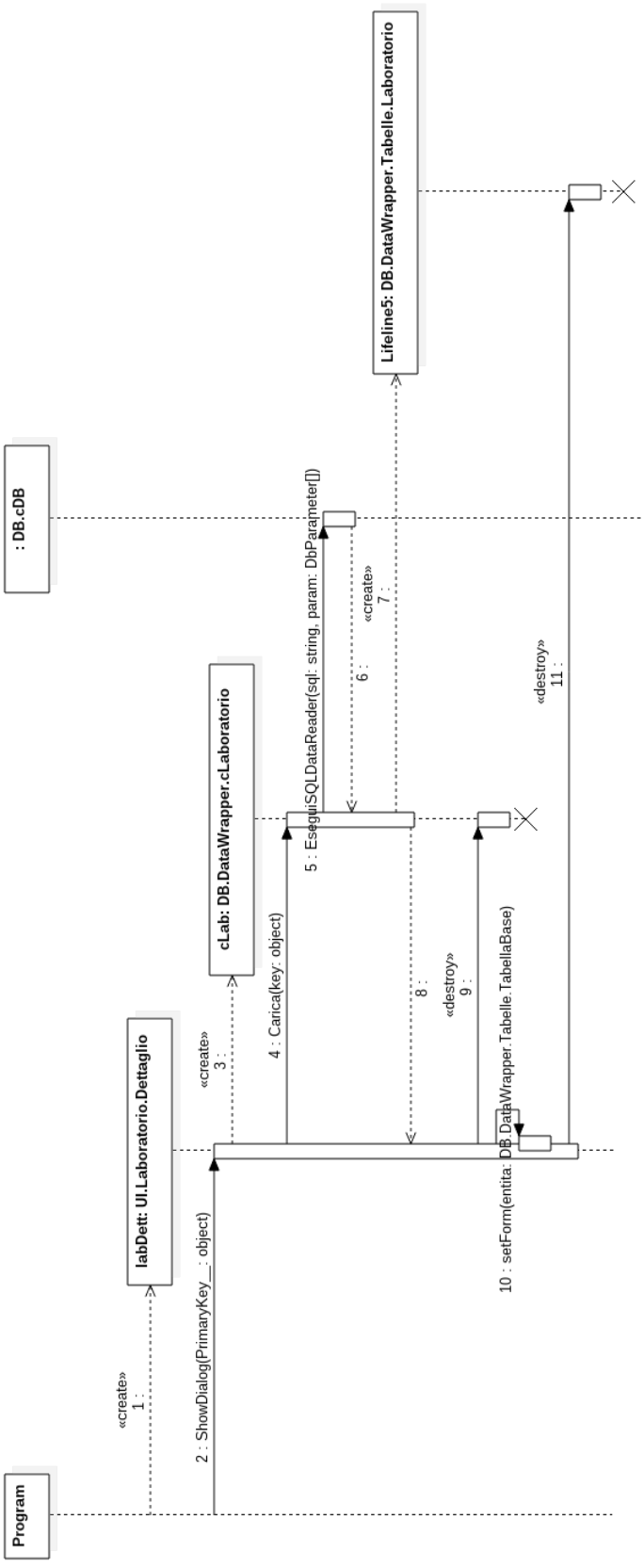


*Illustrazione 14: Sequence diagram dell'apertura del dettaglio di un Laboratorio*

## MVP – Livello 1

- Model: tutte le classi che implementano BaseDBObject, sono dei wrapper delle tabelle del DB. In questo caso la classe espone dei metodi che tramite reflection restituiscono, della chiave primaria se presente, il valore, il nome e se se sia auto-incrementale;
- Presenter: tutte le classi che implementano cBaseDBObject, che a questo livello offre le sole funzioni di select, del record per chiave primaria (per il dettaglio) o dei record per parametri (per la ricerca);
- View: tutte le form che implementano fBase, e tutti i documenti Word che vengono generati, fanno parte della View. La form fBase, gestisce giusto due, tre proprietà che riguardano la mera visualizzazione, come partire in full screen, e chiudersi alla pressione del tasto ESC.

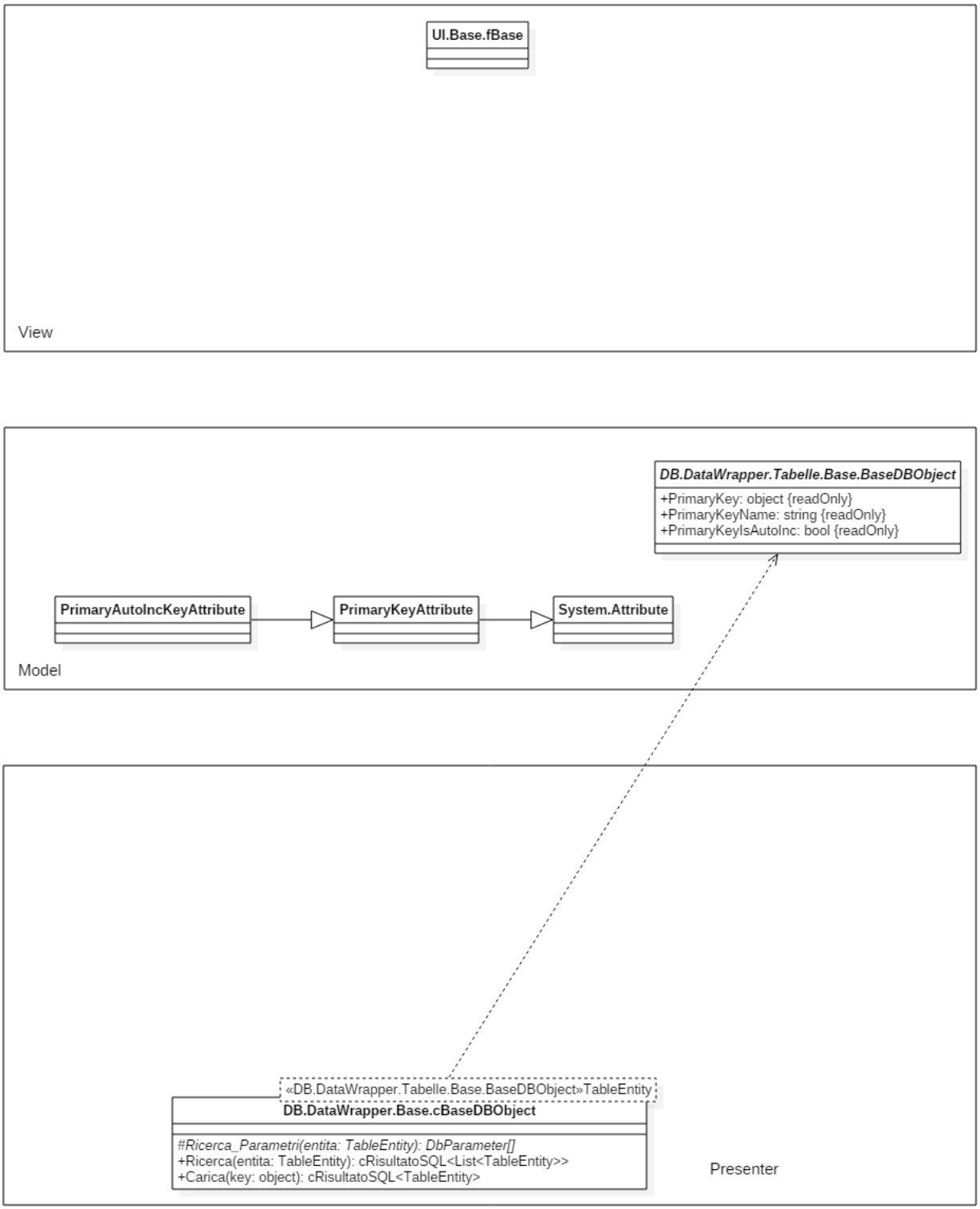


*Illustrazione 15: MVP - Livello 1*

## MVP – Livello 2

- Model: a questo livello, abbiamo la classe TabellaBase che implementa BaseDBObject, ed offre le proprietà che avranno tutte le tabelle: data creazione, data modifica, utente creazione, utente modifica;
- Presenter: qui abbiamo cBaseEntity, che implementa cBaseDBObject, che aggiunge le funzionalità di insert, update e delete;
- View: a questo livello abbiamo le implementazioni fBaseDettaglio e fBaseRicerca, che implementano fBase, che offrono le toolbar con i pulsanti CRUD e la griglia, nel caso di fBaseRicerca, e le toolbar dei dati generici del record (ultima modifica, creazione, utente creazione, utente modifica) e il pulsante di salvataggio.

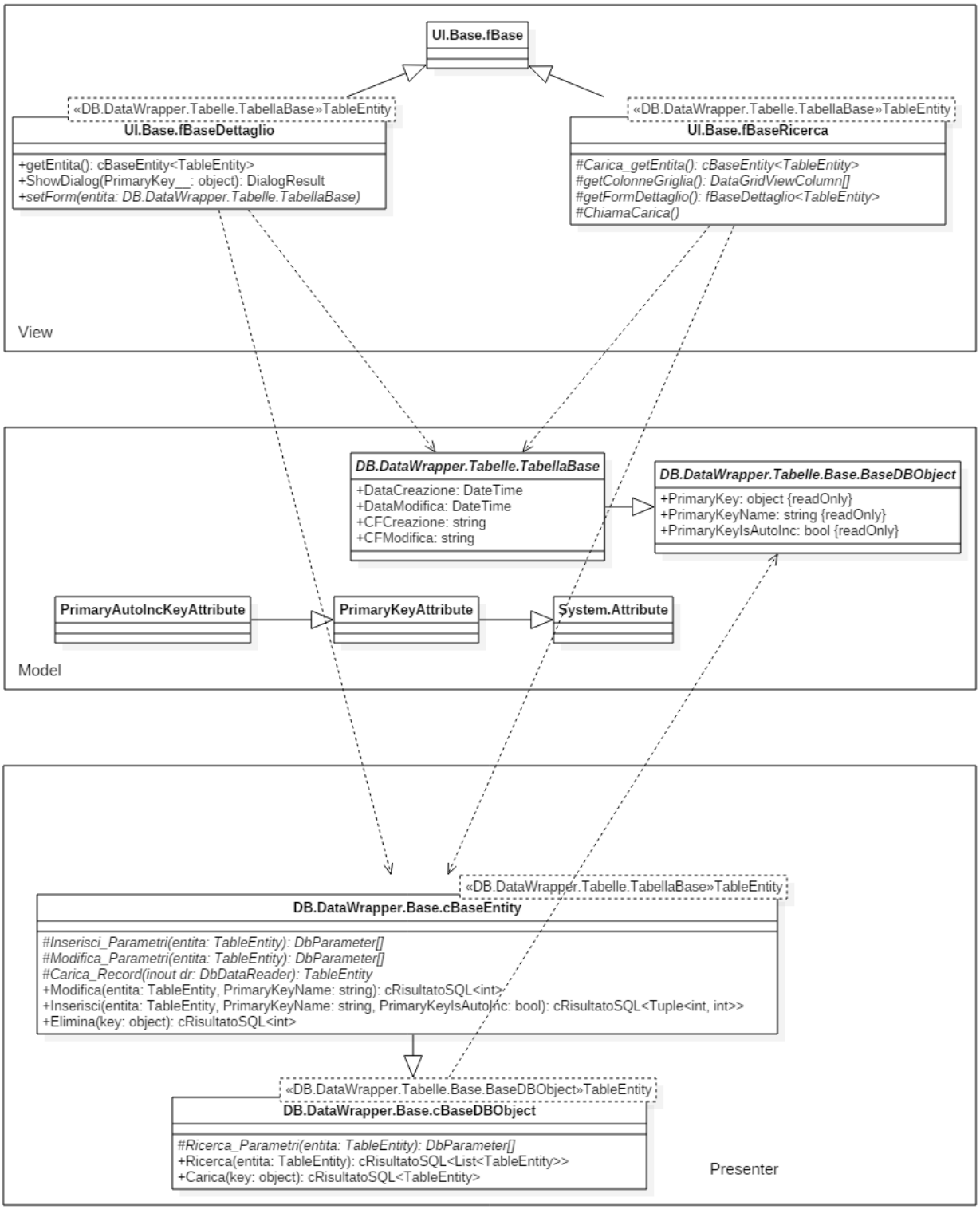


*Illustrazione 16: MVP - Livello 2*

## MVP – Livello 3

Al livello 3 vedremo l'implementazione del Laboratorio.

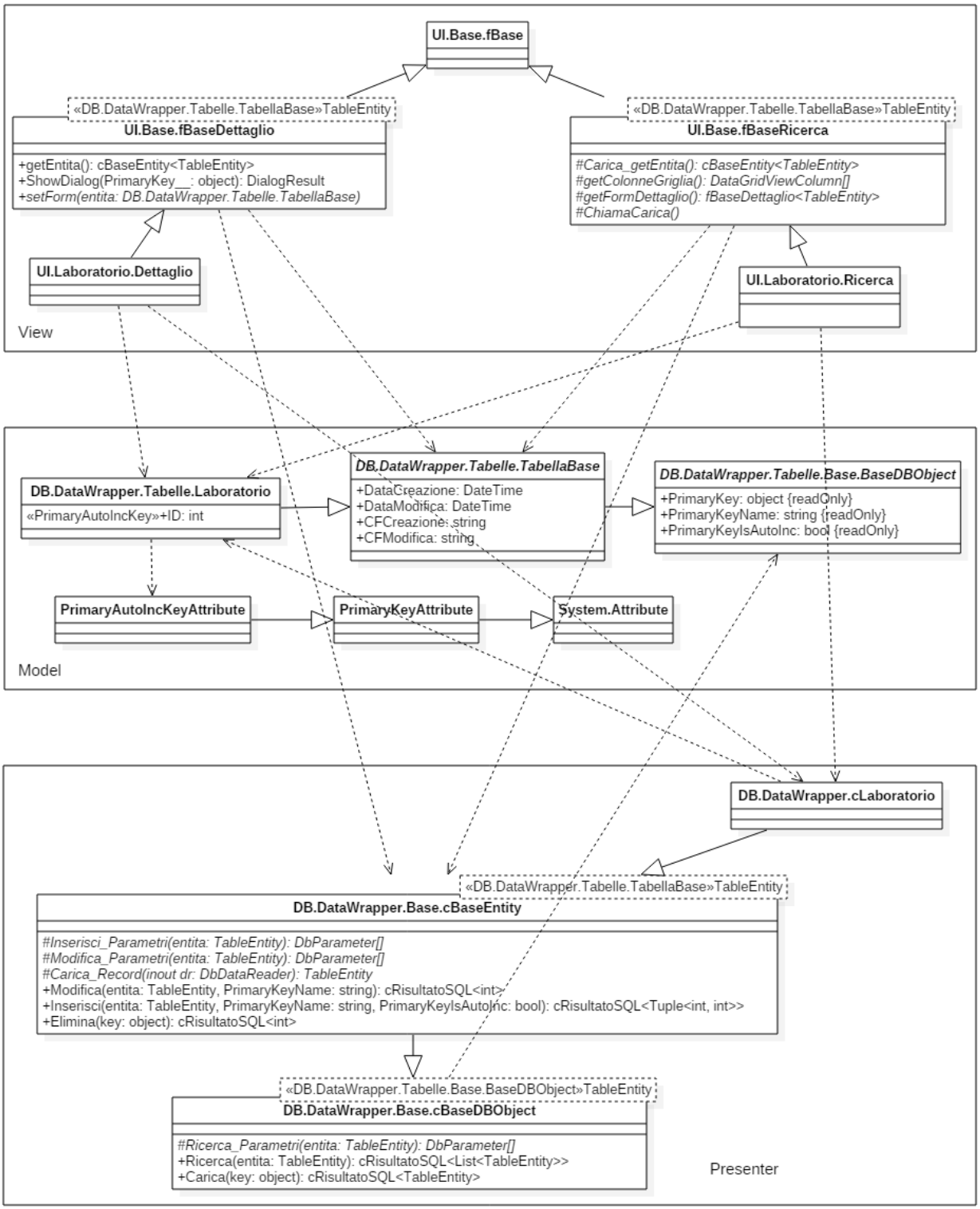


*Illustrazione 17: MVP - Livello 3, implementazione del Laboratorio*

- Model: al livello 3, la classe ha il nome della tabella, in questo caso **Laboratorio**, che eredita da **TabellaBase**. Laboratorio contiene per ogni colonna della tabella, una variabile per memorizzarne il valore;

```
create table Laboratorio
(
	ID int not null primary key,

	Inizio timestamp not null,
	Fine timestamp not null,

	CFResponsabile varchar(16) not null,
	IDScaglioniDiEta int,

	Descrizione varchar(250) not null,
	Note varchar(5000),

	DataCreazione timestamp default CURRENT_timestamp,
	DataModifica timestamp default CURRENT_timestamp,
	CFCreazione varchar(16),
	CFModifica varchar(16),

	foreign key (IDScaglioniDiEta) references ScaglioniDiEta(ID),
	foreign key (CFResponsabile) references Persona(CF),
	foreign key (CFCreazione) references Persona(CF),
	foreign key (CFModifica) references Persona(CF)
);
```

*Tabella 2: Laboratorio: SQL*

```
sealed class Laboratorio : TabellaBase
{
	[PrimaryAutoIncKey]
	public int ID { get; set; }
	public DateTime Inizio { get; set; }
	public DateTime Fine { get; set; }
	public int IDScaglioniDiEta { get; set; }
	public string CFResponsabile { get; set; }
	public string Descrizione { get; set; }
	public string Note { get; set; }
	public sExRicerca ExRicerca;

	public struct sExRicerca
	{
		public string Responsabile, Scaglione;
	}
}
```

*Tabella 3: Laboratorio: Classe*

- Presenter: a questo livello, la classe si chiama **cLaboratorio**, che estende **cBaseEntity**, ed espone i metodi per incapsulare ed estrarre dal model i dati;

```
protected override DB.DataWrapper.Tabelle.Laboratorio getForm()
{
	return new DB.DataWrapper.Tabelle.Laboratorio()
	{
		ID = PrimaryKey,
		CFResponsabile = eCFResponsabile.Value,
		Descrizione = eDescrizione.Text,
		IDScaglioniDiEta = cGB.GetSelectedComboItem_ID(eIDScaglioniDiEta),
		Note = eNote.Text,
		Inizio = eInizio.Value,
		Fine = eFine.Value
	};
}
```

*Tabella 4: Laboratorio: settaggio del model nel view, recupero model dal view*

- View: le classi a questo livello sono **UI.Laboratorio.Dettaglio** e **UI.Laboratorio.Ricerca**, che estendono **fBaseDettaglio** e **fBaseRicerca**.

```
protected override Laboratorio Carica_Record(ref DbDataReader dr)
{
        return new Laboratorio()
        {
                CFResponsabile = cGB.ObjectToString(dr["CFResponsabile"]),
                Descrizione = cGB.ObjectToString(dr["Descrizione"]),
                Note = cGB.ObjectToString(dr["Note"]),
                Inizio = cGB.ObjectToDateTime(dr["Inizio"]),
                Fine = cGB.ObjectToDateTime(dr["Fine"]),
                ID = cGB.ObjectToInt(dr["ID"], -1),
                IDScaglioniDiEta = cGB.ObjectToInt(dr["IDScaglioniDiEta"], -1),
                ExRicerca = new Laboratorio.sExRicerca()
                {
                        Scaglione = cGB.ObjectToString(dr["Scaglione"]),
                        Responsabile = cGB.ObjectToString(dr["Responsabile"])
                }
        };
}

protected override DbParameter[] Inserisci_Parametri(Laboratorio entita)
{
        return new DbParameter[] {
                cDB.NewPar("Inizio", entita.Inizio),
                cDB.NewPar("Fine", entita.Fine),
                cDB.NewPar("CFResponsabile", entita.CFResponsabile),
                cDB.NewPar("Descrizione", entita.Descrizione),
                cDB.NewPar("IDScaglioniDiEta", entita.IDScaglioniDiEta),
                cDB.NewPar("Note", entita.Note)
        };
}
```

*Tabella 5: Laboratorio: Metodi per incapsulare ed estrarre dal model i dati.*

# III. Collaudo del software

In questo capitolo verrà illustrata la fase di testing effettuata per verificare il corretto funzionamento dell'applicazione e l'assenza di malfunzionamenti.

## 1. Introduzione al testing

L'ultima fase del processo di realizzazione del software è stata il testing, fase che ha come scopo principale quello di verificare la correttezza del software sviluppato, identificando eventuali differenze tra il comportamento atteso del software, descritto nelle specifiche del progetto, ed il comportamento osservato dallo stesso. A tal proposito è stato realizzato un piano di testing atto ad accertare il corretto funzionamento dell'applicazione. Questo piano prevede due fasi distinte di testing (distinte per modalità di approccio):

- Al completamento dell'implementazione di ogni singolo modulo, sono stati testati i metodi al fine di controllare eventuali malfunzionamenti, che i requisiti siano stati rispettati e che i risultati attesi siano stati rispettati nell'esecuzione dei moduli testati. Il metodo scelto per collaudare le funzionalità dei moduli è il Black Box Testing. Tale testing non è una vera tipologia di test, ma più una strategia di sperimentazione che non richiede conoscenza di codice e di funzionamento interno del sistema. Le prove effettuate sono incentrate sul testing dei requisiti e delle funzionalità del software. La difficoltà della strategia Black Box sta nel selezionare dati di input adeguati che testino le funzionalità ed i requisiti del sistema. Tale selezione è possibile farla tramite:
    - Classi di equivalenza: rappresentano un insieme di stati validi o non validi per una condizione sulle variabili d'ingresso. Per quanto riguarda i valori delle variabili d'ingresso bisogna fare delle distinzioni in base al tipo definito. Infatti nel caso in cui la variabile possa avere valori compresi in un intervallo bisogna creare almeno una classe di test valida per un valore interno all'intervallo, una non valida per valori inferiori al minimo valore dell'intervallo, una non valida per valori superiori al massimo valore dell'intervallo ed una classe di test per valori uguali o ravvicinati agli estremi dell'intervallo. Nel caso di variabile con valore specifico bisogna generare una classe valida per il valore specificato, una non valida per valori inferiori a quello e una non valida per valori superiori. Se la condizione sulle variabili di ingresso specifica invece un elemento di un insieme di-

screto bisogna creare una classe valida per ogni elemento dell'insieme e una non valida per valori non appartenenti all'insieme. Infine, nel caso di specifica di un valore booleano sono richieste una classe valida per il valore TRUE ed una non valida per il valore FALSE. In conclusione, le regole pratiche per la scelta delle classi di equivalenza richiedono che ogni classe di equivalenza sia coperta da almeno un caso di test, ossia ogni classe di test non valida deve corrispondere ad un solo caso di test. Al fine di rendere migliore tale fase di test bisogna fare in modo che ogni caso di test per classi di equivalenza valide comprenda il maggior numero di classi valide ancora scoperte.

    - Boundary Testing: tipologia di testing nella quale è previsto che i test comprendano i valori limite. In pratica si testa il software verificando valori per le variabili di input all'estremo inferiore, immediatamente sopra il valore minimo, un valore intermedio, immediatamente sotto il valore massimo e all'estremo superiore.

- La seconda metodologia utilizzata per il collaudo dell'applicazione è costituita dagli Unit Testing. Essi sono test programmatici scritti (nel nostro caso) in codice C# e definiti in una classe Test contenente i Test Method. Un Test Method non è altro che una procedura che determina un risultato del test. I Test Method possono utilizzare altro codice sorgente chiamando direttamente i metodi di una classe, inviando i parametri appropriati. Questa tipologia di testing racchiude in sé gli aspetti caratteristici delle più comuni strategie di testing (White Box e Black Box Testing). Infatti, grazie alle istruzioni di Assert presenti negli Unit Test, è possibile verificare la corrispondenza tra il valore atteso di un metodo con quello ottenuto (Black Box Testing) e, attivando la funzionalità denominata CodeCoverage, è possibile verificare quali sono e quante volte vengono eseguite le istruzioni dei metodi da testare (White Box Testing). In Visual Studio è possibile creare Unit Test adoperando la funzionalità di generazione del codice che permette di generare in modo automatico il codice sorgente iniziale del test. È comunque possibile scrivere il test completamente a mano.

## 2. Unit testing

Gli unit test sono codice sorgente definito in una classe Test contenente dei Test Method, i quali possono richiamare direttamente i metodi dei moduli da testare, inviando i parametri appropriati. Un punto di forza di tale tipologia di test è la possibilità di poter verificare, tramite il metodo Assert, la corrispondenza del valore atteso con quello ottenuto. È possibile creare unit test o utilizzando una funzionalità che permetta di generare codice con l'obiettivo di realizzare il codice sorgente iniziale del test, oppure scrivendo il test completamente a mano. Inoltre possiamo sottolineare che gli unit

test, al momento della compilazione precedente il rilascio dell'applicazione, vengono richiamati in automatico, in modo che possa essere impedito il rilascio nel caso in cui qualche metodo non superi il test definito. Infatti, è necessario creare uno unit test per ogni metodo che si desidera testare e, inoltre, non esistendo un ordine preciso di richiamo dei vari test, è importante specificare le priorità nel testare prima un metodo invece che un altro. La vera importanza degli unit test risulta soprattutto dal fatto che è possibile verificare il funzionamento di tutti i metodi, dopo aver effettuato delle modifiche, ed inoltre forniscono la possibilità di sapere quali blocchi di codice sono stati testati, quali solo testati parzialmente e quali non sono mai testati, in modo che lo sviluppatore possa capire e quindi creare i test mancanti.

Diamo ora un esempio di testing con la metodologia degli Unit Test, utilizzando NUnit, che è un tool utilizzato per effettuare unit test su codice C#. In particolare il codice di test basato su NUnit permette di:

- produrre il setup di tutte le condizioni necessarie per il testing (creare gli oggetti necessari, allocare le risorse ecc.), in modo da inizializzare i dati necessari per l'esecuzione dei test, grazie al metodo SetUp();
- chiamare il metodo che deve essere testato;
- verificare che il metodo testato funzioni, grazie ai metodi Assert;
- liberare la memoria grazie al metodo TearDown().

## Esempio

```
using NUnit.Framework;
using GAMSharp.DB.DataWrapper.Tabelle;

namespace GAMSharp.DB.DataWrapper
{
    [TestFixture]
    public class TcLaboratorio
    {
        private cLaboratorio cLaboratorio = new cLaboratorio();

        [Test]
        public void Caricamento()
        {
            var R = cLaboratorio.Ricerca(new Laboratorio());
            Assert.AreEqual(R.Errore, false, R.Eccezione.Message);

            foreach (var lab1 in R.Risultato)
            {
                var lab2 = cLaboratorio.Carica(lab1.ID);
                Assert.AreSame(lab1, lab2);
            }
        }
    }
}
```

*Tabella 6: TcLaboratorio, classe per test NUnit*

Eseguiremo il testing sulla classe **cLaboratorio** dei due metodi che eseguono la **select** sulla tabella **Laboratori**: **Ricerca** e **Carica**.

Il primo passo da fare è creare lo Unit Test tramite le procedure guidate messe a disposizione da Visual Studio o da MonoDevelop e dal tool NUnit. Una volta terminata tale procedura viene generato il codice sorgente iniziale del test e si procede con il completamento del codice del test method.

Come si può notare, il test prevede la realizzazione di una classe **TcLaboratorio**, all'interno della quale vengono definiti i diversi metodi di test. Il metodo **Caricamento** serve per effettuare il test vero e proprio e in esso troviamo due istruzioni Assert che verificano che il valore atteso (il primo parametro) sia uguale al valore ottenuto richiamando il metodo da testare (il secondo parametro). Nella prima istruzione Assert, verifichiamo che il metodo Ricerca non abbia restituito errori. Nella seconda istruzione Assert, verifichiamo invece se per ogni Laboratorio restituito dal metodo Ricerca, il metodo Carica recuperi dal DB lo stesso record. Infatti passiamo al metodo Carica un oggetto avente un ID realmente esistente nel sistema, e quindi il metodo deve restituire un oggetto contenente tutti i dettagli di quel Laboratorio, compreso il suo ID. Il test è da considerarsi superato solo nel momento in cui tutte le Assert verificano l'uguaglianza dei relativi parametri. Se ciò si verifica NUnit mostra una schermata in cui, l'esito del test, è evidenziato da una barra di colore verde con il numero di errori riscontrato pari a zero, altrimenti da una barra di colore rosso e dalla presenza di errori, come nell'illustrazione 18, che ha restituito "**Connection property has not been initialized.**"

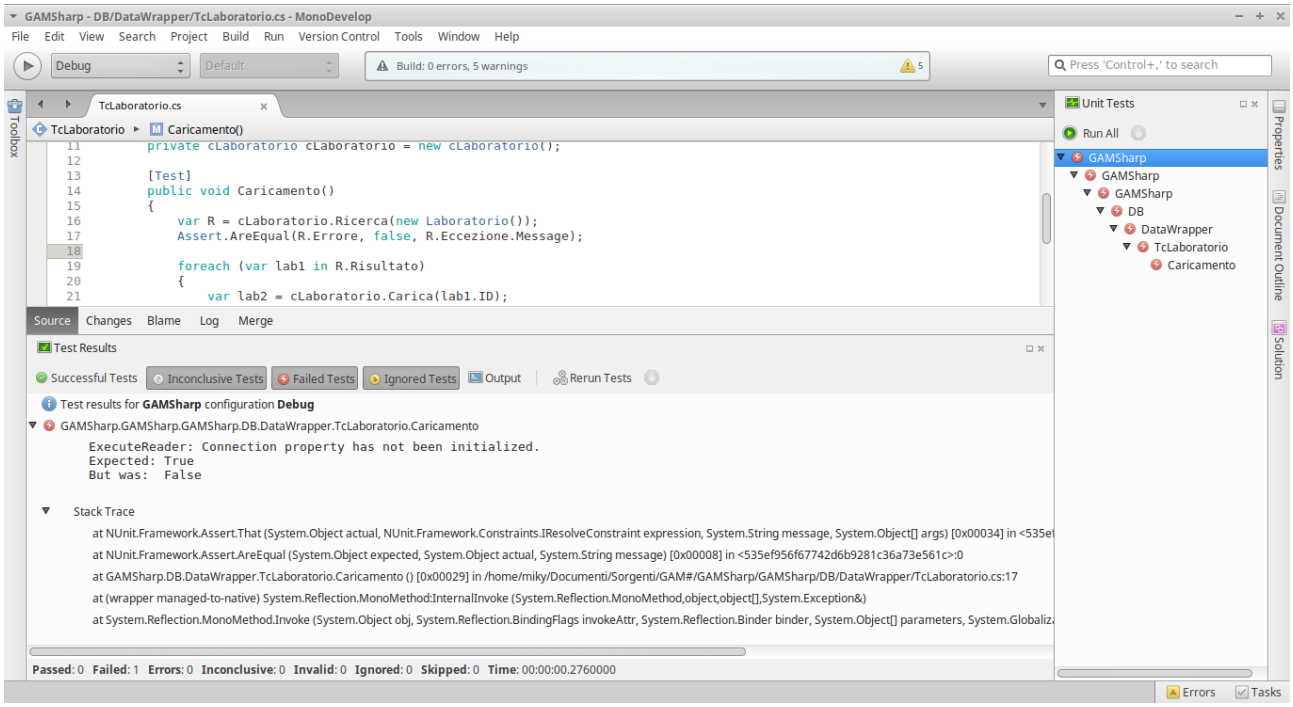


*Illustrazione 18: NUnit integrato in MonoDevelop, test fallito*

# IV. Conclusioni

In questo capitolo vedremo le considerazioni finali sulle attività svolte durante il periodo di tirocinio.

## 1. Considerazioni

Nello sviluppo di questa applicazione, come si evince dalle trattazioni dell'elaborato di tesi, sono stati fatti diversi sforzi al fine di realizzare un'applicazione realmente manutenibile, scalabile e testabile in grado di sfruttare le potenzialità di tutte le tecnologie scelte per lo sviluppo ed in particolare del framework più utilizzato per la realizzazione di applicazioni Windows: .NET. Il software sviluppato è stato necessario per venire incontro alle necessità del Centro "La Tenda" ONLUS di informatizzare procedure che venivano banalmente realizzate, fino a questo momento, utilizzando fogli di calcolo Excel e documenti Word. Il software permette di gestire funzionalità e processi in modo rapido ed efficiente, garantendo anche un grado di affidabilità maggiore rispetto ai documenti Word, dovuta a controlli sui dati immessi e al mantenimento della consistenza degli stessi. Per soddisfare le richieste della ONLUS, dopo una attenta analisi dei requisiti è seguita un'accurata fase di progettazione, dove sono state fatte scelte architetturali che permettessero di sviluppare le funzionalità richieste nel miglior modo possibile rispettando le limitazioni imposte dalla ONLUS. Le scelte fatte, risultano il miglior compromesso per soddisfare sia le richieste funzionali, sia i requisiti interni. Infine, nonostante sia stata fatta un'accurata fase di collaudo e di testing, è impossibile essere certi della mancanza di difetti nell'applicazione. Per tal motivo la ricerca di ulteriori problemi relativi alle funzionalità dell'applicazione è demandata agli operatori della ONLUS, in quanto, essendo i principali utilizzatori dell'applicazione possono riscontrare errori appartenenti a casi di test non presi in considerazione o non riscontrati dall'oracolo umano utilizzato in tale fase; a supporto degli operatori è stato inserito nel software una form per richiedere assistenza (tabella 7) e segnalare guasti, e il programma viene eseguito all'interno di un'istruzione **try** per evitare un imprevisto errore (tabella 8).

```
var FM_resu = DialogResult.Retry;
var FL_resu = DialogResult.OK;

if (DB.cDB.ConnessioneEffettuata)
        while (FM_resu == DialogResult.Retry && FL_resu == DialogResult.OK)
                using (var login = new UI.fLogin())
                        try
                        {
                                FL_resu = login.ShowDialog();

                                if (FL_resu == DialogResult.OK)
                                        using (var main = new UI.fMain())
                                                FM_resu = main.ShowDialog();
                        }
                        catch (Exception ex)
                        {
                                FM_resu = DialogResult.Abort;
                                GB.cGB.MsgBox(ConstStringaErroreInaspettato, MessageBoxIcon.Error);

                                using (var erD = new UI.fSegnalaUnProblema(ex))
                                        erD.ShowDialog();
                        }
```

*Tabella 7: Dettaglio della classe program.cs*

```
sealed partial class fSegnalaUnProblema : Base.fBase
{
        private Exception errore;

        internal fSegnalaUnProblema()
        {
                InitializeComponent();
                PossoEssereFullScreen = true;
        }

        internal fSegnalaUnProblema(Exception ex) : this()
        {
                errore = ex;
        }

        private void bInvia_Click(object sender, EventArgs z)
        {
                if (cGB.MsgBox(
                        "Invio?",
                        MessageBoxButtons.YesNo,
                        MessageBoxIcon.Question
                ) == DialogResult.Yes)
                        using (var e = new maionemikyWS.EmailSendingSoapClient())
                                cGB.MsgBox(
                                        e.MandaMail(
                                                "GAM#: Segnalazione errore - " +
                                                        DB.cMemDB.UtenteConnesso.Email,
                                                eProblema.Text + (errore?.Message ?? ""),
                                                "mikymaione@hotmail.it"
                                        ),
                                        MessageBoxIcon.Exclamation
                                );
        }
}
```

*Tabella 8: Form di segnalazione di problemi*

# 2. Sviluppi futuri

## Ampliamenti

L'applicazione, anche se risulta essere completa per ciò che riguarda i requisiti indicati in fase di analisi, ha una progettazione tale da permettere eventuali personalizzazioni del software, eventualmente richieste in futuro dalla ONLUS, in modo semplice senza dover modificare l'intera applicazione.

## Portabilità

Essendo stato scritto in C# senza l'uso di librerie esterne (tranne quelle relative al DB), la soluzione può tranquillamente essere compilata per i sistemi che supportano il Mono Framework. Infatti al momento della stesura di questo documento, la soluzione è stata aperta con MonoDevelop e compilata su Ubuntu 17.04, senza difficoltà.

## Web

Essendo la UI, logicalmente separata, si potrebbero creare delle WebForm con ASP.NET, implementando lo stesso tipo di interfacce wBase, wBaseDettaglio, wBaseRicerca, e portare il tutto sul web senza toccare la business logic.

# Bibliografia


Roger S. Pressman
*Principi di ingegneria del software*
McGraw-Hill Education, 2008

Ian Sommerville
*Ingegneria del software*
Pearson, 2007

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein
*Introduzione agli algoritmi e strutture dati*
McGraw-Hill Education, 2010

E. Kevin Kline, Daniel Kline
*SQL. Guida di riferimento*
Apogeo, 2003

Mickey Williams
*Programmare Visual C#.Net*
Mondadori Informatica, 2002

Samara Lynn
*Windows Server 2012. La guida*
Tecniche Nuove, 2013

Firebird Project
*Firebird 3.0 Quick Start Guide (*http://www.firebirdsql.org/en/documentation*)*
Consultata a settembre 2015

# Ringraziamenti 2.0

Il capitolo laurea non è stato temporalmente continuo, per questo mi trovo a ringraziarvi in modo particolare:

Babbo per aver pagato le tasse;

Mamma per aver ripetuto ogni giorno: “ma perché non studi?”;

Marika e Sba mie compagne di avventure;

Amelia, Emiliana, Pucio e Giulia per il supporto e i consigli, fatti miei, ma mai seguiti;

Mario, Fede, Nadia, Giorgio, Giovanni e Mariapina per esserci sempre stati;

Alessandro, Carmen e Zaza per gli anni mitici;

Ciro per avermi trovato sempre lavoro, altrimenti non me la sarei potuta permettere;

Morosin per avermi convinto con “oggi si laureano tutti!”;

Elle per le farfalle ai funghi;

I ragazzi della Pianetasoft s.r.l. per aver sopportato la mia rabbia;

I ragazzi della corsia 4 del Circolo Canottieri Napoli per aver sopportato la mia rabbia;

Tutti gli altri, anche se non citati, siete nel mio cuore.

Dedico questa laurea a mio fratello augurandogli di chiudere la sua; boia chi molla!

Grazie a tutti, lo spettacolo continua...