

Università degli Studi di Salerno

# Progetto di Ingegneria, gestione ed evoluzione del software

Domenico Taffuri  
(a.a.2018/2019)

Il sistema su cui intervenire è un gestionale per la società “SoftEngine 2016” che gestisce la rete idrica. Analizzando la documentazione del sistema, in particolare il documento dei requisiti, possiamo risalire alle funzionalità che esso soffre, che sono le seguenti:

- Creazione di un’ingiunzione
- Emissione di un’ingiunzione
- Ricerca di un’ingiunzione
- Modifica o aggiornamento di un’ingiunzione
- Cancellazione di un’ingiunzione
- Sospensione di un’ingiunzione
- Lettura dei contatori (da parte di un tecnico)

Si vogliono aggiungere le seguenti funzionalità al sistema :

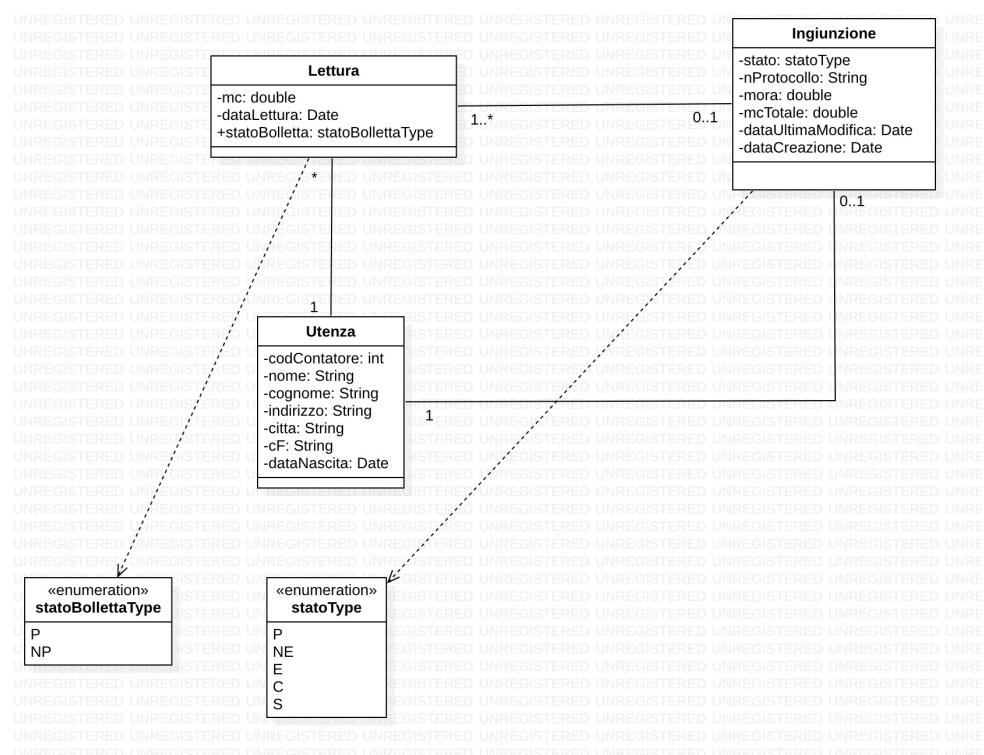
- Il cliente ha la possibilità di effettuare un’autolettura del proprio contatore. Nel caso in cui la lettura del tecnico e l’autolettura del cliente dovessero risultare diverse, la lettura viene annullata.
- Il cliente può visualizzare la propria situazione riguardo i pagamenti.

## DATA REVERSE ENGINEERING

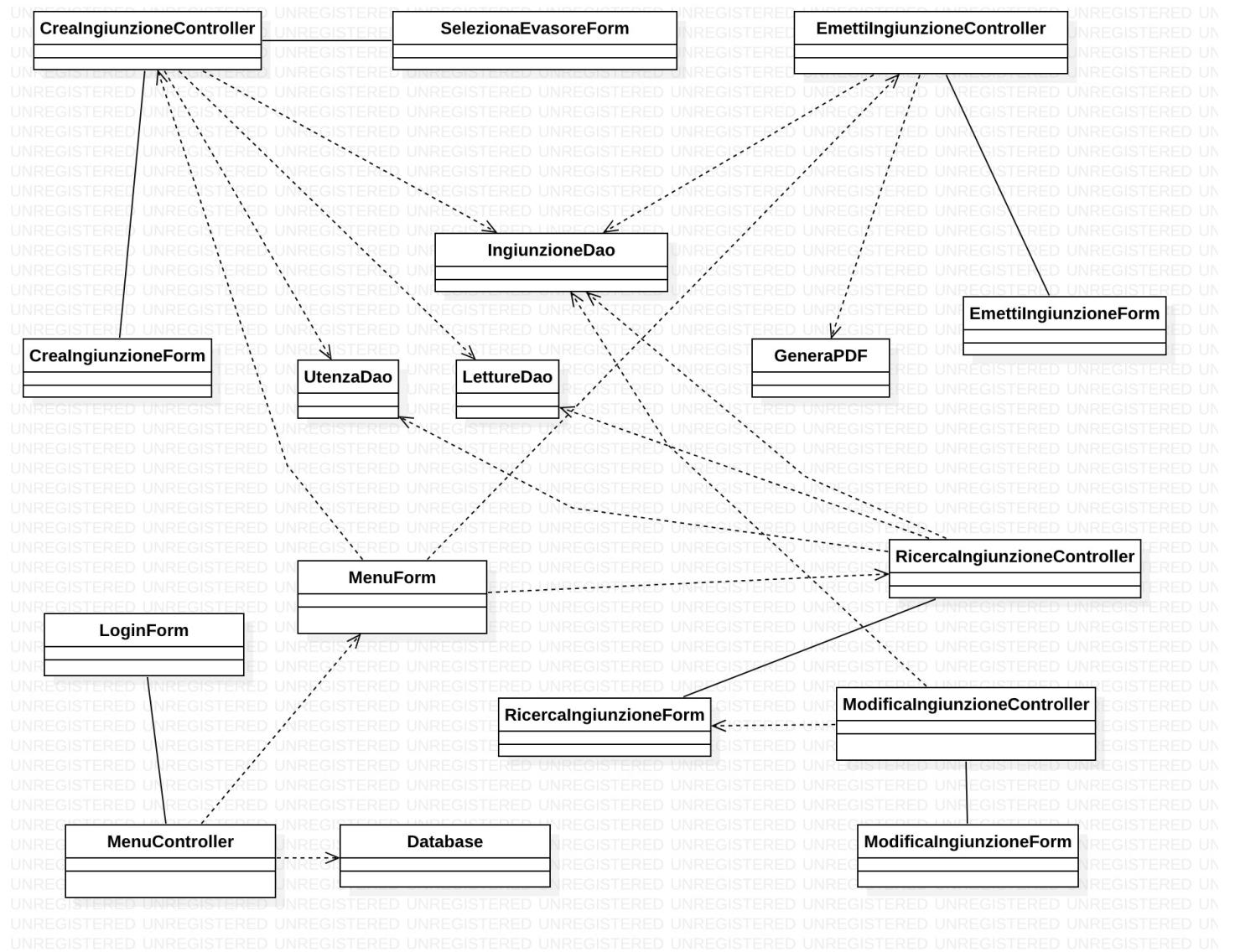
Il primo passo nell'analisi del sistema è quello di astrarre dal codice lo schema concettuale del database. Analizzando il codice sorgente, lo schema concettuale che ne deriva è il seguente :

- **Utenza**(idUtenza,codContatore, nome, cognome, indirizzo, citta, cF , dataNascita, idIngiunzione)
- **Lettura**(idLettura, mc, dataLettura, statoBolletta, idIngiunzione, i dUtenza)
- **Ingiunzione**(idIngiunzione, importo, stato, nProtocollo, mora, mcT otale, DataCreazione, dataUltimaModifica, idUtenza)

Dopo essere risaliti allo schema concettuale, rappresentiamo le informazioni ottenute tramite uno schema logico, in particolare un class diagram.



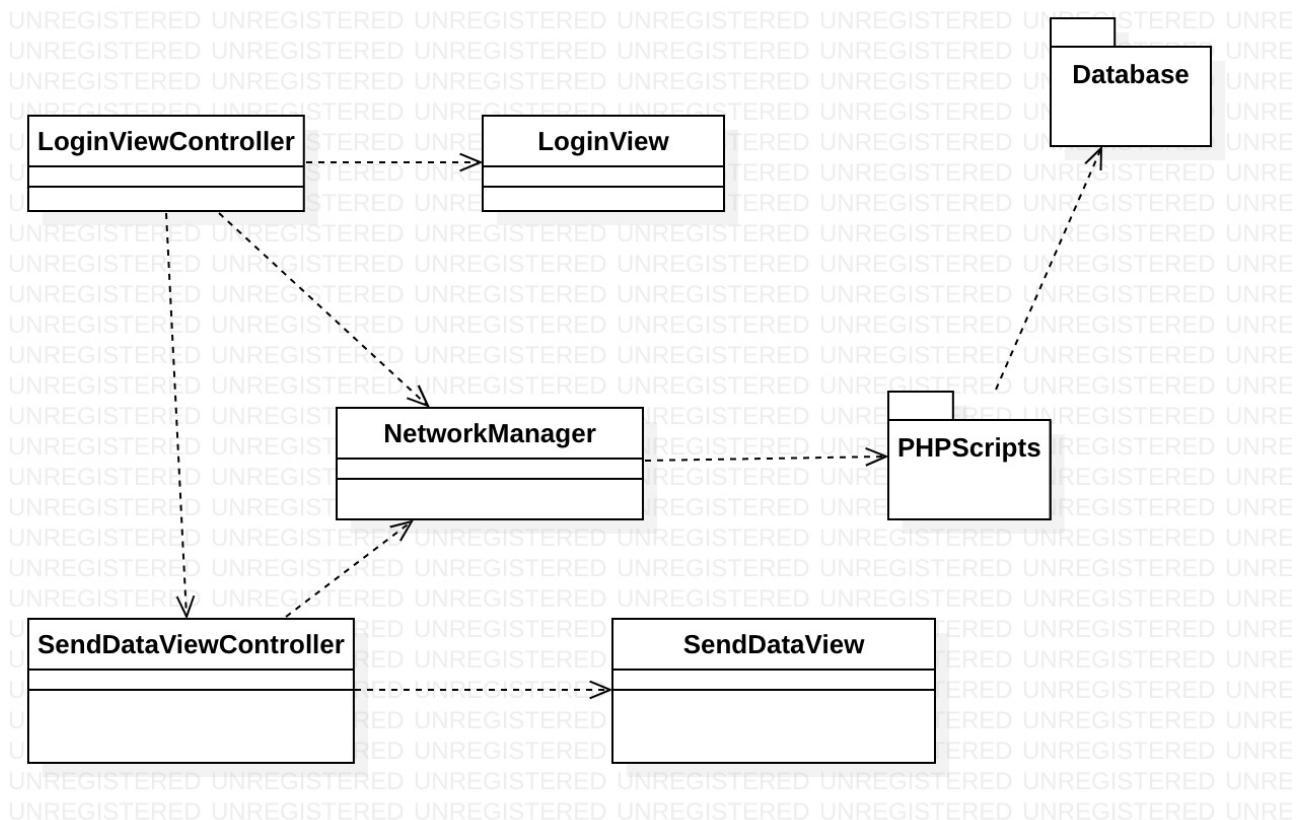
Il sistema è composto da due applicativi, uno desktop ed uno mobile. Analizzando il codice sorgente dell'applicativo desktop, possiamo costruire il seguente diagramma rappresentante relazioni tra le varie classi del sistema.



Per evitare troppa confusione all'interno del grafico, è stata omessa la dipendenza di tutte le classi “DAO” verso il database. Ogni classe “DAO”, però, ha come unica dipendenza il database, in quanto tutte

le operazioni all'interno di una classe di questo tipo sono operazione di interrogazione del database.

La stessa operazione può essere svolta analizzando il codice dell'applicazione mobile.



Analizzando il documento dei requisiti, avendo già compiuto il passo di astrazione dal codice al design, possiamo generare il grafo di delle dipendenze del sistema, in modo da avere una visione chiara e completa del sistema nel suo insieme, per poi andare ad analizzare le change requests e vedere su quali parti del codice vanno ad impattare.

Legenda:

R : Requisiti

C : Codice

R1: Creazione ingiunzione

R2: Emissione ingiunzione

R3: Ricerca ingiunzione

R4: Modifica o aggiornamento ingiunzione

R5: Cancellazione ingiunzione

R6: Sospensione ingiunzione

R7: Lettura dei contatori (tecnico)

C1 : tabella “Lettura”

C2 : tabella “Ingiunzione”

C3 : tabella “Utenza”

C4 : enum “statoBollettaType”

C5 : enum “statoType”

C6 : classe “CreaIngiunzioneController” (Java)

C7 : classe “SelezionaEvasoreForm” (Java)

C8 : classe “EmettiIngiunzioneController” (Java)

C9 : classe “IngiunzioneDao” (Java)

C10 : classe “CreaIngiunzioneForm” (Java)

C11 : classe “UtenzaDao” (Java)

C12 : classe “LettureDao” (Java)

C13 : classe “GeneraPDF” (Java)

C14 : classe “EmettiIngiunzioneForm” (Java)

C15 : classe “LoginForm” (Java)

C16 : classe “MenuForm” (Java)

C17 : classe “RicercaIngiunzioneController” (Java)

C18 : classe “RicercaIngiunzioneForm” (Java)

C19 : classe “ModificaIngiunzioneController” (Java)

C20 : classe “MenuController” (Java)

C21 : classe “Database” (Java)

C22 : classe “ModificaIngiunzioneForm” (Java)

C23 : classe “LoginViewController” (Swift)

C24 : classe “LoginView” (Swift)

C25 : classe “NetworkManager” (Swift)

C26 : classe “SendDataViewController”

C27 : classe “SendDataView”

C28 : script PHP “checkCredential.php”

C29 : script PHP “checkCodContatore.php”

C30 : script PHP “sendData.php”

## GRAFO DELLE DIPENDENZE

$\text{out}(R1) = \{C7, C9, C11, C12, C10\}$

$\text{out}(R2) = \{C14, C9, C13\}$

$\text{out}(R3) = \{C18, C12, C11, C9\}$

$\text{out}(R4) = \{C18, C22, C9\}$

$\text{out}(R5) = \{C18, C22, C9\}$

$\text{out}(R6) = \{C18, C22, C9\}$

$\text{out}(R7) = \{C23, C24, C25, C26, C27, C28, C29, C30\}$

$\text{out}(C1) = \{C2, C3, C4\}$

$\text{out}(C2) = \{C1, C3, C5\}$

$\text{out}(C3) = \{C1, C2\}$

$\text{out}(C6) = \{C7, C10, C9, C11, C12\}$

$\text{out}(C7) = \{C6\}$

$\text{out}(C8) = \{C13, C14, C9\}$

$\text{out}(C9) = \{C2\}$

$\text{out}(C10) = \{C6\}$

$\text{out}(C11) = \{C3\}$

```

out(C12) = {C1}
out(C14) = {C8}
out(C15) = {C20}
out(C16) = {C17,C6,C8}
out(C17) = {C18,C11,C12,C9}
out(C18) = {C17}
out(C19) = {C18,C22,C9}
out(C20) = {C15,C16}
out(C22) = {C19}
out(C23) = {C24,C25,C26}
out(C25) = {C28,C29,C30}
out(C26) = {C27}
out(C29) = {C3}
out(C30) = {C1,C3}

```

Andiamo ad analizzare una delle due change request richiesta dal committente :

- Il cliente ha la possibilità di effettuare un'autolettura del proprio contatore. Nel caso in cui la lettura del tecnico e l'autolettura del cliente dovessero risultare diverse, la lettura viene annullata.

Aggiorniamo il documento dei requisiti con questa nuova funzionalità da aggiungere al sistema, etichettandola con R8.

L'idea per implementare questa nuova funzionalità è la seguente: il cliente utilizza la stessa applicazione mobile utilizzata dal tecnico, nella quale può loggarsi tramite username e password. Ogni qual volta una lettura viene inserita, viene controllata la presenza della lettura effettuata dalla controparte(cliente oppure tecnico) nello stesso mese : nel caso in cui vengano rilevate entrambe le letture, si controlla che i metri cubi rilevati corrispondano e, in caso contrario, le letture passa

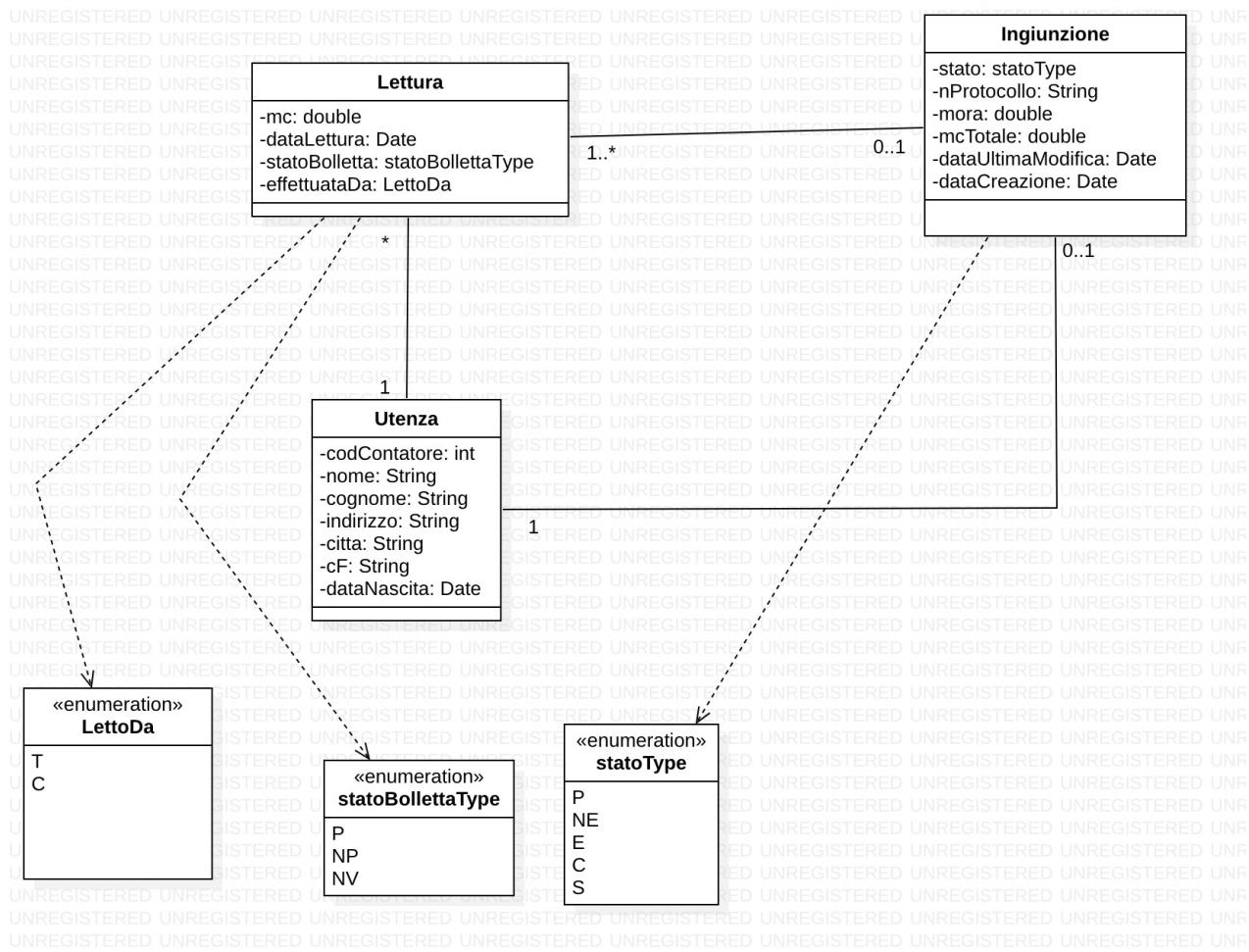
in uno stato NV (non valida). Nel caso in cui , invece, le letture corrispondono, ai fini del pagamento fa fede soltanto la lettura del tecnico mentre quella del cliente viene eliminata per non avere duplicati all'interno del database.

## IMPLEMENTAZIONE DESIGN

Iniziamo ad implementare le modifiche necessarie sullo schema del database. Di seguito sono descritte tutte le aggiunte/modifiche effettuate allo schema del database, con relativa spiegazione e con la costruzione passo passo dello starting impact set.

- Creato la enum “LettoDa” nel database. Questo nuovo elemento viene codificato con il codice C32. C32 viene aggiunta allo SIS.
- Aggiunto alla tabella “lettura” il campo “effettuataDa” di tipo “LettoDa”. C1 viene aggiunta allo SIS. Questo campo conserverà l’informazione della modifica: fatta da tecnico oppure da cliente.
- Aggiunta all’enum “StatoBollettaType” la voce “NV” che sta per “Non Valida”. C4 viene aggiunta allo SIS. Questa voce verrà attivata quando due letture effettuate da tecnico e cliente non concordano sull’ammontare di metri cubi della lettura.

Il class diagram di analisi del database aggiornato alle nuove modifiche è il seguente :



Lo starting impact set (SIS) relativo a questa prima serie di modifiche è : {C32,C1,C4}

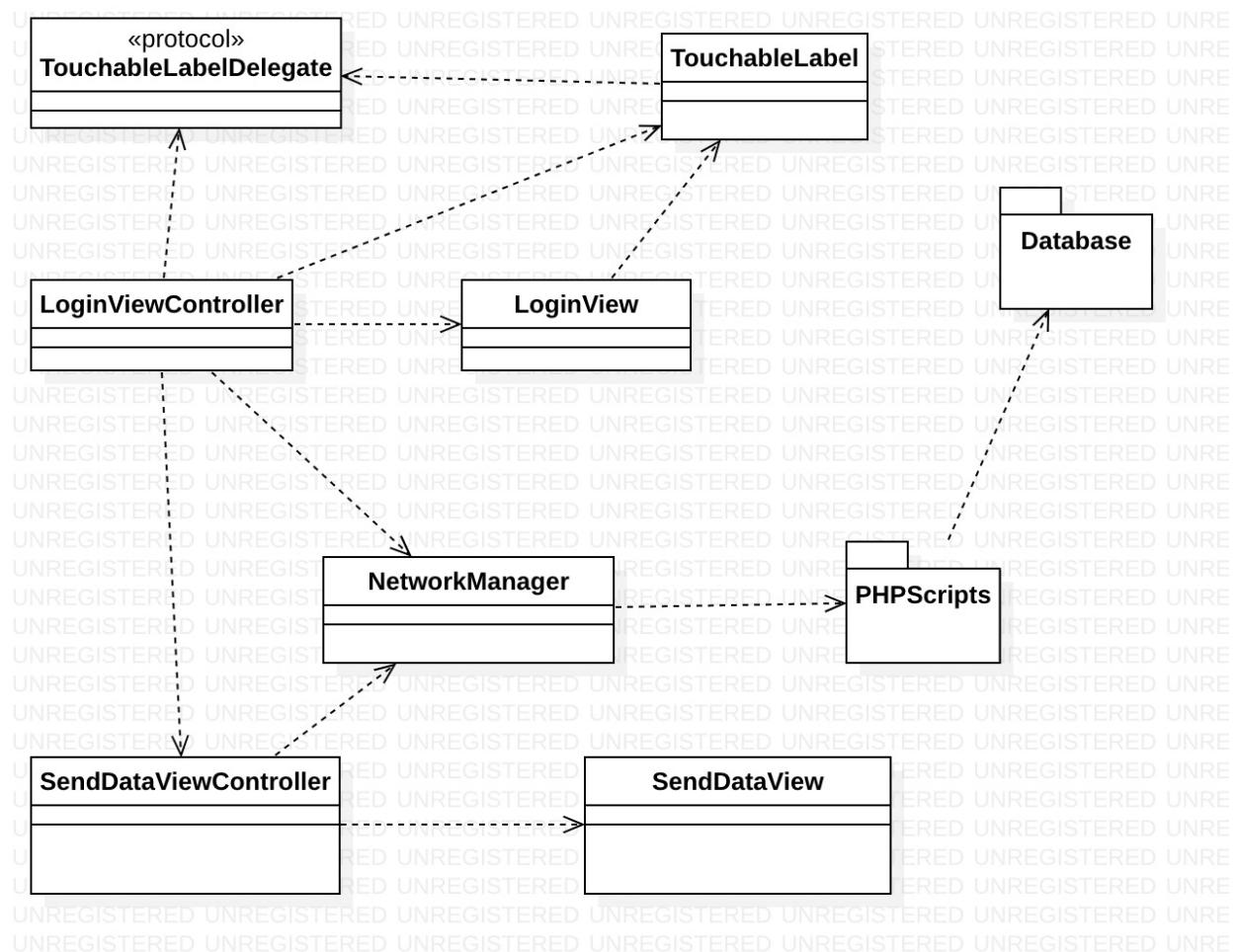
Passiamo ora ad implementare le modifiche necessarie all'applicativo mobile. Di seguito sono descritte tutte le aggiunte/modifiche effettuate allo schema del database, con relativa spiegazione e con la costruzione passo passo dello starting impact set.

- Aggiunta alla classe “LoginView” una label che ti permette di scegliere se loggarti come tecnico o come cliente. C24 viene aggiunta allo SIS.
- Creato la classe “TouchableLabel”, codificata con il nome C33. C33 viene aggiunta allo SIS. Questa classe eredita le funzionalità

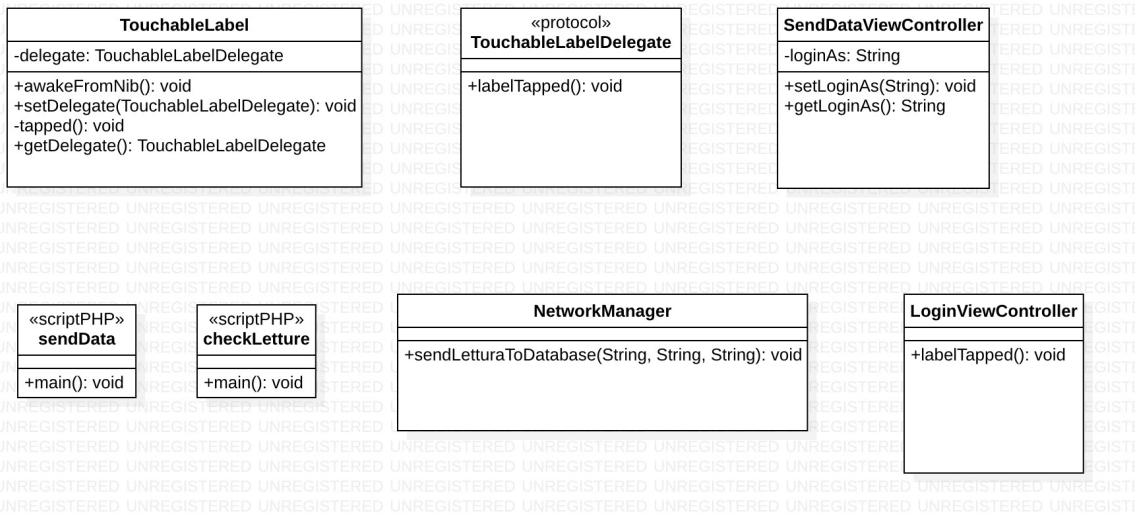
della superclasse “UILabel” delle api di Swift, introducendo una funzionalità in più : la label risponde con un evento al tocco dell’utente.

- Creato il protocollo “TouchableLabelDelegate”, codificato con il nome C34. C34 viene aggiunto allo SIS. Le classi che implementano questo protocollo possono intercettare il messaggio lanciato da un oggetto dalla classe “TouchableLabel” in risposta alla pressione da parte dell’utente.
- Aggiunta l’implementazione del protocollo “TouchableLabelDelegate” alla classe “LoginViewController”. Di conseguenza , alla classe, viene aggiunto il metodo “labelTapped() : void” che è obbligatorio per il suddetto protocollo. C23 viene aggiunta allo SIS.
- Aggiunta la variabile “loginAs” in “SendDataViewController” per tenere traccia dell’utente, se è loggato come tecnico oppure come cliente. C26 viene aggiunto allo SIS.
- Modificato lo script PHP “sendData.php” in cui è stato aggiunto il nuovo campo “effettuatoDa” nella query. C30 è stato aggiunto allo SIS.
- Creato lo script PHP “checkLetture.php” che ad ogni inserimento di una nuova lettura va a controllare se è presente la controparte ed effettua le operazioni richieste. Il nuovo script è codificato con il nome C35. C35 viene aggiunto allo SIS.
- Modificata la classe “NetworkManager”. C25 è stata aggiunta all’AIS. Aggiunta la riga 56 al metodo “sendLetturaToDatabase”. La nuova istruzione lancia lo script C35 dopo aver inserito una lettura all’interno del database. Lo script, come visto in precedenza, controlla che per il mese corrente entrambe le lettura del tecnico e del cliente, se presenti, concordino sui metri cubi e, in caso contrario, vengono effettuate le operazioni richieste dai requisiti.

Il nuovo class diagram di analisi dell'applicativo iOS è il seguente :



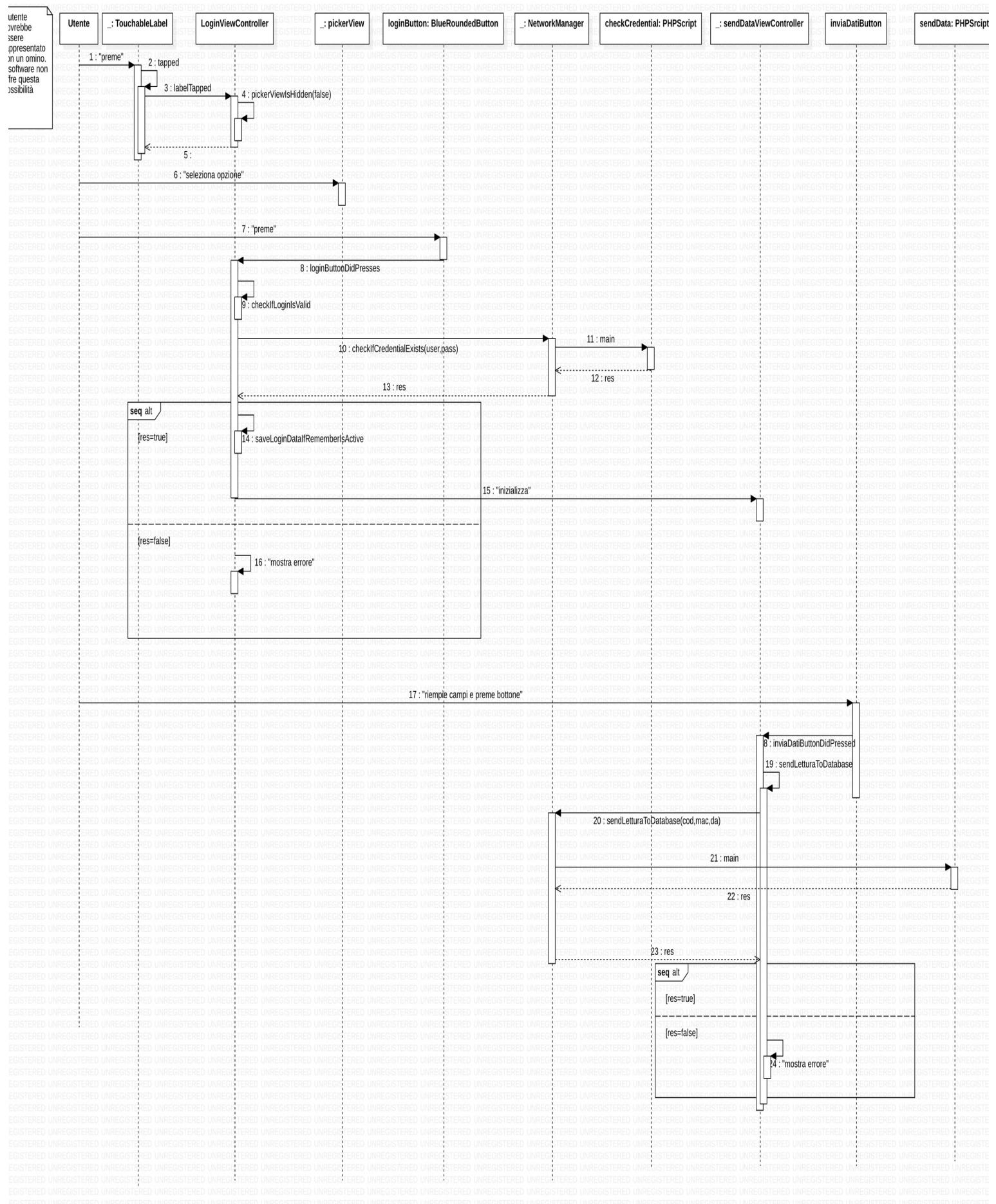
Di seguito invece, vengono mostrare solo le classi aggiunte/modificate con solo i metodi che sono stati aggiunti/cambiati. Si è preferito metterli in un diagramma separato e non nel diagramma precedente per evitare confusione e non confonderli con i metodi che , invece, non hanno subito modifiche.



L'insieme SIS risultante è il seguente :

$$\{C32, C1, C4, C24, C33, C34, C23, C26, C35, C30, C25\}$$

La modifica, quindi, ha impattato in maniera diretta su 11 componenti del sistema. Prima di andare ad analizzare gli effetti a catena, quindi costruire il CIS con le componenti impattate in maniera indiretta, andiamo a costruire il sequence diagram relativo al caso d'uso appena progettato.



$$\text{SIS} = \{\text{C32}, \text{C1}, \text{C4}, \text{C24}, \text{C33}, \text{C34}, \text{C23}, \text{C26}, \text{C35}, \text{C30}, \text{C25}\}$$

Tenendo conto del SIS, andiamo ora a stimare un possibile insieme CIS delle componenti che saranno impattate tramite effetto a catena

$$\text{CIS} = \{\text{C3}, \text{C12}, \text{C2}, \text{C6}, \text{C11}, \text{C17}, \text{C3}, \text{C7}, \text{C9}, \text{C10}, \text{C16}, \text{C18}, \text{C8}, \text{C19}, \text{C14}, \}$$

Questo è quello che è possibile desumere osservando soltanto il grafo delle dipendenze. Tenendo conto della natura delle modifiche fatte ad ogni componente è tenendo conto del codice sorgente e dei documenti, proviamo a restringere il CIS per cercare di renderlo più preciso.

$$\text{CIS} = \{\}$$

Dopo un'analisi più approfondita del codice sorgente e dei documenti, credo che possiamo stimare in maniera abbastanza sicura il CIS vuoto. Questo perché le uniche modifiche che potrebbero impattare l'applicativo desktop Java, sono quelle relative al database. Analizzando le modifiche fatte al database ed analizzando il codice delle componenti dell'applicativo Java presenti nel CIS stimato osservando solo il grafo delle dipendenze, possiamo ragionevolmente concludere che nessuna di esse va ad impattare sulle componenti e sulle funzionalità. Procediamo ora con l'implementazione della funzionalità, andando nel frattempo a costruire l'AIS e il DIS.

$$\text{AIS} = \{\text{C32}, \text{C1}, \text{C4}, \text{C24}, \text{C33}, \text{C34}, \text{C23}, \text{C26}, \text{C35}, \text{C30}, \text{C25}\}$$

$$\text{DIS} = \{\}$$

## TESTING

In questa prima fase andiamo a testare il sistema tramite test di unità partendo da tutti i metodi che hanno subito delle modifiche durante la progettazione della change request.

### **Componente C32 : enum “LettoDa”**

L’enum in questione non può essere testata da sola. Viene usata come attributo della tabella “Lettura”, i cui metodi sql si suppone essere corretti.

### **Componente C1 : tabella “Lettura”**

Come per l’enum precedente, diamo per scontato che i metodi sql di manipolazione della tabella siano corretti. D’ora in avanti vengono omesse questo tipo di componenti.

### **Componente C24 : classe “LoginView”**

L’unica modifica fatta al componente è quello di aver aggiunto una label ed un picker view. Il componente è stato testato tramite la classe di test “LoginViewTest”, in particolare con il metodo “testLabelAndPicker”, che esegue un test dell’ UI.

### **Componente C33 : classe “TouchableLabel”**

metodo “tapped(UIGestureRecognizer)”: il metodo, in base allo stato dell’attributo delegate, invoca il metodo “labelTapped” dell’attributo stesso. Testiamo con strategia WECT in quanto i valori possibili possono essere solo 2.

Classi di equivalenza per “delegate”:

CE1 = oggetto di tipo TouchableLabelDelegate

CE2 = oggetto nullo

Stato “delegate”	Classe di eq	Risultato atteso	Risultato ottenuto
null	CE2	Metodo non invocato	ok
Oggetto TouchableLabelDelegte	CE1	Metodo invocato	ok

La classe creata ed utilizzata per il test è la classe “TouchableLabelTest”.

### **Componente C23 : classe “LoginViewController”**

Il metodo da testare per la classe è il metodo “labelTapped”. Quando il metodo viene invocato, il sistema deve mostrare a schermo la pickerView di selezione della modalità di login. Il metodo è stato testato tramite testing UI insieme alla componente C24.

### **Componente C30 : script PHP “SendData”**

Gli argomenti che lo script prende in input sono 4. Andiamo a creare delle classi di equivalenza per ognuno di essi.

codContatore: amesso intero positivo esistente nel database. Quando lo script viene invocato, siamo sicuri che il codContatore esiste nel database in quanto vi è stato un controllo precedente.

CE1 = codContattore > 0 intero esistente nel database

CE2 = codContattore < 0 intero

CE3 = codContattore decimale

mc : ammesso qualunque reale positivo

M1 = mc > 0 intero

M2 = mc < 0 intero

M3 = mc > 0 reale

M4 = mc < 0 reale

dataLettura : ammessa qualsiasi data in formato YYYY-MM-DD

D1 = data in formato YYYY-MM-DD

D2 = data in formato DD-MM-YYYY

D3 = data in formato DD-MM-YY

D4 = data in formato DD/MM/YY

effettuatoDa : ammessi esclusivamente T e C come valori

EF1 = T

EF2 = C

EF3 = qualunque lettera

EF4 = qualunque numero

I parametri di input non sono collegati tra loro, quindi non c'è bisogno di testare qualunque combinazione dei 4 parametri, per cui possiamo usare una strategia WECT.

codContatore	mc	dataLettura	effettuatoDa	Output atteso	Output ottenuto
1001 : C1	1290 : M1	2019-02-05:D1	T:EF1	Inserimento effettuato	ok
-2009 : C2	-450 : M2	05-02-2019 : D2	C : EF2	errore	ok
20,39 : C3	238,90 : M3	05-02-19 : D3	H : EF3	errore	ok
1001 : C1	-289,90: M4	05/02/19 : D4	10 : EF4	errore	ok

### Componente C35 : script PHP “checkLetture”

Lo script non dipende solo dai parametri ma anche dallo stato del database. Cambia operazione da effettuare in base alle lettura presenti all'interno dello stesso. Anche questo, quindi, è un parametro da tenere in considerazione per testare la componente. I parametri in ingresso sono 4, gli stessi della componente precedente.

codContatore: amesso intero positivo esistente nel database. Quando lo script viene invocato, siamo sicuri che il codContatore esiste nel database in quanto vi è stato un controllo precedente.

CE1 = codContattore > 0 intero esistente nel database

CE2 = codContatore < 0 intero

CE3 = codContatore decimale

mc : ammesso qualunque reale positivo

M1 = mc > 0 intero

M2 = mc < 0 intero

M3 = mc > 0 reale

M4 = mc < 0 reale

dataLettura : ammessa qualsiasi data in formato YYYY-MM-DD

D1 = data in formato YYYY-MM-DD

D2 = data in formato DD-MM-YYYY

D3 = data in formato DD-MM-YY

D4 = data in formato DD/MM/YY

effettuatoDa : ammessi esclusivamente T e C come valori

EF1 = T

EF2 = C

EF3 = qualunque lettera

EF4 = qualunque numero

Aggiungiamo due classi di equivalenza che rappresentano lo stato del database.

DB1 = lettura della controparte presente con stato NP e mc uguale

DB2 = lettura della controparte presente con stato NP e mc diverso

DB3 = lettura della controparte presente con stato P e mc uguale

DB4 = lettura della controparte presente con stato P e mc diverso

DB5 = lettura della controparte non presente

Utilizziamo anche per questa componente una strategia di tipo WECT

codContato re	mc	dataLettu ra	effettuato	statoD B	Risultat o atteso	Risulta to ottenut o
1002 : C1	12:M 1	2019-02- 05 : D1	T:EF1	DB1	Cancella lettura cliente	ok
-76 : C2	-12:M2	05-02- 2019 : D2	C : EF2	DB5	errore	ok
20,39:C3	- 20,2:M 3	05-02-19 : D3	H:EF3	DB5	errore	ok
1001 : C1	289,90: M4	05/02/19 : D4	10 : EF4	DB5	errore	ok
1001:C1	10:M 1	2019-02- 05 : D1	T:EF1	DB2	Entram be le lettura passano in NV	ok
1001:C1	10:M 1	2019-02- 05 : D1	C:EF2	DB3	Non modifica db	ok
1001:C1	10:M 1	2019-02- 05 : D1	C:EF2	DB4	Non modifica db	ok

## TEST DI REGRESSIONE

Andiamo ora ad effettuare dei test sul tutto il sistema, in particolare sulle componenti che non sono state modificate, per verificare che le modifiche effettuate non abbiano impattato in maniera negativa sul sistema stesso. Il test sarà effettuato tenendo conto del documento “PIANO DI TEST CASES sistema vecchio.pdf” che è stato ereditato dal vecchio sistema.

### Ricerca ingiunzione

Test effettuato usando come casi di test la tabella presente nel documento. Non sono stati riscontrati errori. Durante l'inserimento dei dati si è riscontrato un piccolo bug : il bottone ricerca non veniva abilitato quando si riempiva la casella idIngiunzione, rendendo impossibile la ricerca inserendo unicamente questo parametro. Il problema è stato risolto.

### Modifica ingiunzione

Test effettuato usando come casi di test la tabella presente nel documento.

Problema riscontrato con il caso di test 4 della tabella. Alla pressione di “calcolaImporto” non viene mostrato nessun messaggio che invita l'utente a cambiare mora in quanto il dato inserito non è valido. Risolto modificando il metodo “getImporto” della classe “ModificaIngiunzioneController”, righe 72 e 73.

Il documento di test del vecchio sistema tiene presenta piani di test solo per queste due funzionalità.

### Emetti Ingiunzione

Testata con varie ingiunzioni presenti all'interno del database senza riscontrare errori.

### Cancella Ingiunzione

Testata con varie ingiunzioni presenti all'interno del database senza riscontrare errori.

### Sospendi Ingiunzione

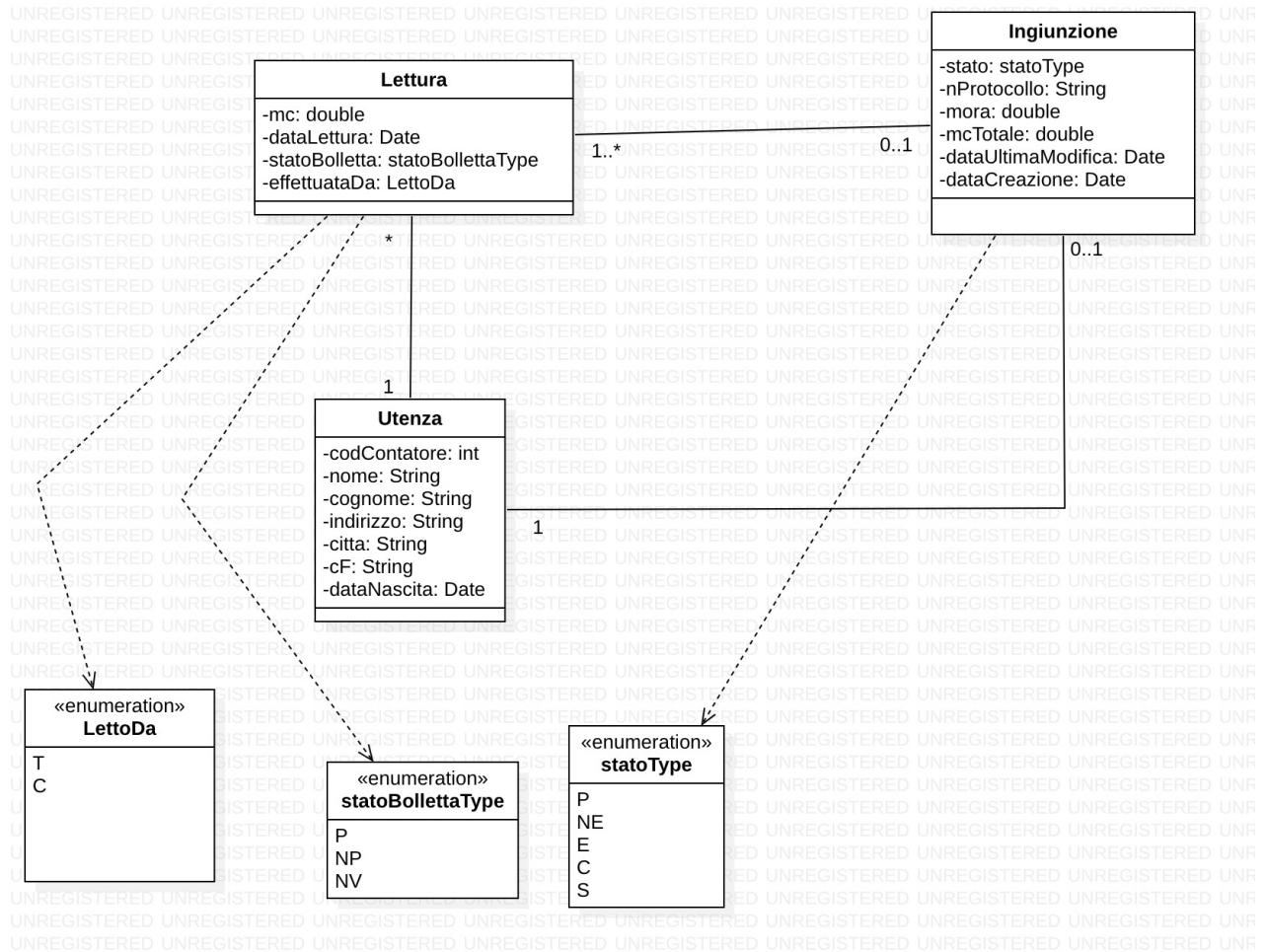
Testata con varie ingiunzioni presenti all'interno del database senza riscontrare errori.

## PROGETTAZIONE DELLA SECONDA MODIFICA

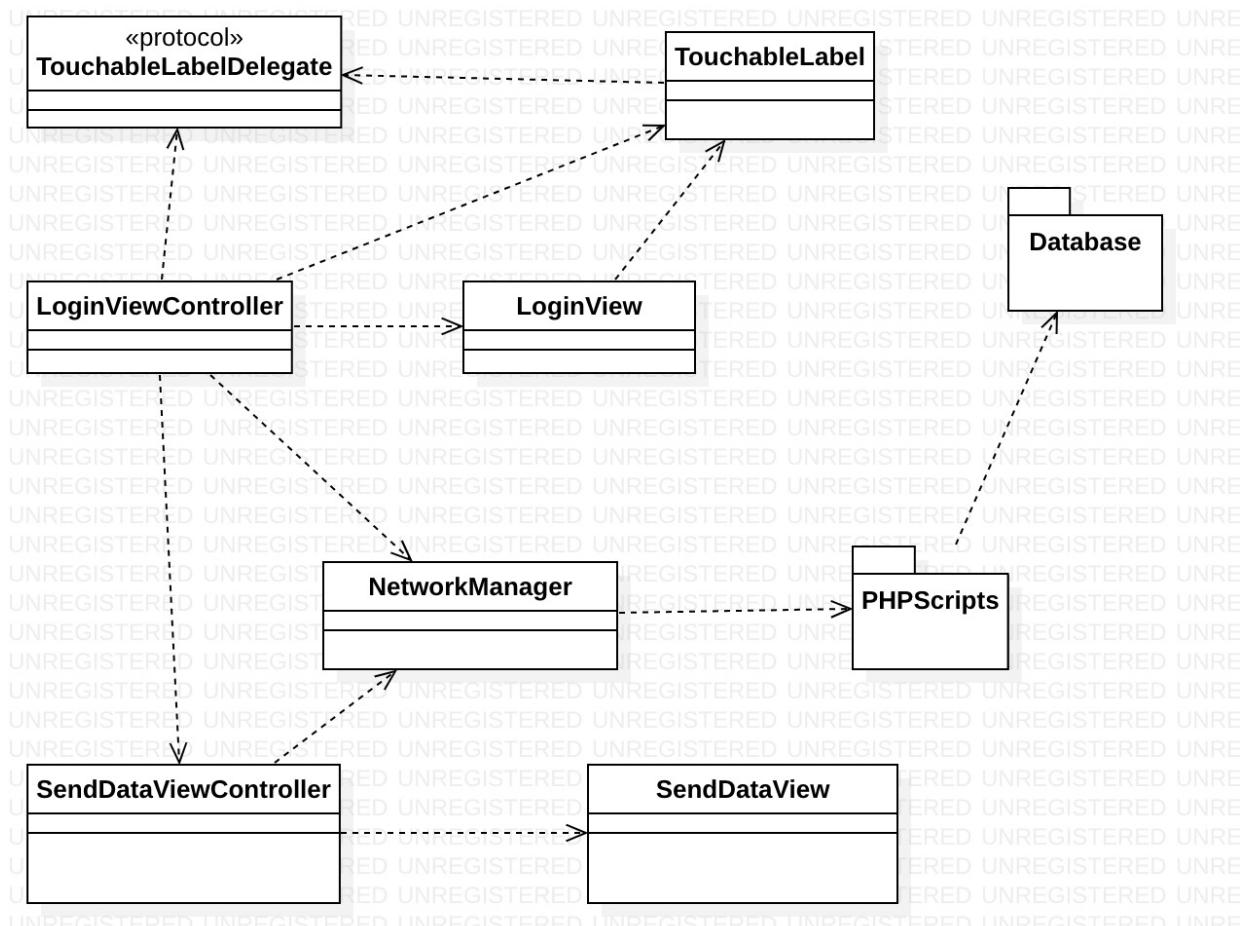
- Il cliente può visualizzare la propria situazione riguardo i pagamenti.

Dopo la prima modifica effettuata, ci ritroviamo con i seguenti class diagram delle componenti del sistema.

### Class diagram del database



## Class diagram applicativo mobile iOS



Il nuovo grafo delle dipendenze è il seguente :

Legenda:

R : Requisiti

C : Codice

R1: Creazione ingiunzione

R2: Emissione ingiunzione

R3: Ricerca ingiunzione

R4: Modifica o aggiornamento ingiunzione

R5: Cancellazione ingiunzione

R6: Sospensione ingiunzione

R7: Lettura dei contatori (tecnico)

R8: Lettura del contatore(cliente)

C1 : tabella “Lettura”

C2 : tabella “Ingiunzione”

C3 : tabella “Utenza”

C4 : enum “statoBollettaType”

C5 : enum “statoType”

C6 : classe “CreaIngiunzioneController” (Java)

C7 : classe “SelezionaEvasoreForm” (Java)

C8 : classe “EmettiIngiunzioneController” (Java)

C9 : classe “IngiunzioneDao” (Java)

C10 : classe “CreaIngiunzioneForm” (Java)

C11 : classe “UtenzaDao” (Java)

C12 : classe “LettureDao” (Java)

C13 : classe “GeneraPDF” (Java)

C14 : classe “EmettiIngiunzioneForm” (Java)

C15 : classe “LoginForm” (Java)

- C16 : classe “MenuForm” (Java)  
 C17 : classe “RicercaIngiunzioneController” (Java)  
 C18 : classe “RicercaIngiunzioneForm” (Java)  
 C19 : classe “ModificaIngiunzioneController” (Java)  
 C20 : classe “MenuController” (Java)  
 C21 : classe “Database” (Java)  
 C22 : classe “ModificaIngiunzioneForm” (Java)  
 C23 : classe “LoginViewController” (Swift)  
 C24 : classe “LoginView” (Swift)  
 C25 : classe “NetworkManager” (Swift)  
 C26 : classe “SendDataViewController”  
 C27 : classe “SendDataView”  
 C28 : script PHP “checkCredential.php”  
 C29 : script PHP “checkCodContatore.php”  
 C30 : script PHP “sendData.php”  
 C32 : enum “LettoDa”  
 C33 : classe “TouchableLabel”  
 C34 : protocol “TouchableLabelDelegate”  
 C35 : script PHP “checkLettura”  
 C36: tabella “Credenziali”  
 C37 : classe “VisualizzaPagamentiView”  
 C38 : classe “VisualizzaPagamentiViewController”

out(R1) = {C7,C9,C11,C12,C10}

out(R2) = {C14,C9,C13}

out(R3) = {C18,C12,C11,C9}

out(R4) = {C18,C22,C9}

out(R5) = {C18,C22,C9}

out(R6) = { C18,C22,C9}

out(R7) = {C23,C24,C25,C26,C27,C28,C29,C30}

out(R8) = {C23,C24,C25,C26,C27,C28,C29,C30,C35,C32,C33,C34}

out(C1) = {C2,C3,C4,C32}  
out(C2) = {C1,C3,C5}  
out(C3) = {C1,C2,C36}  
out(C6) = {C7,C10,C9,C11,C12}  
out(C7) = {C6}  
out(C8) = {C13,C14,C9}  
out(C9) = {C2}  
out(C10) = {C6}  
out(C11) = {C3}  
out(C12) = {C1}  
out(C14) = {C8}  
out(C15) = {C20}  
out(C16) = {C17,C6,C8}  
out(C17) = {C18,C11,C12,C9}  
out(C18) = {C17}  
out(C19) = {C18,C22,C9}  
out(C20) = {C15,C16}  
out(C22) = {C19}  
out(C23) = {C24,C25,C26,C33,C34}  
out(C25) = {C28,C29,C30}  
out(C26) = {C27}  
out(C28) = {C31}  
out(C29) = {C3}  
out(C30) = {C1,C3}  
out(C33) = {C34}  
out(C34) = {C1}  
out(C36) = {C3}

Codifichiamo la nuova funzionalità richiesta con R9. L'idea per implementare questa nuova funzionalità è la seguente : quando il cliente si logga nella sua area riservata dall'applicativo mobile, avrà la possibilità non solo di inserire la lettura, ma anche di visualizzare la propria situazione riguardo i pagamenti.

## IMPLEMENTAZIONE DESIGN

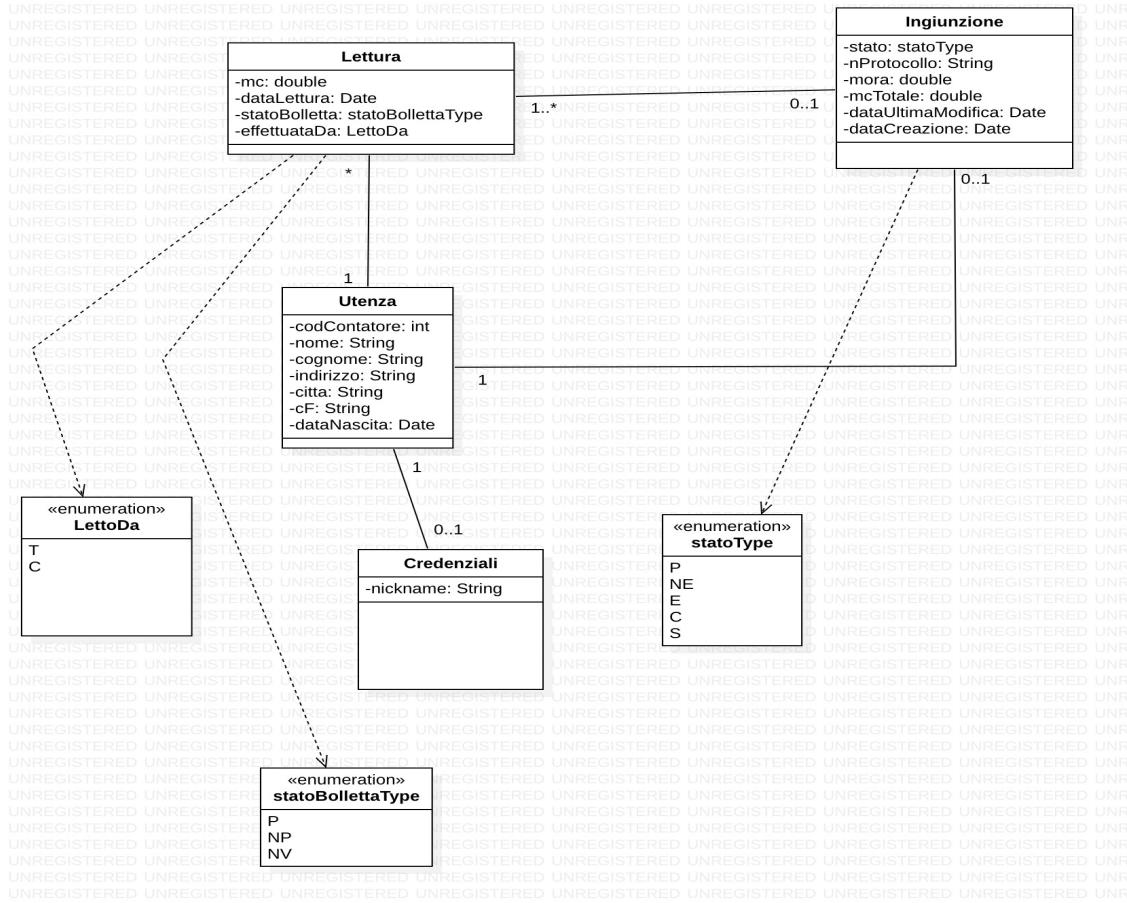
Come fatto per la precedente modifica,iniziamo ad implementare le modifiche necessarie sullo schema del database. Di seguito sono descritte tutte le aggiunte/modifiche effettuate allo schema del database, con relativa spiegazione e con la costruzione passo passo dello starting impact set.

- Creiamo una nuova tabella all'interno del database chiamata “Credenziali”, la quale associa ogni username registrato al corrispettivo idUtente. Ogni qual volta un nuovo utente effettuerà la registrazione, verrà aggiunta una riga alla tabella con nickname per accedere all'area riservata e idUtente. La tabella sarà codificata con C36. C36 è aggiunto all'insieme SIS.

Alla fine di questa prima serie di operazioni, l'insieme SIS è il seguente

$$\text{SIS} = \{\text{C36}\}$$

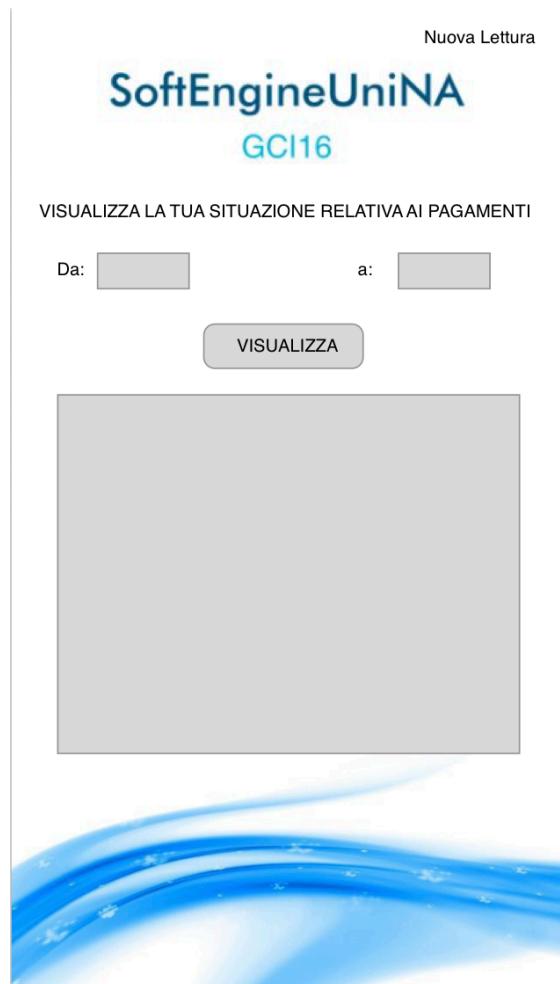
Il nuovo class diagram del database aggiornato con la nuova tabella è il seguente:



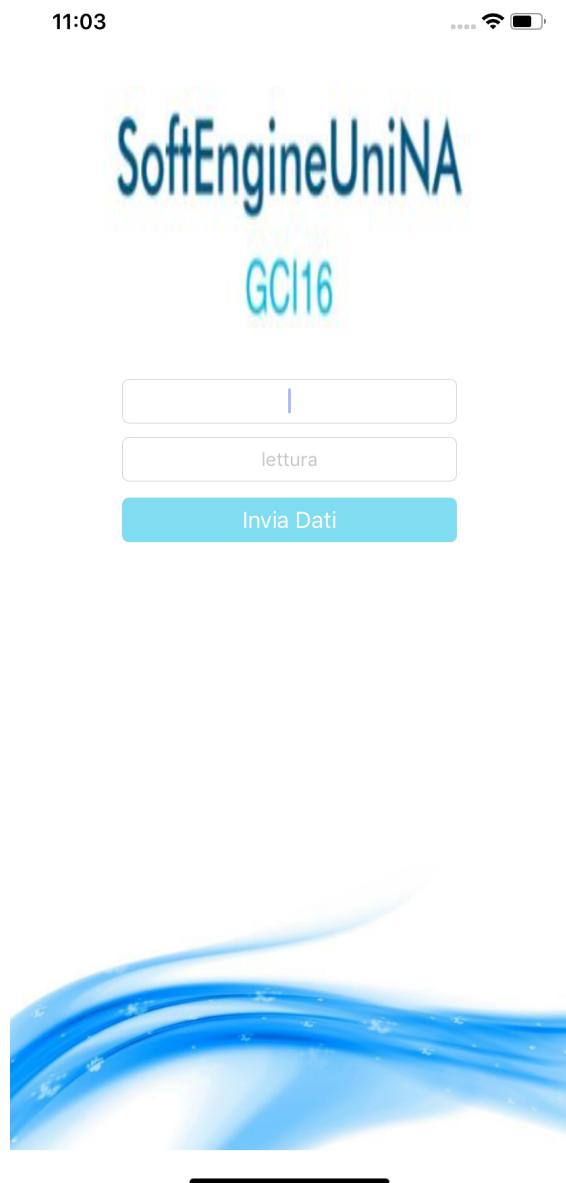
Aggiorniamo il grafo delle dipendenze con la nuova componente C36.

Prima di tutto, le seguenti schermate mostrano i mock up di come sarà l'applicativo una volta implementate le modifiche poiché le componenti che andremo ad aggiungere/modificare in questa fase dipendono dal tipo di UI che vogliamo dare all'applicativo.

Appena verrà effettuato il login come cliente, la prima schermata che sarà mostrata sarà la seguente (ovviamente i componenti a livello grafico non saranno uguali, questo è solo un mock up):



Il cliente potrà inserire il periodo e, dopo aver premuto visualizza, il sistema mostrerà la situazione relativa ai pagamenti, sia per quanto riguarda le letture sia per quanto riguarda eventuali ingiunzioni. Il cliente potrà aggiungere una nuova autolettura pignando il pulsante in alto a destra: a questo punto si aprirà la schermata dell'inserimento della lettura, come avveniva in precedenza:



Andiamo ora ad elencare le nuove componenti che serviranno e le eventuali modifiche da effettuare alle vecchie componenti per progettare con successo questa nuova modifica :

- Modificare la classe “LoginViewController”. In particolare va modificato il metodo “loginButtonDidPressed”, il quale deve lanciare un messaggio alla prossima schermata che va mostrata, che sarà diversa nel caso di login da parte del tecnico piuttosto che del cliente. C23 viene aggiunto all’insieme SIS.
- Implementare la classe “VisualizzaPagamentiView”. La view dovrà contenere i seguenti oggetti :
  - Un bottone in alto a destra per passare alla schermata di inserimento di una nuova lettura
  - Due labels con due rispettivi date picker per selezionare le date del periodo di riferimento
  - Una table view con rispettive celle per poter visualizzare i dati
  - Un bottone per lanciare la query e visualizzare i dati a schermo

La classe sarà codificata con C37. Viene aggiunta all’insieme SIS e al grafo delle dipendenze.

- Implementare la classe “VisualizzaPagamentiViewController” che si occuperà di catturare e gestire le interazioni dell’utente con la view “VisualizzaPagamentiView”. In particolare, dovrà fornire i seguenti metodi :
  - Il metodo “nuovaLetturaButtonDidPressed” per catturare la pressione del bottone “Nuova Lettura”. Il metodo dovrà lanciare a schermo la view “SendDataView”.
  - Implementare i protocolli dataSource e delegate per la gestione delle picker views. I metodi dovranno occuparsi di caricare i picker view da mostrare all’utente e memorizzare la scelta della data.
  - Il metodo “visualizzaButtonDidPressed” per catturare la pressione del bottone “Visualizza”. Il metodo manderà un

messaggio alla classe “NetworkManager” che si occuperà di effettuare la query e ritornare i dati.

- Implementare i protocolli dataSource e delegate per la table view che andrà a mostrare a schermo i dati scaricati dal database.

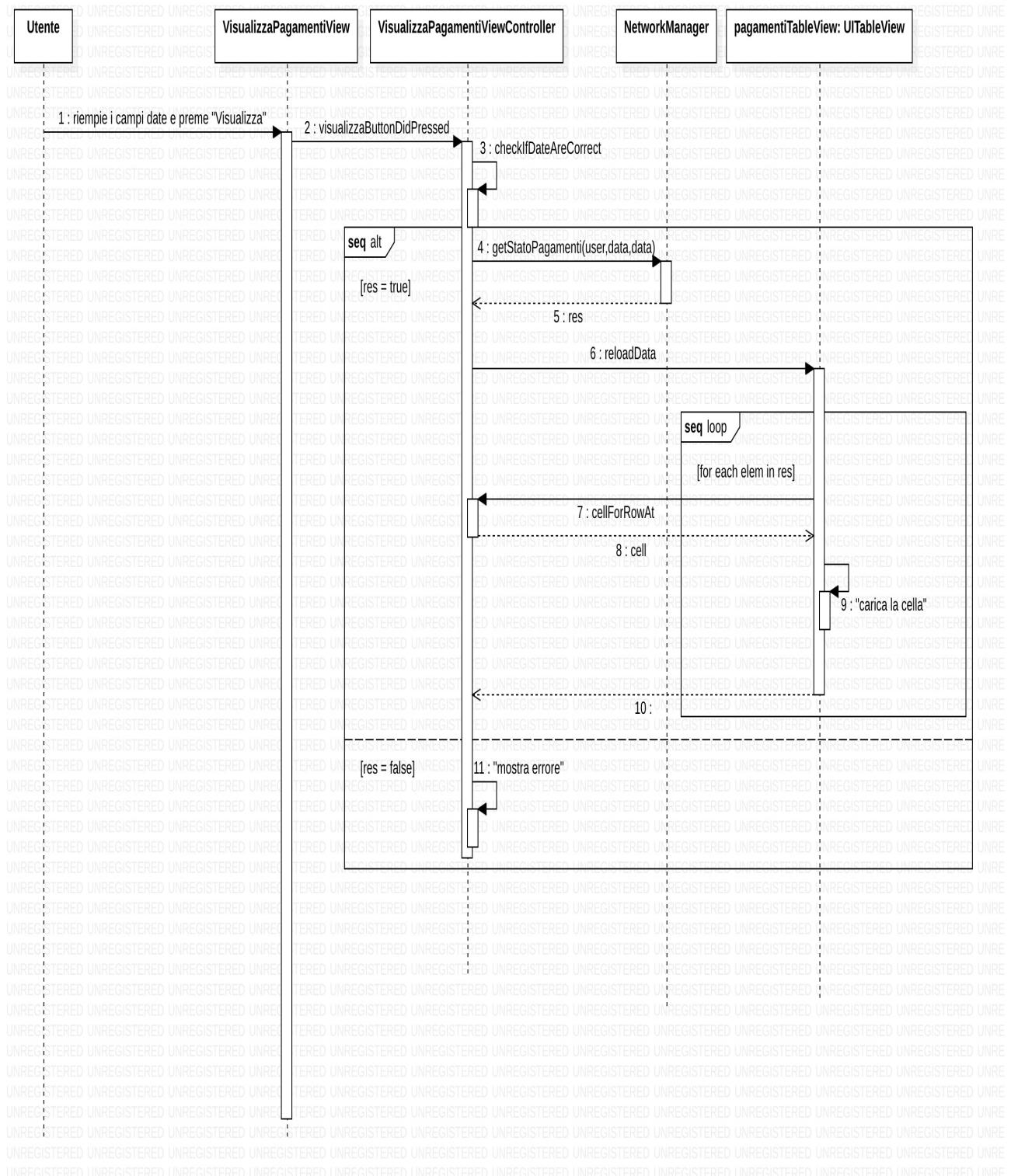
La classe sarà codificata con C38. Viene aggiunta all’insieme SIS e viene aggiornato il grafo delle dipendenze.

- Modificare la classe “NetworkManager”. Aggiungere il metodo “getStatoPagamenti(idUtente)” che si occuperà di lanciare due script diversi che ritorneranno le letture e le ingiunzioni associate a quell’utente. Il metodo le memorizzerà in un array e le ritornerà al controller chiamante che potrà così mostrarle a schermo. C25 viene aggiunto all’insieme SIS e viene aggiornato il grafo delle dipendenze.
- Creare due script PHP , “getLetture” e “getIngiunzioni” che prendono in ingresso idUtente e restituiscono i dati associati a quell’utente. C39 e C40 vengono aggiunti all’insieme SIS e viene aggiornato il grafo delle dipendenze.

L’insieme finale è SIS = {C23,C25,C37,C38,C39,C40}

Anche in questo caso, analizzando la natura delle modifiche, si può stimare che l’insieme CIS sia vuoto. Le modifiche aggiungono per lo più nuove componenti che si isolano rispetto alle vecchie funzionalità. Delle uniche due componenti vecchie modificate, verranno modificati e aggiunti metodi che non dovrebbero intaccare le vecchie funzionalità.

Andiamo ora a rappresentare il sequence diagram della modifica. Si assuma che l'utente abbia già fatto il login come cliente.



Implementando il sistema siamo andati a costruire l'AIS.

$$\text{AIS} = \{\text{C37}, \text{C38}, \text{C23}, \text{C25}, \text{C39}, \text{C40}\}$$

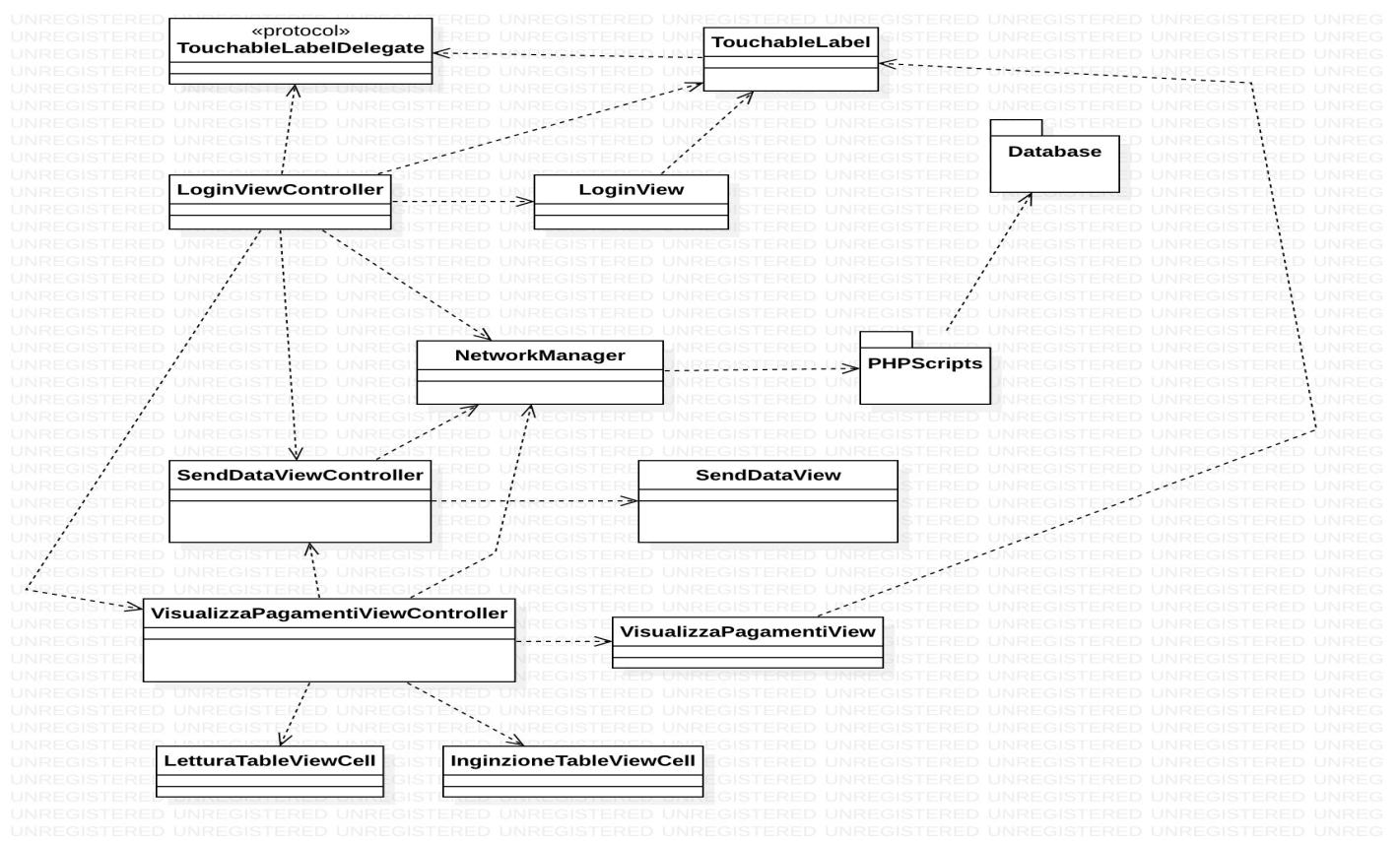
Sono state create, inoltre due classi che gestiscono le celle della table view che non erano state preventivate. Quindi vengono inserite nel discovered impact set.

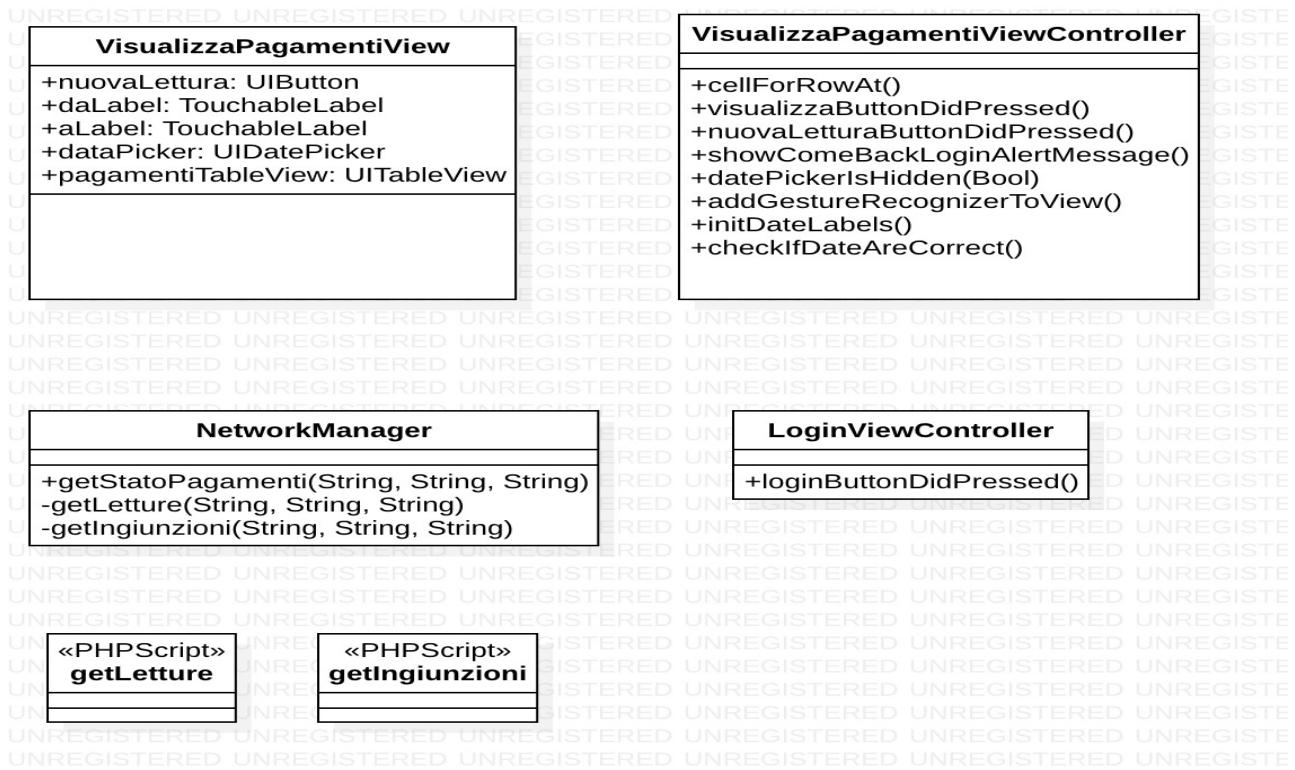
$$\text{DIS} = \{\text{C41}, \text{C42}\}$$

C41 = “LetturaTableViewCell”

C42 = “IngiunzioneTableViewCell”

Di seguito il class diagram aggiornato dell'applicativo iOS, seguito dal class diagram delle componenti nuove/modificate dove vengono messi in evidenza solo i metodi e le proprietà che sono state create o che sono state modificate.





## TESTING

Andiamo dapprima a testare le componenti del AIS u DIS che sono quelle che sono state create/modificate e che impattano direttamente sulla nuova funzionalità implementata.

### Componente C38 : “VisualizzaPagamentiViewController”

#### Metodo “cellForRowAt”

Il metodo restituisce alla table view chiamante le celle che deve mostrare a schermo. Esso costruisce la cella da ritornare in base al contenuto del campo di classe “lettura” di tipo dizionario stringa:stringa.

Contenuto “lettura”	Risultato attes	Risultato ottenuto
[effettuataDa:T],[effettuataDa:nil]	Una cella lettura e una cella ingiunzione	
[effettuataDa:T],[effettuataDa:C]	Due celle lettura	
[effettuataDa:nil],[effettuataDa:C]	Una cella lettura e una cella ingiunzione	
effettuataDa:nil],[effettuataDa:nil]	Due celle ingiunzione	

Il metodo non è testabile se non a mano poiché non è possibile fare unit test in quanto l’oggetto table view non può essere creato come stub. Non può essere testato come test di UI in quanto bisognerebbe appoggiarsi al database e non si potrebbe decidere a priori il valore del campo “lettura”. Testato a mano con i risultati della tabella qui sopra.

### Metodo “visualizzaButtonDidPressed”

Metodo testato con test di interfaccia tramite il metodo di test “testVisualizzaButtonDidPressed” della classe “PagamentiViewControllerUITest”

### **Metodo “nuovaLetturaButtonDidPressed”**

Metodo testato con test di interfaccia tramite il metodo di test “testNuovaLetturaButtonDidPressed” della classe “PagamentiViewControllerUITest”.

### **Metodo “showComeBackLoginAlertMessage”**

Metodo testato con test di interfaccia tramite il metodo di test “testShowComeBackLoginAlertMessage” della classe “PagamentiViewControllerUITest”.

### **Metodo “datePickerIsHidden”**

Il metodo prende in ingresso un valore booleano ed, in base ad esso, va a nascondere o mostrare alcune componenti grafiche dell’interfaccia.

Valore input : true

Risultato atteso :

dataPicker non visibile a schermo  
visualizzaButton visibile a schermo  
aLabel.isHidden visibile a schermo  
daLabel.isHidden visibile a schermo

Valore input : false

Risultato atteso :

dataPicker visibile a schermo  
visualizzaButton non visibile a schermo  
aLabel non visibile a schermo  
daLabel non visibile a schermo

Metodo testato con test di interfaccia tramite il metodo di test  
“testDatePickerIsHidden” della classe  
“PagamentiViewControllerUITest”.

### **Metodo “addGestureRecognizerToView”**

Il metodo serve ad intercettare il tocco dell’utente su un qualsiasi punto dello schermo che non sia un componente(bottone, labels ecc). Il metodo deve inserire la data corretta selezionata tramite il picker all’interno della label destinata alla data.

Testato con valori:

2017-5-4

2017-6-2

2018-1-14

2018-12-12

2018-1-15

Metodo testato con test di interfaccia tramite il metodo di test “testAddGestureRecognizerToView” della classe “PagamentiViewControllerUITest”.

### **Metodo “checkIfDateAreCorrect”**

Il metodo controlla che la data presente nella label “daLabel” sia minore della data presente nella label “aLabel”

daLabel	aLabel	Risultato atteso	Ris ottenuto
2017-01-01	2017-08-01	true	ok
2017-01-01	2018-05-02	true	ok
2019-01-01	2018-05-01	false	ok
2019-02-01	2017-02-01	false	ok
2018-04-02	2015-01-01	false	ok

### **Componente C25 : “NetworkManager”**

#### **Metodo “getStatoPagamenti”**

Il metodo scarica dal database le letture e le ingiunzioni relative all’utente loggato relative al periodo di tempo tra le due date selezionate. In caso non vi siano risultati, il sistema mostra un messaggio a schermo.

daLabel	aLabel	Risultato atteso	Ris ottenuto
2017-01-01	2017-08-01	2 elementi	ok
2017-01-01	2018-05-02	11 elementi	ok
2019-01-01	2019-02-02	Nessun elemento	ok
2015-02-01	2015-12-31	Nessun elemento	ok
2016-01-02	2016-12-01	Nessun elemento	ok

Insieme a questo metodo sono stati testati anche i metodi delle componenti PHP “getLettura” e “getIngiunzioni” in quanto entrambe vengono utilizzate all’interno del metodo “getStatoPagamenti”.

## TEST DI REGRESSIONE

Sono stati rieseguiti tutti i metodi presenti all’interno dei pacchetti di test “SoftEngineUniNAUITestUI”, “SoftEngineUniNATest2”, tutti con esito positivo. Sono state testate, inoltre, tutte le funzionalità dell’applicativo Desktop, con gli stessi valori dei test della precedente funzionalità implementata, tutti con esito positivo. Ciò era abbastanza plausibile, in quanto la nuova funzionalità implementata riguardava soltanto lettura dal database e veniva implementata totalmente sull’app mobile, non andando in alcun modo ad impattare sull’applicativo desktop. Il test sulle funzionalità dell’applicativo desktop è stato effettuato avvalendosi del documento “PIANO DI TEST CASES sistema vecchio.pdf”, come già accaduto per la precedente modifica.

## PRECISAZIONI SUL TESTING

Si vuole precisare che, per alcune componenti, non è stato possibile effettuare un accurato test di unità in quanto molte di esse, la maggior parte, andavano ad interagire con componenti grafiche della view che non potevano essere sostituite. I test eseguiti su queste componenti sono test di interfaccia.