Università degli Studi di Modena e Reggio Emilia
Dipartimento di Ingegneria “Enzo Ferrari”

Corso di Laurea Triennale in Ingegneria Informatica

Test di carico e stress su piattaforma clinica

Relatore:
Prof. Sonia Bergamaschi

Laureando:
Naima El Khattabi

Anno Accademico 2020-2021

Prima di iniziare con la trattazione, vorrei dedicare questa pagine a chi mi è stato vicino durante tutto il mio percorso.

Un ringraziamento speciale è dedicato a mia mamma Ilham e mio padre Bouchta che mi hanno sempre sostenuta sin da quando ero piccola, appoggiando ogni mia scelta e decisione, e che mi hanno sempre mostrato la giusta via da seguire.
Avete fatto tanti sacrifici per permettermi di raggiungere tutti i miei traguardi, e un 'grazie' non sarà mai abbastanza. Siete i migliori genitori che si possano desiderare. Grazie.

Grazie al mio relatore Sonia Bergamaschi per essere stata un punto fermo in tutto il mio percorso universitario e per avermi fatto amare ancora di più questo indirizzo.

Ringrazio l'azienda DataRiver Srl per avermi dato l'opportunità di imparare e mettermi in gioco in un progetto di grande importanza.

Ringrazio le mie sorelle Yasmine e Sofia per essermi state accanto nei momenti di studio intenso e quando pensavo di non farcela più, e per essere le migliori sorelle al mondo.

Ringrazio mia zia Nawale per essere stata una costante ispirazione sempre, e per avermi dato la forza di credere e intraprendere i miei sogni.

Ringrazio le mie due maestre del cuore, Mariuccia e Giuseppina, per avermi insegnato tutto quello che so; è grazie a voi che sono fiera di essere diventata quello che sono oggi. Non finirò mai di ringraziarvi.

Ringrazio i miei amici e colleghi, Amrit, Denise, Sabrina, Roberta, Federico e Alessandro, per essermi stati sempre accanto,per le giornate di studio e non, per le risate e i pianti che hanno reso tutto questo periodo unico e indimenticabile.

Ringrazio anche Mario e tutto lo staff del CarpeDiem, che sono stati una costante durante il mio percorso e che mi hanno fatto crescere come persona.

Infine, dedico questa tesi a me stessa, che dopo sacrifici lavorativi e di studio non ho mai mollato e ho sempre creduto in me stessa.

INDICE

1. Introduzione

Ho svolto un periodo di tirocinio presso l'azienda *DataRiver SRL, c*he è una PMI[11] innovativa accreditata come Laboratorio di Ricerca Industriale della Rete Alta Tecnologia della Regione Emilia Romagna. La sua mission è quella di permettere alle aziende di comprendere facilmente i propri dati, attraverso una visione chiara ed unificata delle sorgenti informative interne ed esterne. L'analisi dei Big Data consente alle imprese di imparare dall'esperienza e di ottimizzare i processi decisionali, produttivi e previsionali.

Lo scopo del mio tirocinio è stato quello di individuare una procedura automatizzata per inserire dati in blocco sulla piattaforma MyHealth e verificare l'andamento delle prestazioni e il comportamento dell'applicazione a regime.

- Andrò a realizzare 4 tipologie di test sulla piattaforma
    - numero di accessi contemporanei da parte di utenti distinti
    - numero di soggetti arruolati nella piattaforma
    - numero di centri presenti nel progetto
    - numero di form programmati nel flusso di raccolta dati

E' stata eseguita l'implementazione delle seguenti attività :

- caricamento di dati direttamente sui database della piattaforma in modo automatico e mantenibile
- analisi e implementazione di una modalità per testare automaticamente gli accessi concorrenti
- Utilizzo ed estensione dei Vaadin Test Bench per verificare i tempi di risposta dell'applicazione nei casi di stress (es. accessi concorrenti).

Il seguente elaborato ha lo scopo di mostrare il lavoro svolto e di comprendere le varie tecnologie e metodologie utilizzate per raggiungere gli obiettivi posti dall'azienda.

Il secondo capitolo si occupa di analizzare gli strumenti che sono stati utilizzati per la realizzazione di questo progetto. Partiremo parlando di Vaadin Test Bench che ci servirà principalmente nella parte finale del progetto, per poter creare i test per verificare i tempi di risposta dell'applicazione. Successivamente si passerà ad una descrizione esaustiva sul tipo di database utilizzato, sulle tecnologie che hanno permesso l'implementazione del lavoro da remoto, dell'ambiente di sviluppo Eclipse utilizzato per la realizzazione della procedura automatica per l'inserimento dati ed infine verrà descritto anche Tomcat.

Il database utilizzato ci permette di gestire la piattaforma clinica MyHealth e in particolare agire sui dati dei pazienti che vengono raccolti in dei form. Si agirà sul database per analizzare la struttura della piattaforma MyHealth e per permettere l'inserimento dei dati dei pazienti nei vari form per poter realizzare i vari test di carico/stress sulla piattaforma.

Il terzo capitolo tratterà tutto ciò che è stato detto precedentemente e affronterà tematiche come la scelta del database impiegato per il progetto, la scelta del DBMS, la configurazione di esso ,la sua organizzazione e le novità introdotte per adempiere allo scopo del progetto.

Il quarto capitolo consiste nella vera e propria descrizione del progetto.

Inizierò con una descrizione sulla configurazione del progetto sull'ambiente di sviluppo Eclipse, poi parlerò delle idee sulle possibili implementazioni per l'inserimento dati nella piattaforma e dell'effettiva scelta intrapresa a riguardo.

In un secondo momento verranno descritti in 2 macro step i vari punti svolti per l'effettiva generazione dei dati.

Ed infine si parlerà del processo di validazione.

Il quinto capitola riporta le note conclusive relative al lavoro svolto e affronta le possibili estensioni future del progetto.

Il sesto capitolo riporta la sitografia utilizzata nel seguente elaborato.

2. Strumenti utilizzati

2.1 Vaadin Test Bench

Il Vaadin Test Bench(1) è uno strumento che crea automaticamente i PDF dove riportiamo i test svolti sulle performance della piattaforma.

Vaadin TestBench è uno strumento per la creazione e l'esecuzione di test di integrazione basati su browser per l'applicazione Vaadin. TestBench simula un utente dell'applicazione, esegue le attività specificate utilizzando il codice Java e verifica che le azioni previste vengano eseguite nell'applicazione.

TestBench include anche un supporto speciale per altri prodotti Vaadin, rendendo i test facili e robusti rispetto alle soluzioni di test web generiche.

Sebbene non sia lo scopo principale di TestBench, lo si può anche utilizzare per automatizzare attività banali come la compilazione di moduli.

La validazione è un'operazione che viene eseguita per testare le funzionalità nell'applicazione che utilizziamo , per esempio nei crf l'aggiunta di un gruppo, visita ecc.

L'obiettivo è quello di far aprire una finestra comandata dal java(test bench) che faccia l'inserimento dati, clicchi i bottoni e quindi simuli tutte le operazioni classiche che un utente dovrebbe svolgere.

Ci sono le OQ (operation qualification) ovvero delle suite di test che si possono svolgere.

Una volta fatte le insert potremo creare un altra OQ dove ci sono i passaggi che ci interessano, es. quanto ci mette a caricare l'elenco dei pazienti dopo che ne ho inseriti 60000.

Quando il test va a buon fine o non produce un pdf standard ovvero il report che contiene screenshot e risultati di ogni step svolto all'interno del test. Se invece l'applicazione, per qualche motivo, non risponde correttamente ci darà un failed.

Negli OQ abbiamo prima FUNCTIONAL REQUIREMENT e per ogni macro step ci sono altri sottopassaggi. Dopo aver contenuto i macropoint (step principali), passiamo ai sottopassaggi da svolgere; questi sono tutti gli step da svolgere all'interno del test.

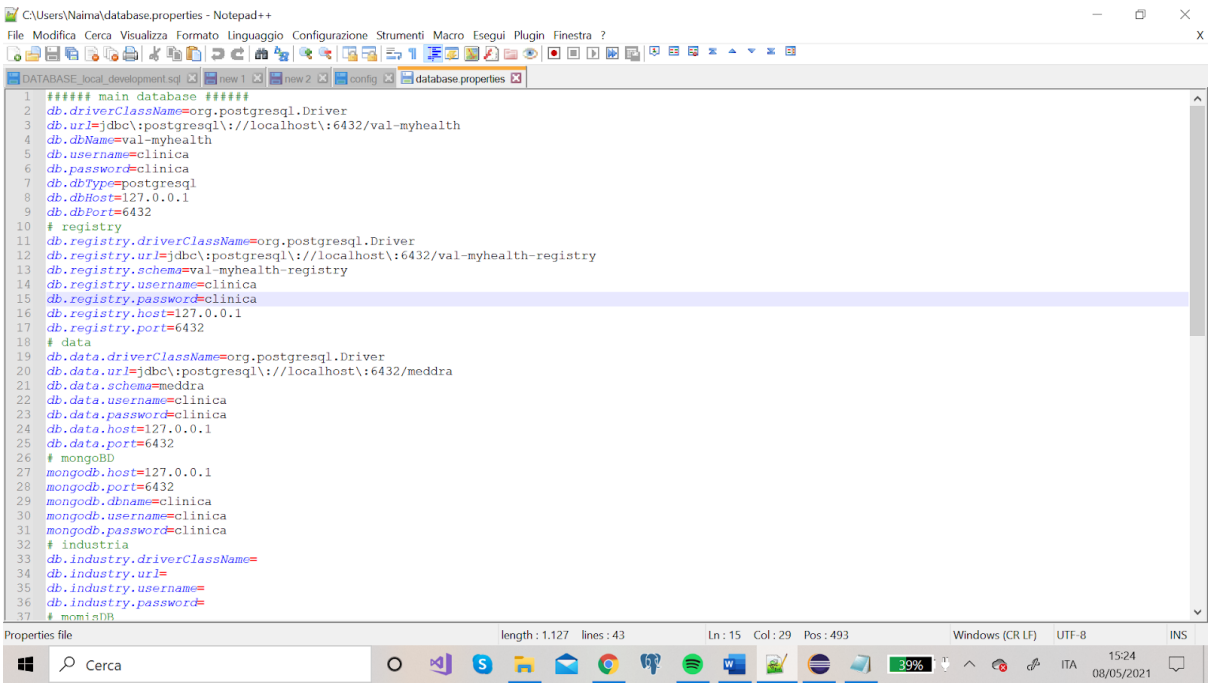
Es. *Faccio login, inserisco l'utente, crf e vedo quanto tempo ci mette primo dell'insert e dopo l'insert.* Così posso fare un paragone effettivo.

Ad ogni step vengono fatti degli screenshot sia in caso di fallimento che in caso di riuscita.

Nel nostro caso dopo aver riportato gli step da svolgere, bisognerebbe riportare nel report le tempistiche dei vari step.

2.1.1 INSTALLAZIONE TEST BENCH :

- modifichiamo il file database properties :



  Infatti, come mostra la figura, il database adesso non è più 'myhealth' ma 'val-myhealth'

- in eclipse Iot-> backend->...

- entity -> mappatura ad oggetti del database tramite hibernate --> tabella phase ci sono gli attributi
- hibernate mi genera un'interfaccia con il database
- In entity ci sono tutte le tabelle del db.
- bin-> phase bin sarà uguale a phase entity. Gli oggetti qui dentro non modificano il db e mi permette di mettere insieme vari campi che in java conviene avere in un unico oggetto
- dao = data access object -> oggetti java con cui vado ad interagire con le strutture; praticamente quì ci sono tutte le query che vado ad eseguire

2.2 GIT

Git[2] è un software per il controllo di versione distribuito utilizzabile da interfaccia a riga di comando, creato da Linus Torvalds nel 2005.
Git nacque per essere un semplice strumento per facilitare lo sviluppo del kernel Linux ed è diventato uno degli strumenti di controllo versione più diffusi.
Git pensa i suoi dati come un flusso di *snapshot* di un file system in miniatura.

Ogni volta che si effettua un commit, o si salva lo stato del progetto, Git fondamentalmente scatta una fotografia dello stato di tutti i file in quel momento e memorizza un riferimento a tale snapshot.

Per essere efficiente, se i file non sono cambiati (stesso checksum), Git non memorizza nuovamente il file, ma soltanto un collegamento al file identico precedentemente memorizzato *(blob)*.

Quindi crea dei commit che contengono un puntatore all'oggetto tree del progetto root, per creare uno snapshot quando richiesto.

Ha 3 stati :

1. Committed -> memorizzato nel db locale
2. Modified -> modificato
3. Staged -> contrassegnato un file modificato per essere incluso nel prossimo commit

Per far uso di GIT ho ricorso all'utilizzo di Cygwin, che introdurrò successivamente, anche se Eclipse ci permette già il suo utilizzo tramite wizard.

2.3 PostgreSQL

PostgreSQL[3] è un potente sistema di database relazionale a oggetti open source che utilizza ed estende il linguaggio SQL combinato con molte funzionalità che archiviano e scalano in modo sicuro i carichi di lavoro di dati più complicati.
PostgreSQL ha guadagnato una solida reputazione per la sua comprovata architettura, affidabilità, integrità dei dati, robusto set di funzionalità, estensibilità e dedizione della comunità open source dietro il software per fornire costantemente soluzioni performanti e innovative.
PostgreSQL inoltre funziona su tutti i principali sistemi operativi.
PostgreSQL è dotato di molte funzionalità volte ad aiutare gli sviluppatori a creare applicazioni, gli amministratori a proteggere l'integrità dei dati e creare ambienti a tolleranza d'errore e ad aiutarti a gestire i tuoi dati, indipendentemente da quanto grande o piccolo sia il set di dati. Oltre ad essere gratuito e open source , PostgreSQL è altamente estensibile.

Si è deciso di utilizzare PostgreSQL (versione 10.16) come database per questioni di uniformità per l'intero progetto che appunto ha un database postgres, e anche perchè Postgres è un database relazionale a oggetti e include funzionalità come l'ereditarietà delle tabelle, cosa che Mysql non permette essendo un database puramente relazionale.

2.4 Cygwin

Cygwin[4] è una distribuzione di software libero, che consente a diverse versioni di Microsoft Windows di svolgere alcuni compiti in maniera esteticamente e funzionalmente simile a un sistema Unix. Lo scopo principale della sua esistenza è il porting di software che gira su sistemi POSIX (come i sistemi Linux, o quelli basati sulla

BSD, e altri ancora) su Microsoft Windows, in maniera tale che sia necessaria poco più di una ricompilazione dai sorgenti degli stessi per ottenere un funzionamento corretto.

Cygwin è uno strumento Open Source che simula la shell di Linux su Windows, senza dover ricorrere all'installazione di una macchina virtuale, e ci aiuta in locale ad utilizzare i vari comandi GIT.

Quando dovrò fare delle modifiche dovrò lanciare i seguenti comandi su cygwin :

- git add -> scelgo i file da committare
- git commit -m
- git push -> per rendere visibile il mio commit a tutti gli altri utenti

Gli altri comandi che mi serviranno sono :

- git fetch -> controllo se sul server (datariver_srv) ci sono nuovi aggiornamenti
- git status -> vede com'è la mia working directory confrontando la mia situazione locale
- git log -> permette di vedere tutti i commit
- git pull -> prende giù le modifiche e aggiorna il mio master locale
- git branch -a -> mostra la posizione all'interno dell'albero
- git checkout *nome_branch* -> permette di spostarsi su un altro branch

2.5 Eclipse

Per il nostro progetto si è deciso di utilizzare come ambiente di sviluppo, principalmente per comodità, in quanto il progetto è stato sviluppato su eclipse e viene modificato sempre quì.

Eclipse[5] è un ambiente di sviluppo integrato multi-linguaggio e multipiattaforma. Ideato da un consorzio di grandi società quali Ericsson, HP, IBM, Intel, MontaVista Software, QNX, SAP e Serena Software, chiamato *Eclipse Foundation*. È un software libero distribuito sotto i termini della Eclipse Public License.

Viene scelto anche perchè ci permette di utilizzare i *Wizard* che utilizzeremo più avanti:

Un wizard[6] è una serie di pagine che guidano l'utente attraverso un'attività complessa. I pulsanti Indietro e Avanti consentono all'utente di spostarsi avanti e indietro tra le pagine. In genere, ogni pagina raccoglie un informazione; quando l'utente fa clic sul pulsante Fine, le informazioni vengono utilizzate per eseguire un'attività. In qualsiasi momento prima di fare clic su Fine, l'utente può annullare l'attività, che dovrebbe annullare tutti gli effetti collaterali dei passaggi completati fino a quel momento.

Un wizard viene generalmente presentato in una finestra di dialogo, ma non è necessaria. L'astrazione denominata IWizardContainer rappresenta il contesto in cui viene eseguito un wizard. È garantito che un contenitore di wizard abbia un titolo, un'area messaggi e un monitor di avanzamento.

2.6 Tomcat

Il software Apache Tomcat®[7] è un'implementazione open source delle specifiche Jakarta Servlet, Jakarta Server Pages, Jakarta Expression Language, Jakarta WebSocket, Jakarta Annotations e Jakarta Authentication. Queste specifiche fanno parte della piattaforma Jakarta EE. La piattaforma Jakarta EE è l'evoluzione della piattaforma Java EE. Tomcat 10 e versioni successive implementano le specifiche sviluppate come parte di Jakarta EE. Tomcat 9 e versioni precedenti implementano le specifiche sviluppate come parte di Java EE.

Tomcat[8] è un Servlet Container ed un JSP Engine. Un motore quindi in grado di eseguire lato server applicazioni Web basate sulla tecnologia J2EE e costituite da componenti Servlet e da pagine JSP. L’interazione tra queste due tecnologie rende disponibili tutti gli strumenti necessari per Web application per ogni utilizzo. Da non sottovalutare poi il fatto che Tomcat è disponibile per tutti i sistemi operativi, quindi lo potremo usare sia con Windows sia con Linux se volessimo.

La versione scaricata : tomcat 9 030 -> servlet container, applic crea una cartella web app e lui si comporta come il jetty nella porta 8080 l'applicazione; rende l'applicazione raggiungibile da browser.

Non posso usare jetty perchè mi servirà per l'altra applicazione altrimenti sarebbe troppo dispendioso per il pc mandare contemporaneamente da eclipse entrambe le run, dal momento che l'app e la piattaforma dei test dovranno lavorare insieme.

Nel nostro caso lo andremo ad utilizzare per eseguire la **validazione test bench** -> localhost 8090.

La home di tomcat si trova in : *C:\Windows\ServiceProfiles\LocalService.*

3. Database

Il database che andremo ad utilizzare è di tipo verticale, così non avremo la necessità di creare una nuova scheda per ogni dato modificato.

Un database verticale[9] è uno in cui il layout fisico dei dati è colonna per colonna anziché riga per riga. Anziché essere disposti in strutture di record orizzontali ed elaborati verticalmente, i dati in un database verticale sono disposti in strutture verticali, noti come alberi predicati o P-tree, ed elaborati orizzontalmente.

Organizzando i dati per colonna anziché per riga, è possibile eseguire query o ricerche sui dati senza accedere alle pagine su un disco rigido non interessate dalla query e quindi aumentare la velocità di recupero dei dati .

Un altro vantaggio dei database verticali è che consentono di archiviare i dati in pagine di grandi dimensioni. Le dimensioni di una pagina di grandi dimensioni consentono di recuperare un numero elevato di elementi di dati rilevanti in un'unica operazione di lettura. Al contrario, una singola operazione di lettura su un database orizzontale recupera non solo elementi di dati rilevanti, ma anche attributi, o colonne, che non sono rilevanti per la query in questione e favorisce dimensioni di pagina ridotte.

3.1 Configurazione

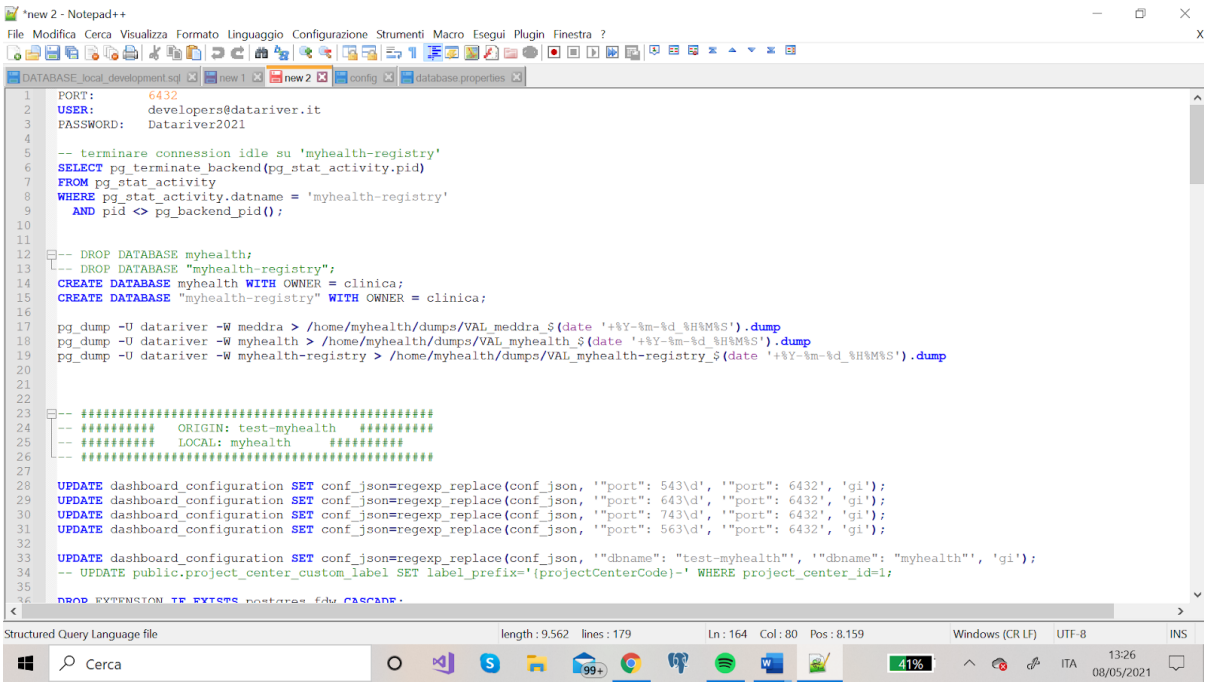
Per configurare il database è stato necessario lanciare dei dump e restore sul terminale di Windows.

I dump non sono altro che dei backup di PostgreSQL nella forma compressa, quindi sono i dati veri e propri del nostro database.

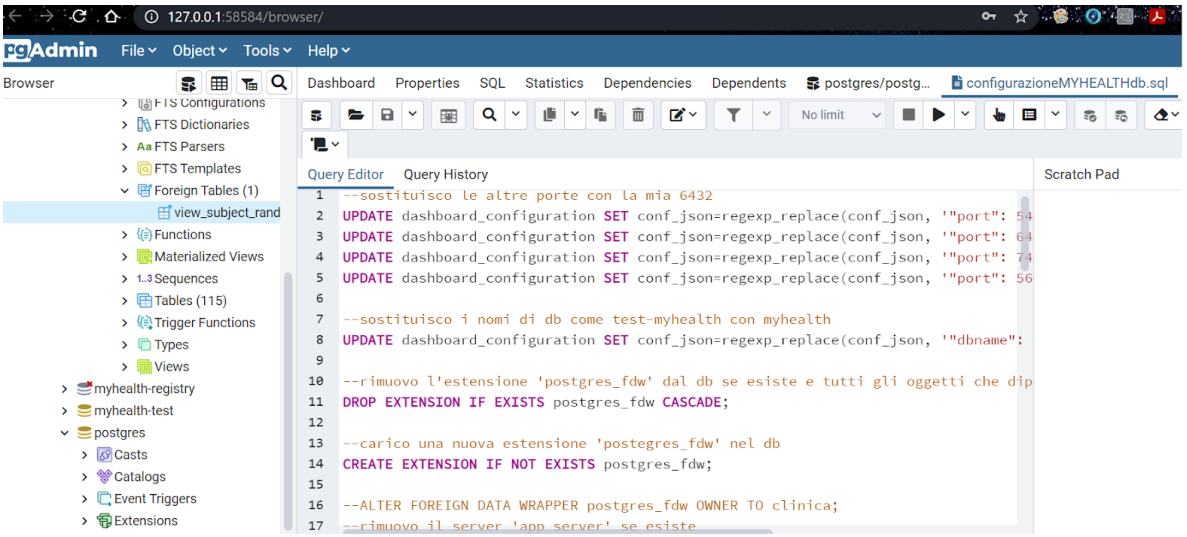
1. Per prima cosa bisogna posizionarsi sul prompt di Windows e andare su "c:\ProgramFiles\PostgreSQL\10\bin"
2. Fare il **dump/restore** della tabella myhealth
3. prima facciamo un **restore** : .\psql.-p 6432 -u clinica -w myhealth < c:*path del backup file*.dump
4. Dopo aver lanciato questo comando ci verrà chiesta la password dell'utente CLINICA
5. Fare allo stesso modo ma con il **dump** : .\pg_dump -p 6432 -u clinica -w myhealth < c:*path del backup file*.dump
6. Fare il **dump/restore** della tabella myhealth-registry
7. Al momento non eseguo il restore/dump della tabella meddra perchè non è necessario.

Le operazioni da svolgere dalla parte del database su postrgres sono :

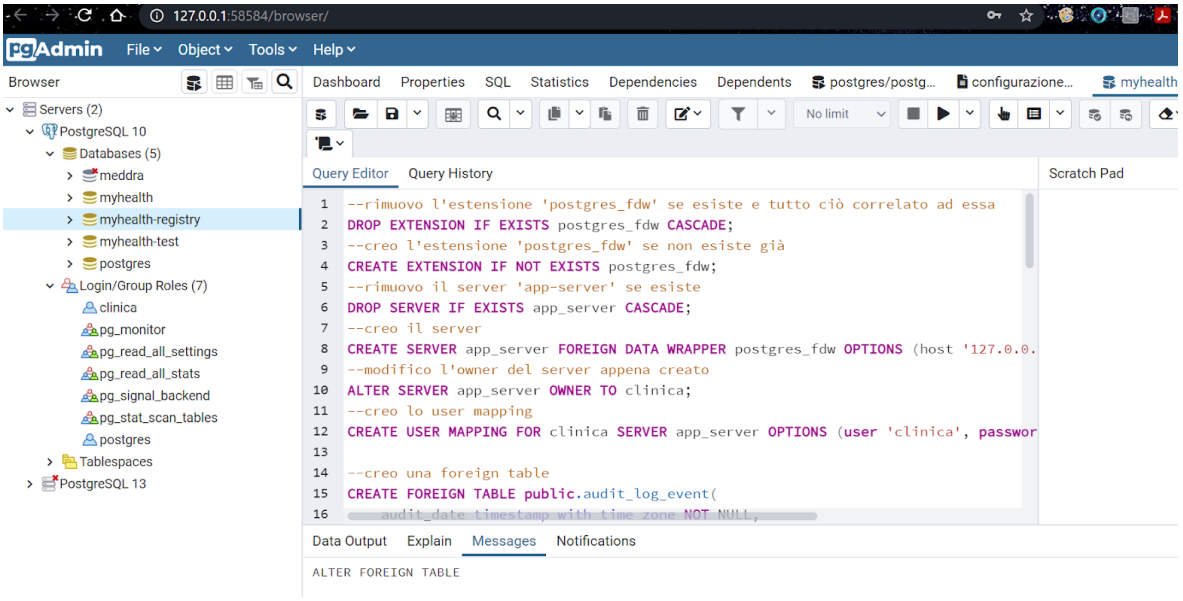
1. Terminare *connession idle* su '*myhealth-registry*'
2. Creare i database
3. Lancio delle query sul db myhealth, ma prima di far questo :
    - vado su dashboard configuration con la mia porta 6432 e riconfiguro tutte le altre porte con la mia.



4. Adesso sono pronta a lanciare le query sul database ‘myhealth’

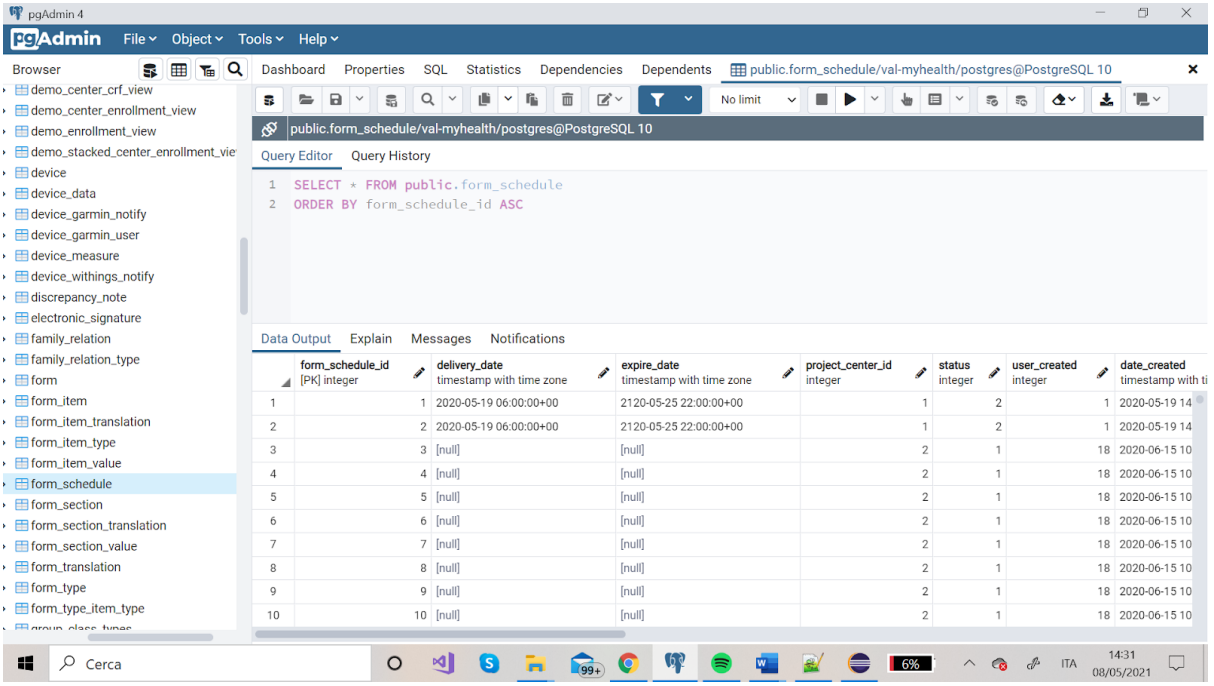


5. Posso procedere a fare lo stesso anche per la tabella ‘*myhealth-registry*’ che raccoglie le informazioni anagrafiche

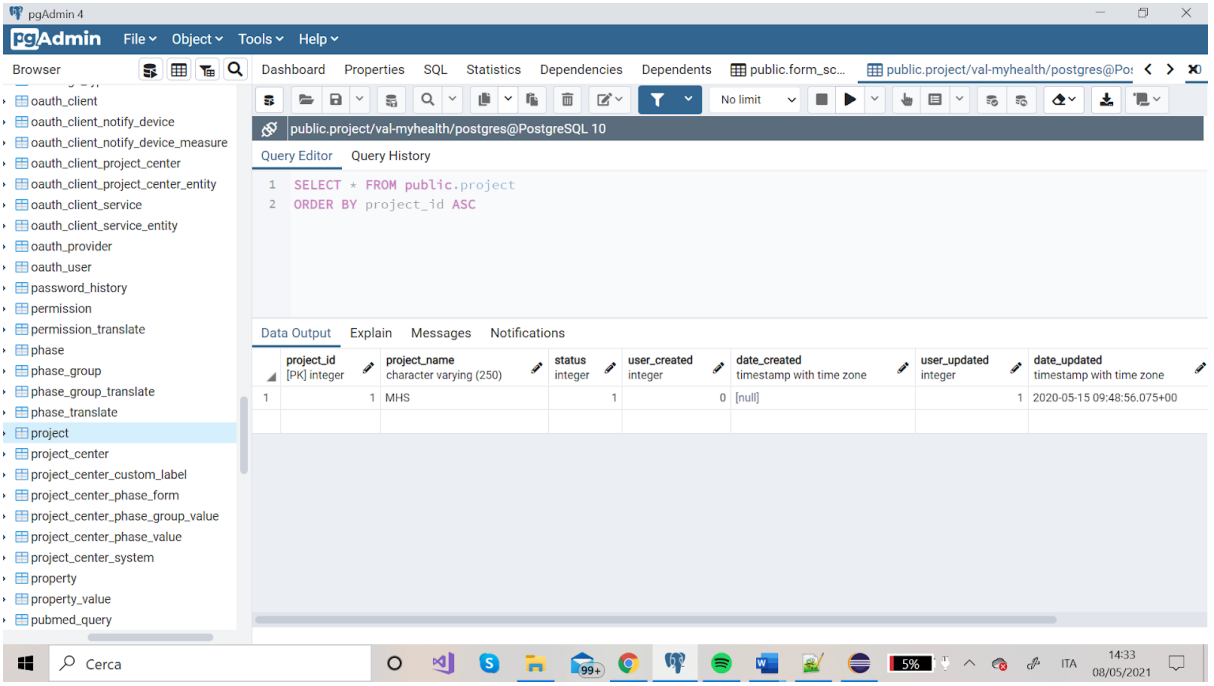


3.2 Organizzazione : tabelle

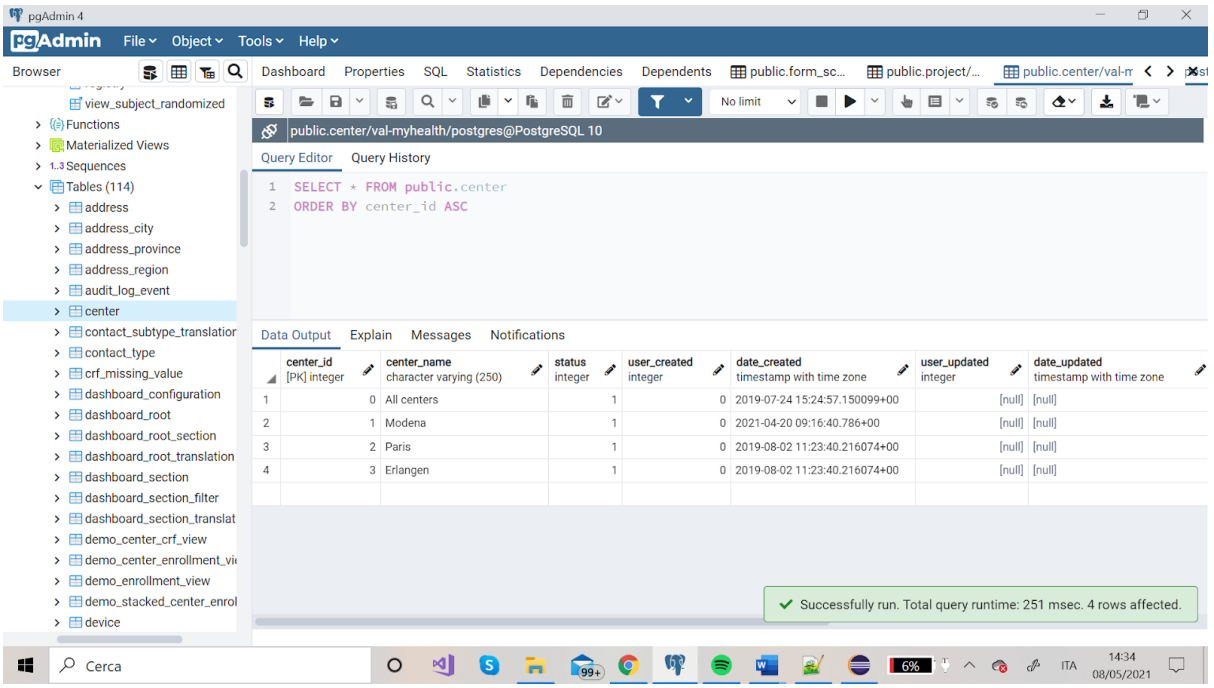
- *form_schedule* : è la tabella centrale per i dati ; mi dà l'informazione per raggiungere il form del singolo user



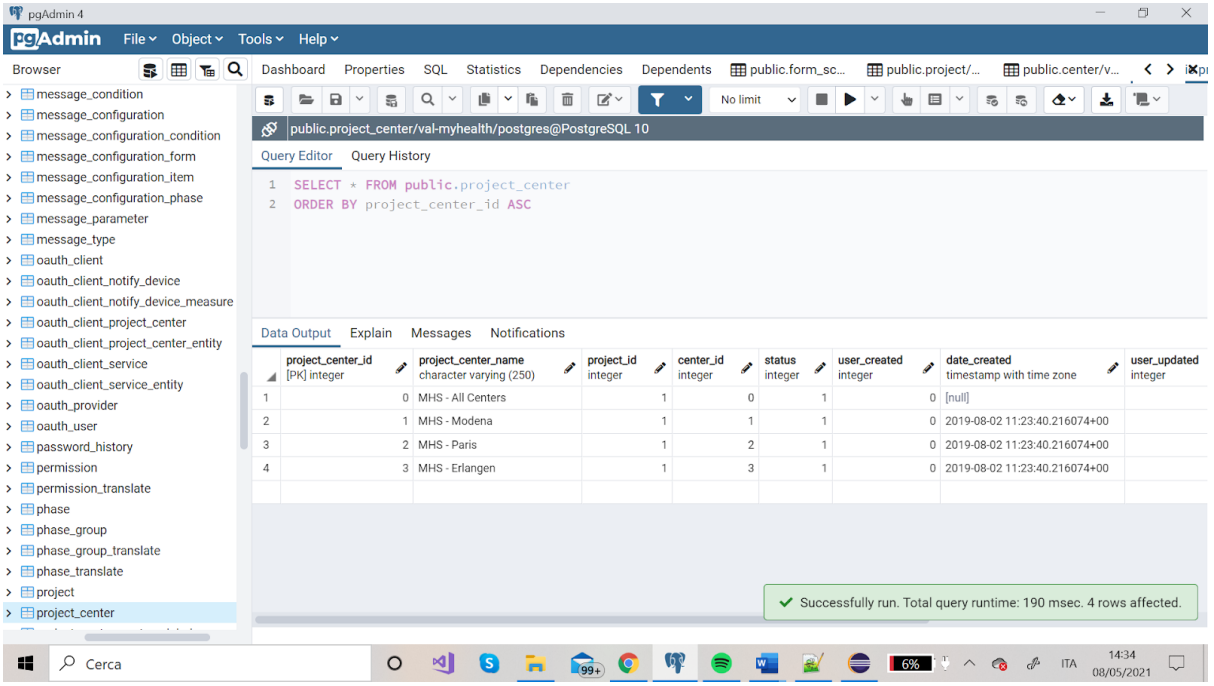
- *project* : informazione generale sullo studio



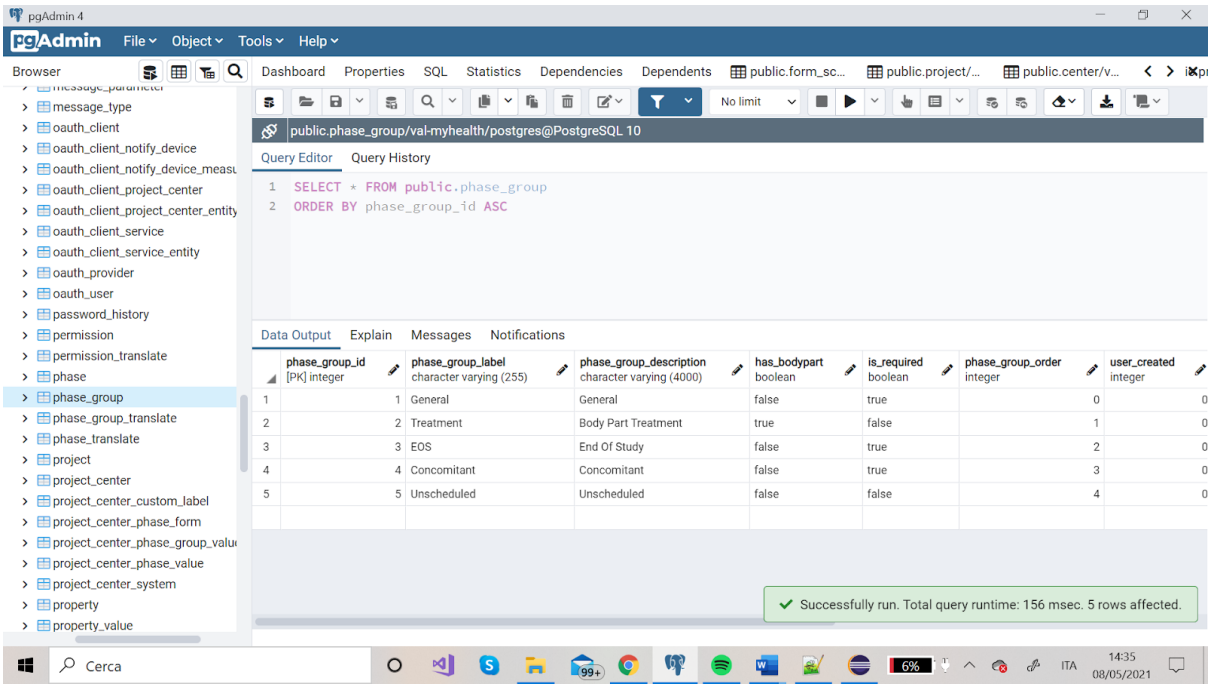
- *center* : anagrafica di tutti i centri (sono 4 al momento)



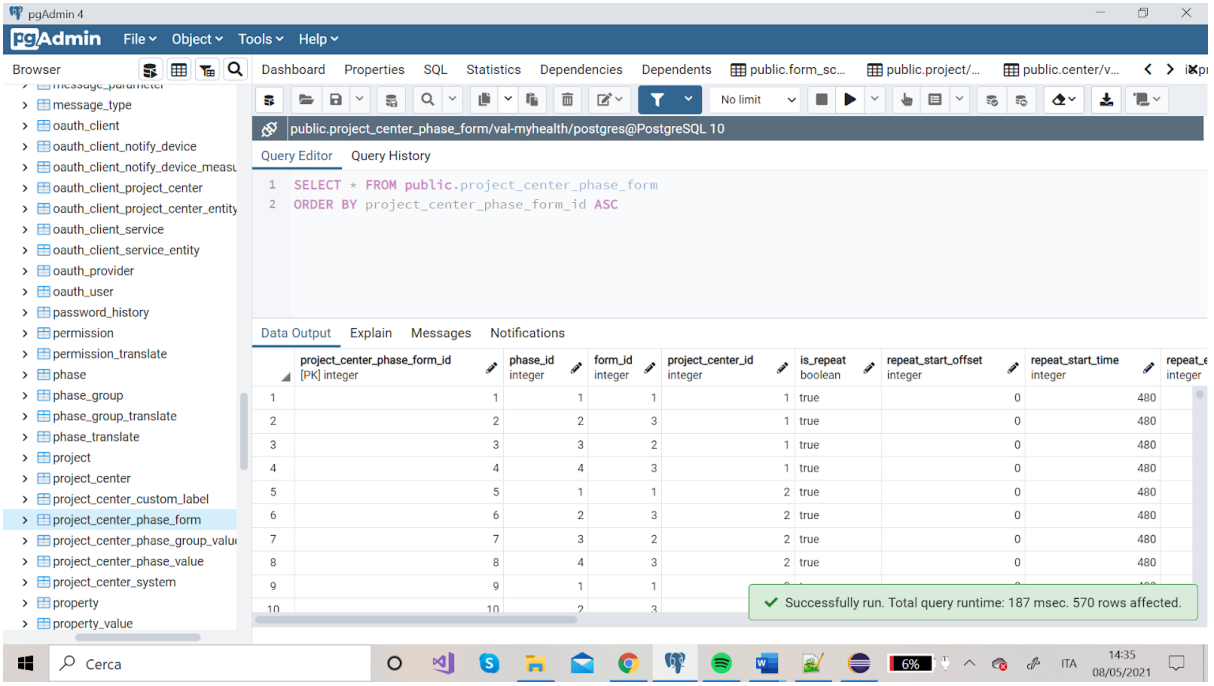
- *project-center* : associazione tra project e center ; realizzo una struttura ad albero (registration, diagnosi)



- *phase-group* : sono associati ai centri ; anagrafica dei gruppi associati a phase (anagrafica delle fasi, ovvero delle visite)

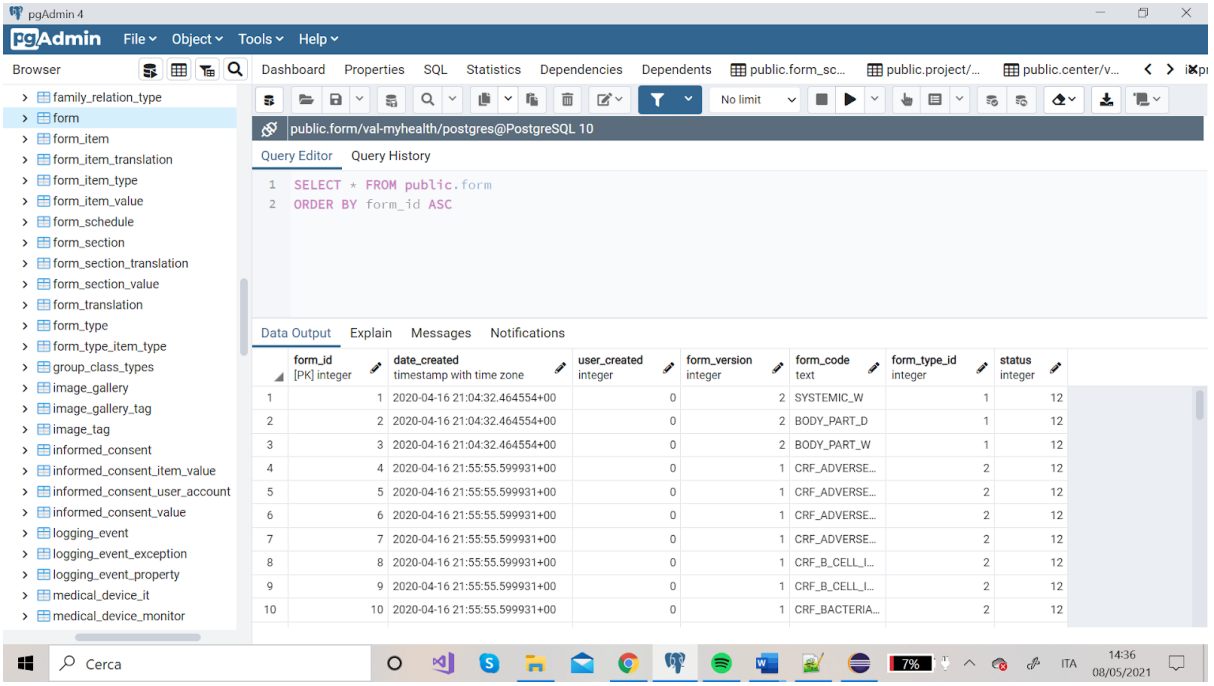


- *project-center-phase-form* : mette in relazione le tabelle di cui abbiamo parlato, es : “la fase 4 è disponibile per il ventro 1 e ha i form 7,22,18 e 13”

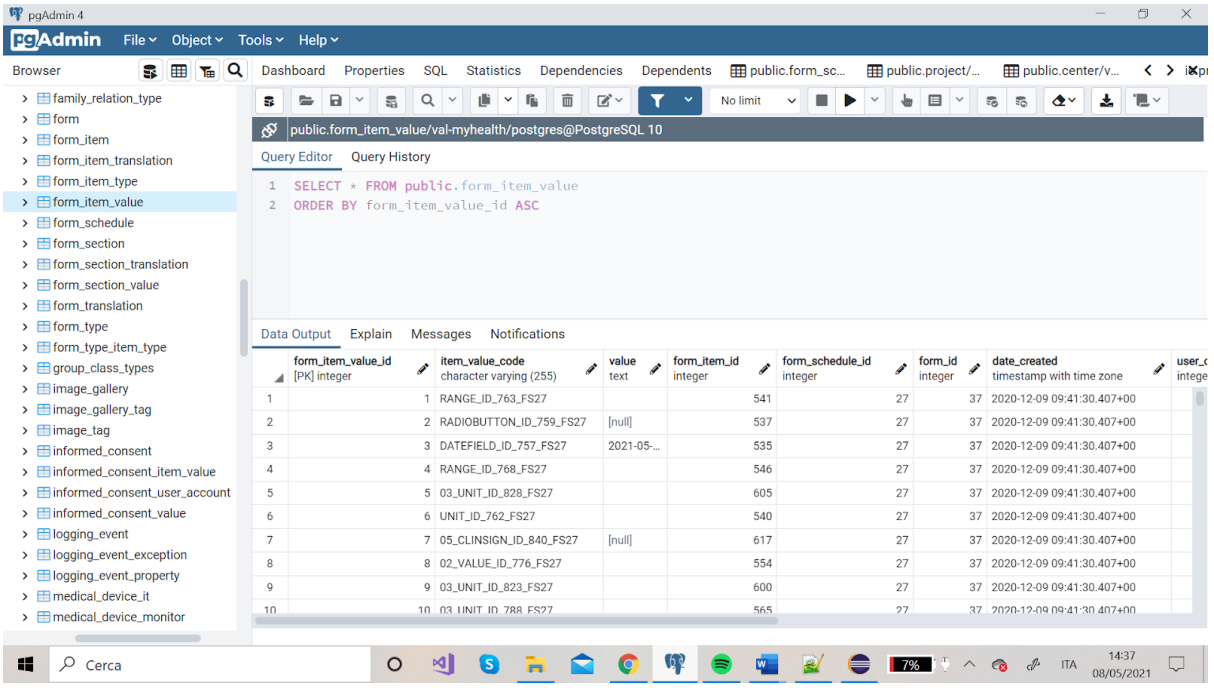


Altre tabelle fondamentali sono :

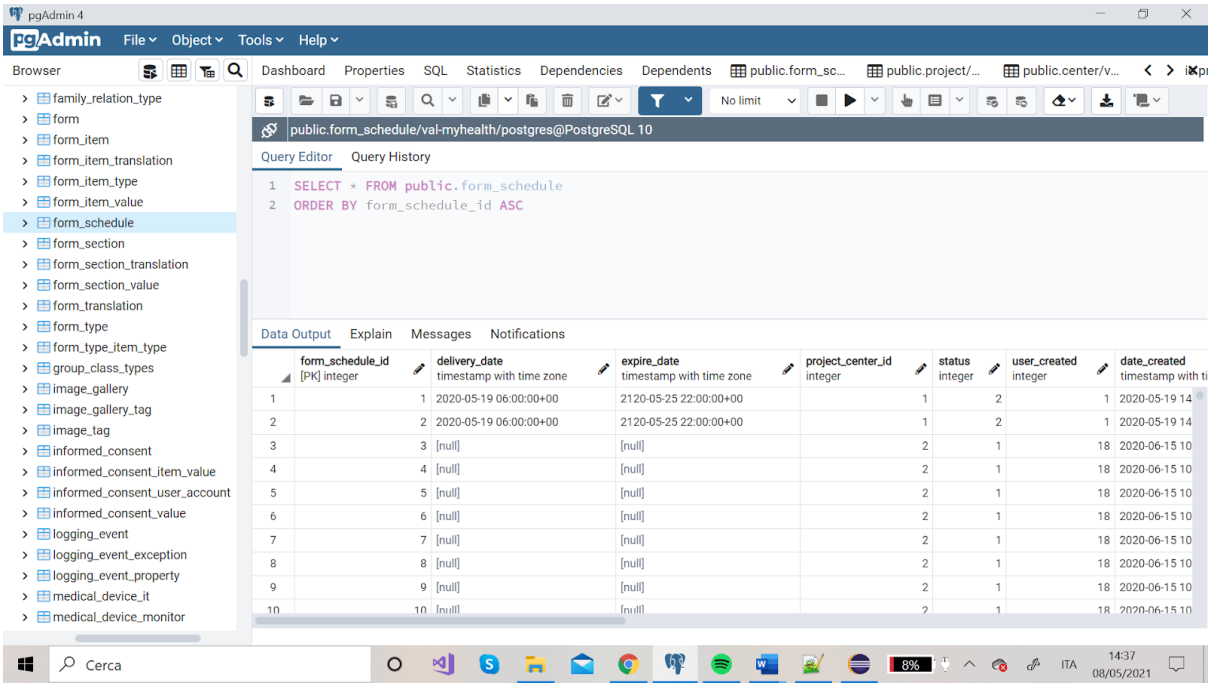
- *audit_log_event* : status; l'audit trail mantiene lo storico di tutte le tabelle, di tutte le modifiche
- *dashboard_configuration* : configurazione dei grafici presenti sulla piattaforma
- *discrepancy_note* : annotazioni nelle CRF
- *form* : tabella dei questionari e CRF



- *form_item_value* : contiene tutti i dati dei questionari

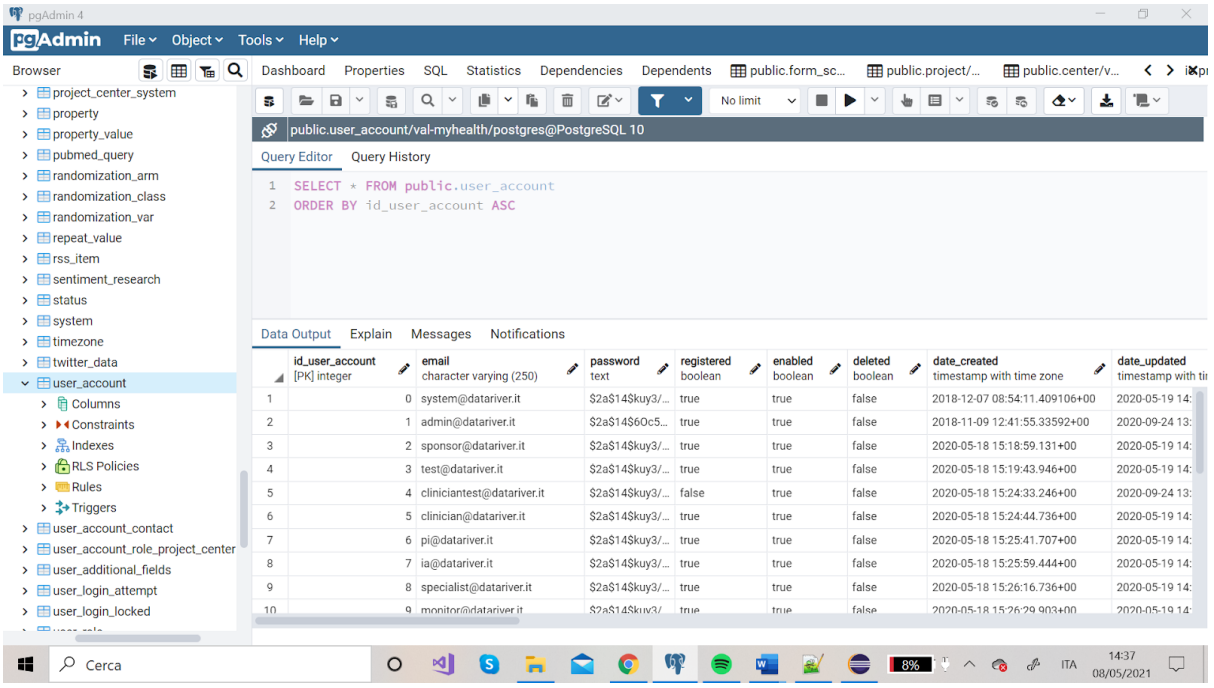


- *form_schedule* : è la rappresentazione di tutti i form presenti per ogni utente



- property : elenco di tutte le configurazioni del database
- *randomization_class* : permette di dividere i pazienti in 2 gruppi in modo casuale in base ad alcune variabili ; es. “se trattamento A è migliore di trattamento B…”

- *user_account* : contiene i dati degli utenti



4. Progetto

4.1 Configurazione progetto su eclipse

1. Bisogna per prima cosa cliccare sul progetto IoT
2. modifica in MAVEN project
3. new configuration maven

Adesso qui andrò a creare 3 tipi di maven configurations :

1. FULL BUILD

   Build su tutto IoT (clean validate compile package install). Produce il file war nella cartella target.
2. BUILD BACKEND

   fa la build solo del backend, e questo è utile perchè dopo aver fatto delle modifiche nel codice, prima di lanciare la run configuration, bisogna salvarle facendo questa build (clean compile package install)

3. RUN CONFIGURATION
   esegue la run del progetto su IoT-ui (jetty:run -> lancia l'applicazione su localhost:8080)

4.2 Analisi inserimento dati

L'inserimento potrebbe essere implementato in 2 modi :

1) tramite db (non conviene)

2) metterlo nella piattaforma perchè è già connesso al db, ed utilizzare thread per fare più inserimenti concorrenti nel database (una delle task).

Quando l'applicazione parte ci sono dei thread che fanno da background.

Noi abbiamo il controllo su alcuni thread(10), più correttamente : *Un programma multithread contiene due o più parti che possono essere eseguite contemporaneamente. Ogni parte di un programma di questo tipo viene definita thread e ogni thread definisce un percorso separato di esecuzione. Ne consegue che il multithreading è una forma specializzata di multitasking. In un ambiente multitasking basato sui thread, il thread è l'unità più piccola di codice inviabile. Questo significa che un singolo programma può eseguire contemporaneamente due o più attività.*).

Uno dei file più comuni è WEB XML dell'applicazione (che non andremo a toccare nel nostro caso). Qui si impostano tante cose generiche che accadono quando si avvia l'app.

Tra questi ci sono dei file java che devono essere eseguiti in modalità servlet.

Appservlet è quella che fa partire tutto quanto e crea le sessioni,ma a nel nostro caso mi interessa **ServletOnStartup** per capire quali sono le altre servlet che vengono fatte partire oltre a quella principale.

Quello che sfrutterò è **ProjectCenterSystemTask** (classe) e viene invocato *startProjectCenterSystemTalk* che prende dal database tutti i record che si trovano nella

tabella *ProjectCenterSystem* ,che è importante per il mapping tra id e project center, quindi vede quanti center ha e quanti *system* ha e ne fa partire N.

La tabella System descrive le operazioni di sistema disponibili che possono essere attivate per centro.

Quindi questi sono i thread disponibili.

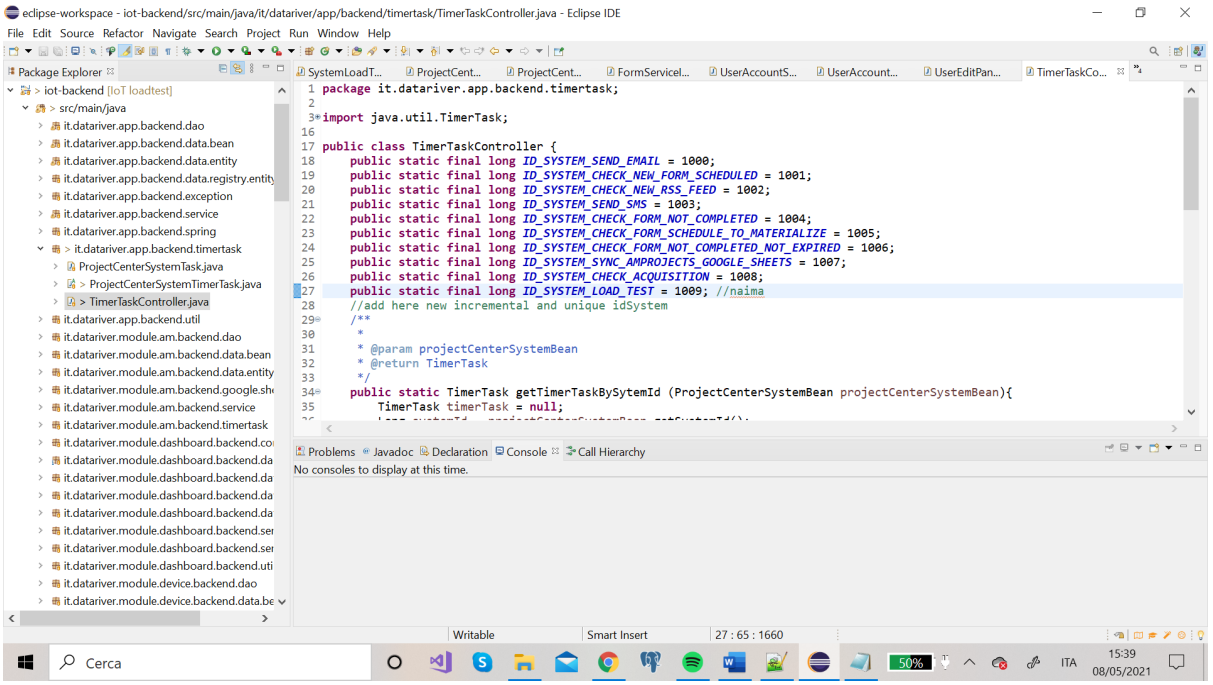
In *project_center_id* ci sono i thread attivati per gli N centri.

Ricapitolando : l'ordine è -> *Webxml, Appservlet* (non mi interessa), *ServletonStartup*.

Le uniche 2 modifiche da fare al codice sono :

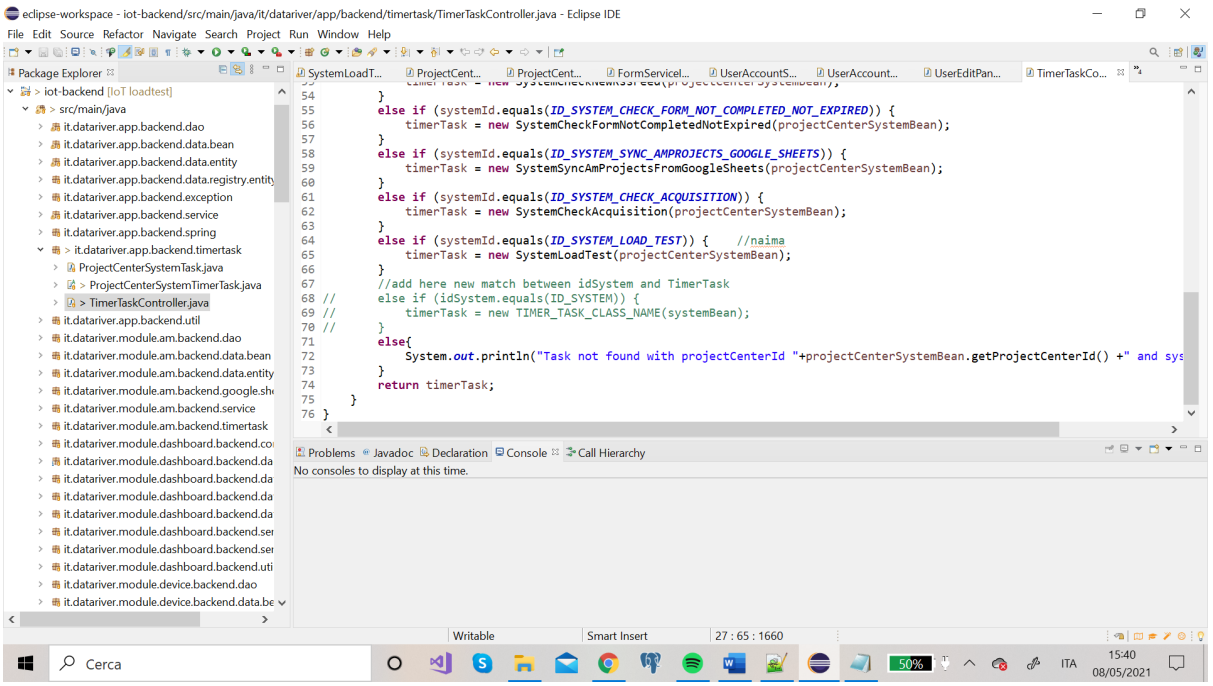
1) aggiungere una op di sistema es.
2) Test di carico inserimento dati CRF e dopo inserirlo in project center id, e settare la frequenza di ripetizione del thread che ho aggiunto (è in millisecondi).

   In *TimeTaskController* c'è un pezzo dove possiamo aggiungere i thread (io aggiungerò la 1009).



Dopo andrò ad aggiungere un ELSE IF uguale a quello commentato a riga 67.

E bisogna andare a cambiare l'id di riferimento e sotto chiamare la classe che andremo a creare per questi test.



Dentro questa classe viene chiamato ogni N millisecondi quello che ho impostato nel db.

I service permettono di fare le insert (concorrenti).

Esempio : *AppService.getUserAccountService().newUserAccount* =

Possiamo modificarlo così: *creationdata* = now; *ownerid* = newlong 0 (sistema); *idUserroll*= lo recupero dal tabella user roll; *useraccountbean* = compilare le informazioni base.

Altre informazioni generali del codice :

- *getLogger().info* -> serve per stampare

  es . *getLogger().info*("aaaa" + eventuale parametro)

- *getSystemBean().getIdSystemProjectCenter()* -> ha una serie di funzionalità

esempio utile per vedere se ciò che ho scritto funziona e mi ritorna il system id: *getLogger().info("getIdSystemProjectCenter "+ getSystemBean().getIdSystemProjectCenter());*

- Ogni volta che faccio una modifica devo lanciare prima il backend e dopo la run oppure utilizzo i Junit (che mi permettono di fare la build di una singola cosa così da evitare di aspettare molto tempo e dopo fare la run.)

Per i form ci sarà molto utile :

- Partiamo dal service : ***ModuleFormService.getFormService().saveOrUpdateFormItemValues()*** -> questo scrive nel *form_item_value* quindi ci sono gli elementi dei questionari.

C'è un vantaggio, perchè c'è una tabella che ha i dati sia di form CRF che dei form mobile.

Il form_id (in form_item_value) e form_item_id sono essenziali come il *form_schedule_id.*

Per gli *audit trial* ci sono 4 dati obbligatori quindi nel java lo possiamo risolvere come prima mettendo longint 0, now, e nel campo *ItemValues* che è una hashmap <string,string>.

Quindi :

- HashMap<String, String> itemValues = new HashMap<String, String>();
- value della map è il value nel db della tabella *form_item_value*
- Nelle hashmap abbiamo KEY, VALUE
- VALUE = value inserito dall'utente (risposta) che verrà messo in *form_item_value*
- KEY = item_label presente in form_item (es. RADIOBUTTON_ID_477)

Nella mia classe dovrò implementare dei service.

Per prima cosa, in *src/test/resources* devo verificare se applicationContext è aggiornato e la cosa importante è che ci sia il mio service all'interno.

Il service del form per esempio non c'è , quindi mi basta copiarlo da applicationContext nella parte di formService.

Dopo devo copiare il mio database properties in *src/test/resources* in *databaseproperties.*

In src/test/java ci sono già tanti test esistenti...

Le classi dao test devono avere il metodo *beforeclass*(uguale per tutti) che fa l'inizializzazione del test e dentro metto i **dao** o i **service** che andrò ad utilizzare.

Creo l'oggetto e dopo lo chiamo appunto facendo *applicationContext.getBean. nome service* e dopo con tasto destro sul metodo posso fare run as JUnit test.

Quindi i dao test hanno la parte con before dove scrivo solo i service e i dao che mi servono.

Dopo posso creare il mio metodo @test dove metto tutti i dati che mi servono e poi posso far partire il test.

Il metodo *.getObjectLog()* mi stampa tutti i campi, quindi itera tutti i metodi degli oggetti bean.

Il primo passo da svolgere verso l'inserimento di dati all'interno della piattaforma e del database è quello di fare funzionare i thread, e poi fare un inserimento.

Partirò dall'inserimento di form_item_value e poi risalirò la corrente fino ad arrivare ad inserire un utente collegato alla sua phase, schedule, form e form_item_values.

4.3 Generazione dati

Eseguendo la query in *FORM_ITEM_VALUES* con un form_schedule_id 27 ho form_id 37;

Nella tabella FORM_SCHEDULE ho con formscheduleid 27 un *phase_id* 5, *pc_phase_value_id 58*, *form_id* 37, *target_user_account_id* 12;

Nella tabella PHASE (visite) in phase_id 5 ho un phase_group_id 1;

Nella tabella PHASE_GROUP in phase_group_id 1 ho General come label;

Nella tabella PROJECT_CENTER_PHASE_FORM , alla riga con phase_id 5 e form_id 37, corrisponde project_center_phase_form_id 26 e c'è un campo di testo che posso compilare da ecplipse,ovvero condition, ma prima di fare questo bisogna settare has_condition a TRUE;

Nella tabella PROJECT_CENTER_PHASE_GROUP_VALUE, con un phase_group_id 1 e un target_user_account_id 12 corrisponde un pcp_group_value_id 74;

Nella tabella PROJECT_CENTER_PHASE_VALUE, con phase_id 5, target_user_account_id 12 e

project_center_id 1(che corrisponde ad ALL CENTERS), corrisponde un pc_phase_value_id 58;

L'attributo target_user_account_id è una foreign key della tabella USER_ACCOUNT, quindi con target_user_account_id 12 abbiamo mail non settata, tutti gli attributi settati a null, username MHS-02-03, label MHS-02-03.

La prova che ho fatto mi va scrivere in Haematology ovvero form_id 37 in tabella form.

Questa è la QUERY che ha che mi collega tutte le tabelle che ho elencato quì sopra però andando a modificare il form DEMOGRAPHY.

```
select ua.id_user_account, ua.username, p.phase_id, p.phase_code, f.form_id,
f.form_code,fit.form_item_id, fit.label_id,fit.label_value

--, fs.*

from user_account ua

JOIN form_schedule fs ON fs.target_user_account_id = ua.id_user_account

JOIN form f ON f.form_id = fs.form_id

JOIN phase p ON p.phase_id = fs.phase_id
```

JOIN project_center_phase_form pcpf ON pcpf.project_center_id = fs.project_center_id AND fs.phase_id = pcpf.phase_id AND pcpf.form_id = fs.form_id

JOIN form_item fi ON fi.form_id = fs.form_id

JOIN form_item_translation fit ON fit.form_item_id = fi.form_item_id AND fit.language = 'en'

WHERE username = 'MHS-02-03'

AND phase_code = 'v1_screening'

AND form_code = 'CRF_DEMOGRAPHY'

ORDER BY pcpf.form_phase_ordinal

Se voglio continuare a modificare il form HAEMATOLOGY :

select ua.id_user_account, ua.username,fs.form_schedule_id, p.phase_id, p.phase_code, f.form_id, f.form_code,fi.item_label,fit.form_item_id, fit.label_id,fit.label_value,fi.*

--, fs.*

from user_account ua

JOIN form_schedule fs ON fs.target_user_account_id = ua.id_user_account

JOIN form f ON f.form_id = fs.form_id

JOIN phase p ON p.phase_id = fs.phase_id

JOIN project_center_phase_form pcpf ON pcpf.project_center_id = fs.project_center_id AND fs.phase_id = pcpf.phase_id AND pcpf.form_id = fs.form_id

JOIN form_item fi ON fi.form_id = fs.form_id

JOIN form_item_translation fit ON fit.form_item_id = fi.form_item_id AND fit.language = 'en'

WHERE username = 'MHS-02-03'

AND phase_code = 'v1_screening'

AND form_code = 'CRF_HAEMATOLOGY'

ORDER BY pcpf.form_phase_ordinal

I prossimi passi da eseguire sono quelli di fare delle prove di riempimento dei vari form.

Poi passare a creare nuovi form nelle schedule (la vista dove si vedono tutti i form compilati per paziente e visita),poi phase, utente ecc.

Dopo aver fatto tutto questo da codice e aver fatto le nostre prove, allora ci dovremmo occupare dell'inserimento dei dati da prendere da un foglio di testo, o da un csv, o da un database già popolato e far si che nel nostro codice avvenga l'inserimento di tot pazienti/form/ecc.

In realtà, in seguito si vedrà che abbiamo optato per un inserimento dati prettamente da codice, con valori generati casualmente in base a ciò che ogni dato accetta.

L'unica cosa che andremo a prendere dal db sarà il range (dateMin,dateMax) su cui generare valori di date random da assegnare ai campi Date della nostra piattaforma.

Ho suddiviso la generazione di dati random in 2 macro step. *(segue)*

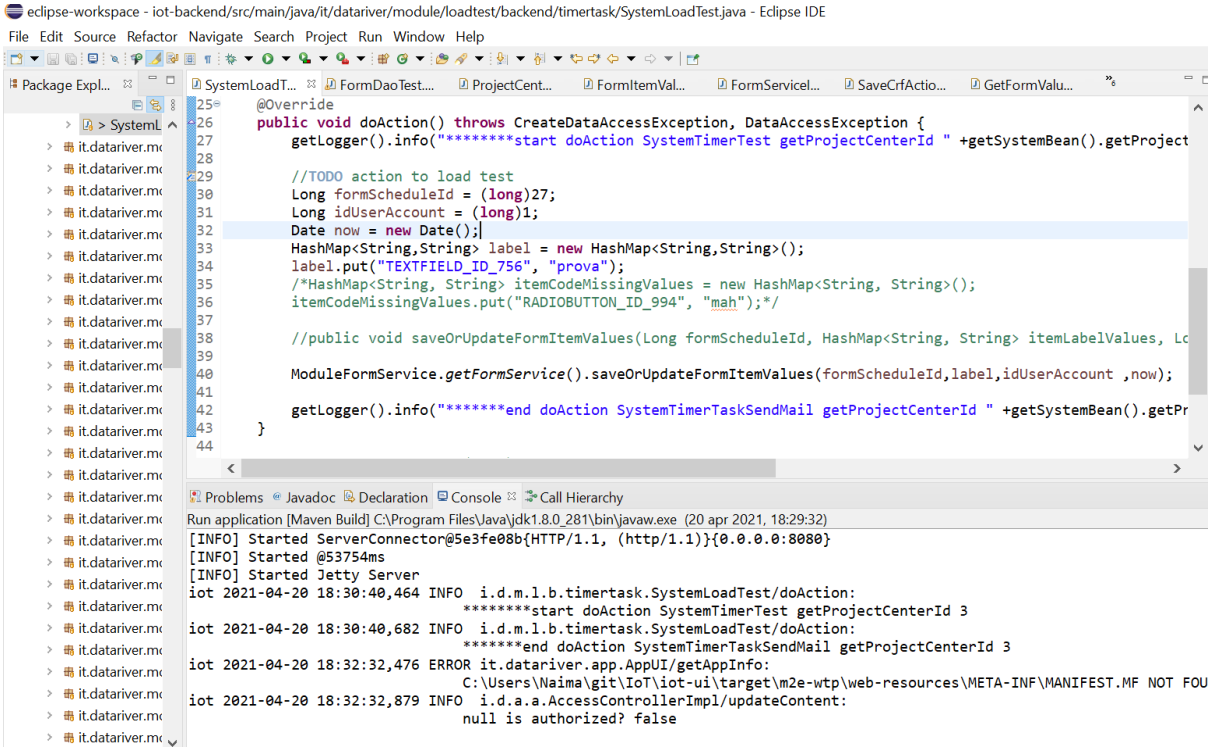
4.3.1 Primo step : generazione dati con valori piazzati nel codice

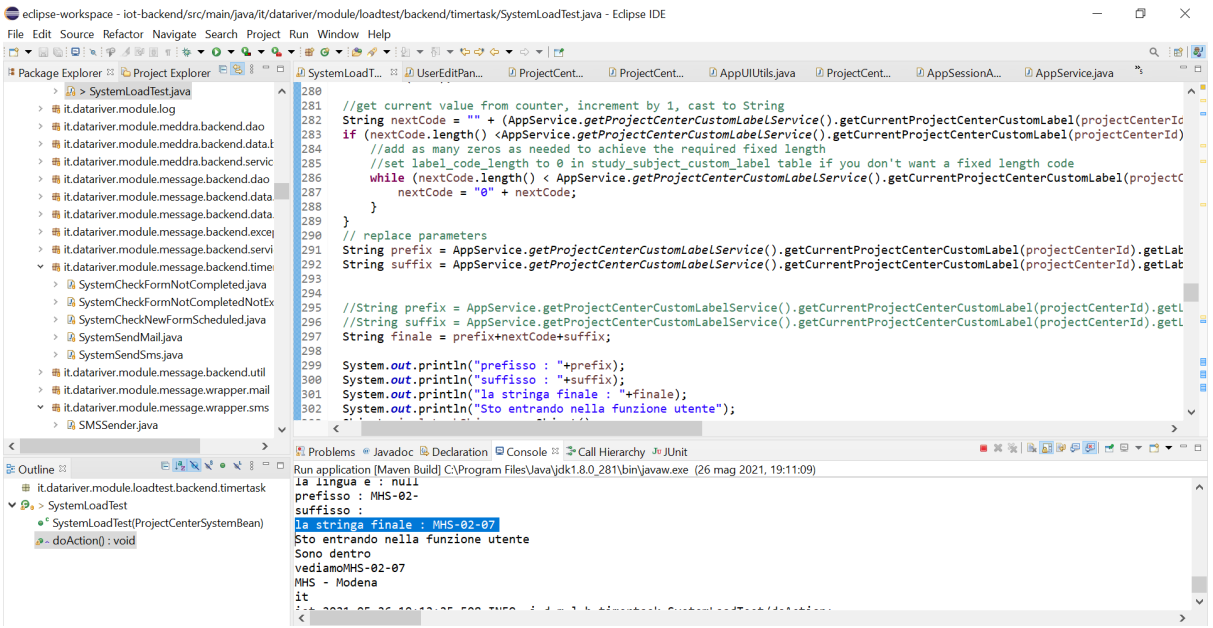
Come primo passo, ho eseguito tutte le mie prove per l'inserimento dati riguardanti nuovo paziente, con relativi *phaseGroup,phase,form e formItemValues,* con dei dati fissati nel codice.

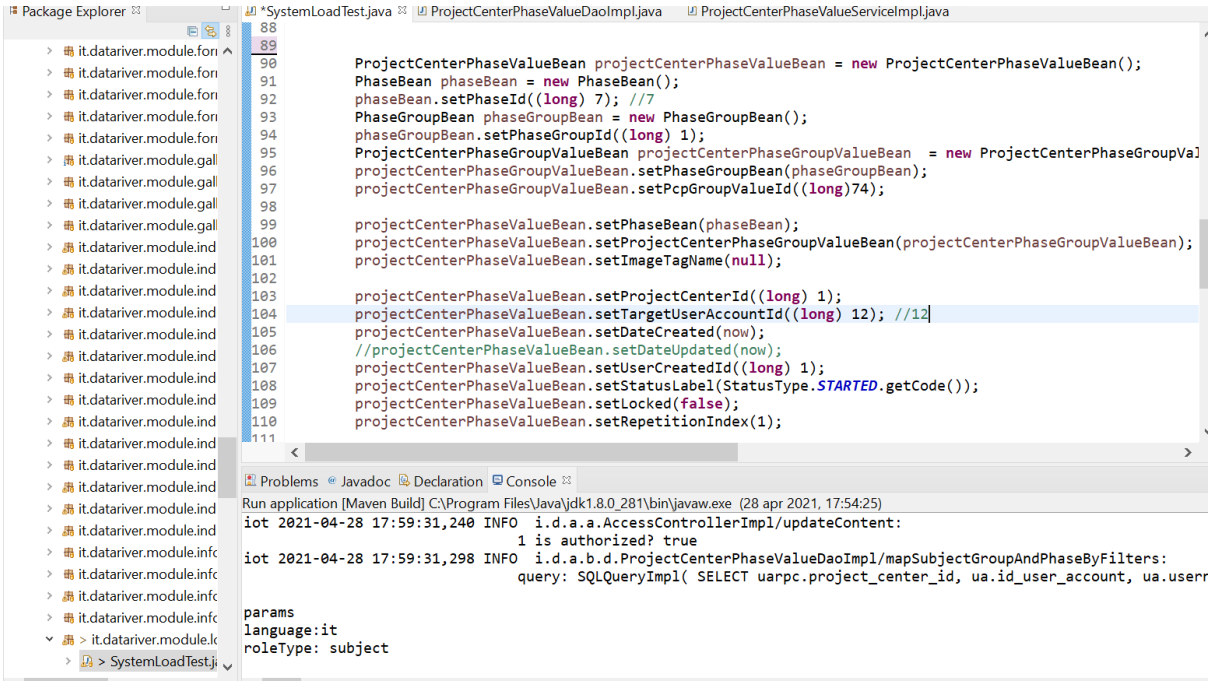
Questo è stato fatto per aiutarmi nella comprensione dei vari componenti della piattaforma nel codice.

Con dei dati fissati nel codice infatti, potevo basare le mie prove su dati che potevo visualizzare sia dal db che dalla piattaforma e questo mi ha aiutata a capire meglio come tutte le cose fossero collegate.

Qui sotto vi sono degli screenshot di esempio sul codice che ho scritto:


eclipse-workspace - iot-backend/src/main/java/it/datariver/module/loadtest/backend/timertask/SystemLoadTest.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Expl...
SystemLoadT...
FormDaoTest...
ProjectCent...
FormItemVal...
FormServicel...
SaveCrfActio...
GetFormValu...
@Override
public void doAction() throws CreateDataAccessException, DataAccessException {
getLogger().info("********start doAction SystemTimerTest getProjectCenterId " +getSystemBean().getProject
//TODO action to load test
Long formScheduleId = (long)27;
Long idUserAccount = (long)1;
Date now = new Date();
HashMap<String,String> label = new HashMap<String,String>();
label.put("TEXTFIELD_ID_756", "prova");
/*HashMap<String, String> itemCodeMissingValues = new HashMap<String, String>();
itemCodeMissingValues.put("RADIOBUTTON_ID_994", "mah");*/
//public void saveOrUpdateFormItemValues(Long formScheduleId, HashMap<String, String> itemLabelValues, Lo
ModuleFormService.getFormService().saveOrUpdateFormItemValues(formScheduleId,label,idUserAccount ,now);
getLogger().info("*******end doAction SystemTimerTaskSendMail getProjectCenterId " +getSystemBean().getPr
}
Problems Javadoc Declaration Console Call Hierarchy
Run application [Maven Build] C:\Program Files\Java\jdk1.8.0_281\bin\javaw.exe (20 apr 2021, 18:29:32)
[INFO] Started ServerConnector@5e3fe08b{HTTP/1.1, (http/1.1)}{0.0.0.0:8080}
[INFO] Started @53754ms
[INFO] Started Jetty Server
iot 2021-04-20 18:30:40,464 INFO i.d.m.l.b.timertask.SystemLoadTest/doAction:
********start doAction SystemTimerTest getProjectCenterId 3
iot 2021-04-20 18:30:40,682 INFO i.d.m.l.b.timertask.SystemLoadTest/doAction:
*******end doAction SystemTimerTaskSendMail getProjectCenterId 3
iot 2021-04-20 18:32:32,476 ERROR it.datariver.app.AppUI/getAppInfo:
C:\Users\Naima\git\IoT\iot-ui\target\m2e-wtp\web-resources\META-INF\MANIFEST.MF NOT FOU
iot 2021-04-20 18:32:32,879 INFO i.d.a.a.AccessControllerImpl/updateContent:
null is authorized? false


eclipse-workspace - iot-backend/src/main/java/it/datariver/module/loadtest/backend/timertask/SystemLoadTest.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
Project Explorer
SystemLoadTest.java
it.datariver.module.log
it.datariver.module.meddra.backend.dao
it.datariver.module.message.backend.dao
it.datariver.module.message.backend.util
it.datariver.module.message.wrapper.mail
it.datariver.module.message.wrapper.sms
SMSSender.java
Outline
it.datariver.module.loadtest.backend.timertask
SystemLoadTest
SystemLoadTest(ProjectCenterSystemBean)
doAction() : void
//get current value from counter, increment by 1, cast to String
String nextCode = "" + (AppService.getProjectCenterCustomLabelService().getCurrentProjectCenterCustomLabel(projectCenterId
if (nextCode.length() <AppService.getProjectCenterCustomLabelService().getCurrentProjectCenterCustomLabel(projectCenterId)
//add as many zeros as needed to achieve the required fixed length
//set label_code_length to 0 in study_subject_custom_label table if you don't want a fixed length code
while (nextCode.length() < AppService.getProjectCenterCustomLabelService().getCurrentProjectCenterCustomLabel(projectC
nextCode = "0" + nextCode;
}
}
// replace parameters
String prefix = AppService.getProjectCenterCustomLabelService().getCurrentProjectCenterCustomLabel(projectCenterId).getLab
String suffix = AppService.getProjectCenterCustomLabelService().getCurrentProjectCenterCustomLabel(projectCenterId).getLab
//String prefix = AppService.getProjectCenterCustomLabelService().getCurrentProjectCenterCustomLabel(projectCenterId).getL
//String suffix = AppService.getProjectCenterCustomLabelService().getCurrentProjectCenterCustomLabel(projectCenterId).getL
String finale = prefix+nextCode+suffix;
System.out.println("prefisso : "+prefix);
System.out.println("suffisso : "+suffix);
System.out.println("la stringa finale : "+finale);
System.out.println("Sto entrando nella funzione utente");
Problems Javadoc Declaration Console Call Hierarchy JUnit
Run application [Maven Build] C:\Program Files\Java\jdk1.8.0_281\bin\javaw.exe (26 mag 2021, 19:11:09)
la lingua e : null
prefisso : MHS-02-
suffisso :
la stringa finale : MHS-02-07
Sto entrando nella funzione utente
Sono dentro
vediamoMHS-02-07
MHS - Modena
it

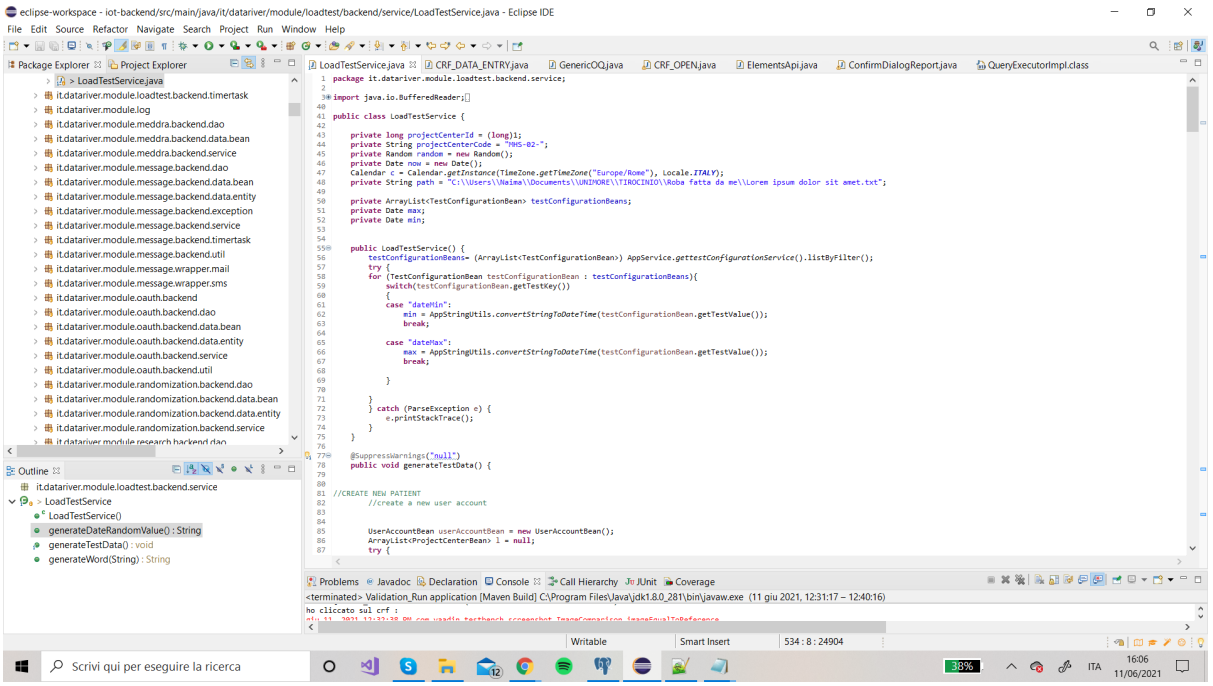


4.3.2 Secondo step : generazione dati con valori random

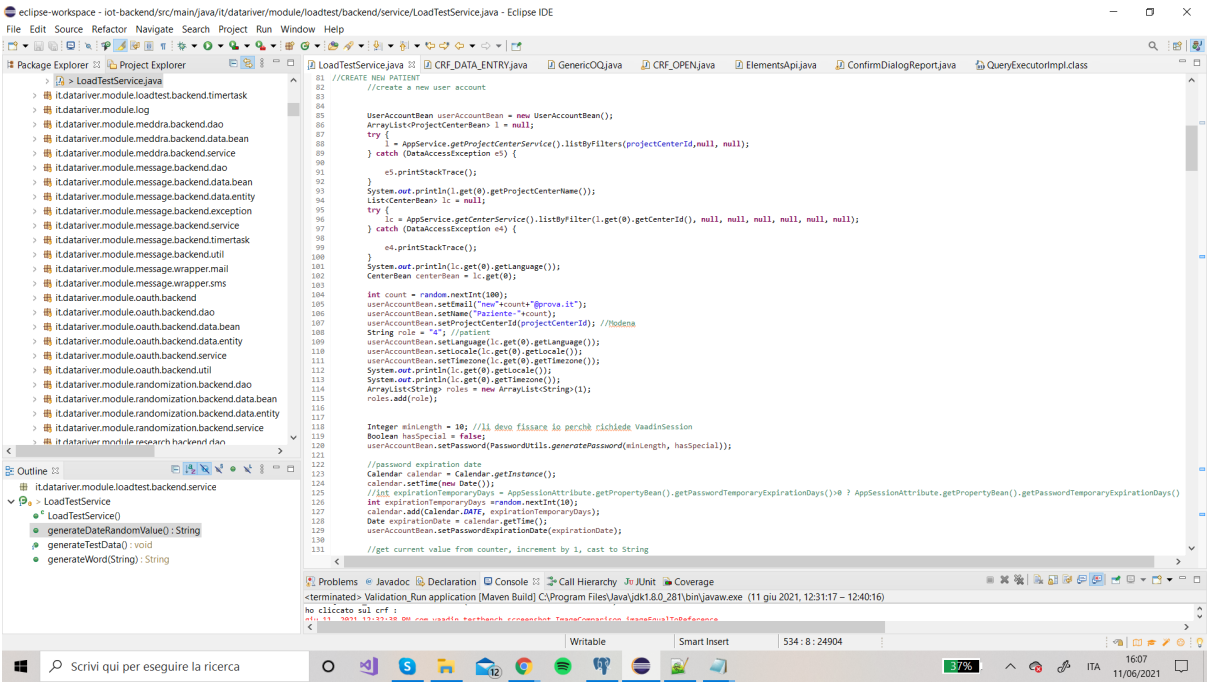
Successivamente, nello step 2, prima di effettuare l'effettiva generazione dei dati casuali, si è deciso di organizzare tutta questa parte in un altro file java a cui abbiamo dato il nome di "*LoadTestService*" in quanto tutte le prove che avevo precedentemente svolto in *SystemLoadTest* in realtà va messo nella parte di **ui** perché *SystemLoadTest* fa parte del backend, e quindi ci dovrebbe essere solo la parte di connessione al db.

Qui sotto è riportato il codice sviluppato :

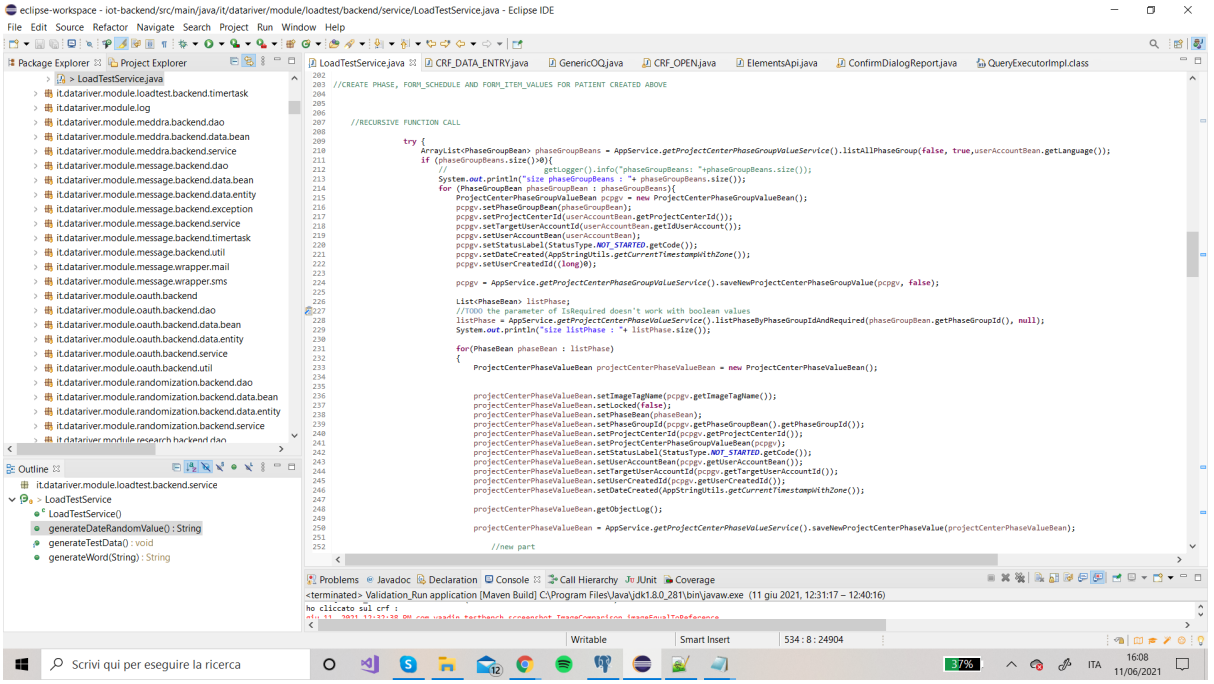
1) Prendiamo i dati dalla tabella *test_configuration* di cui parlerò a breve

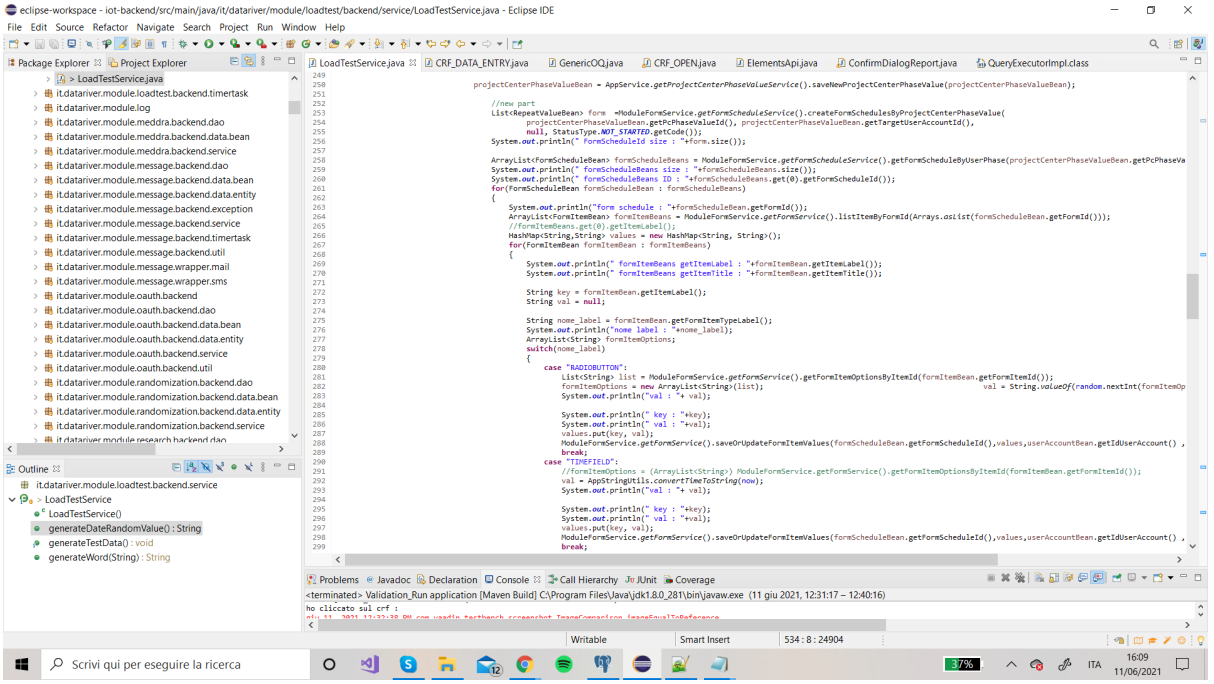


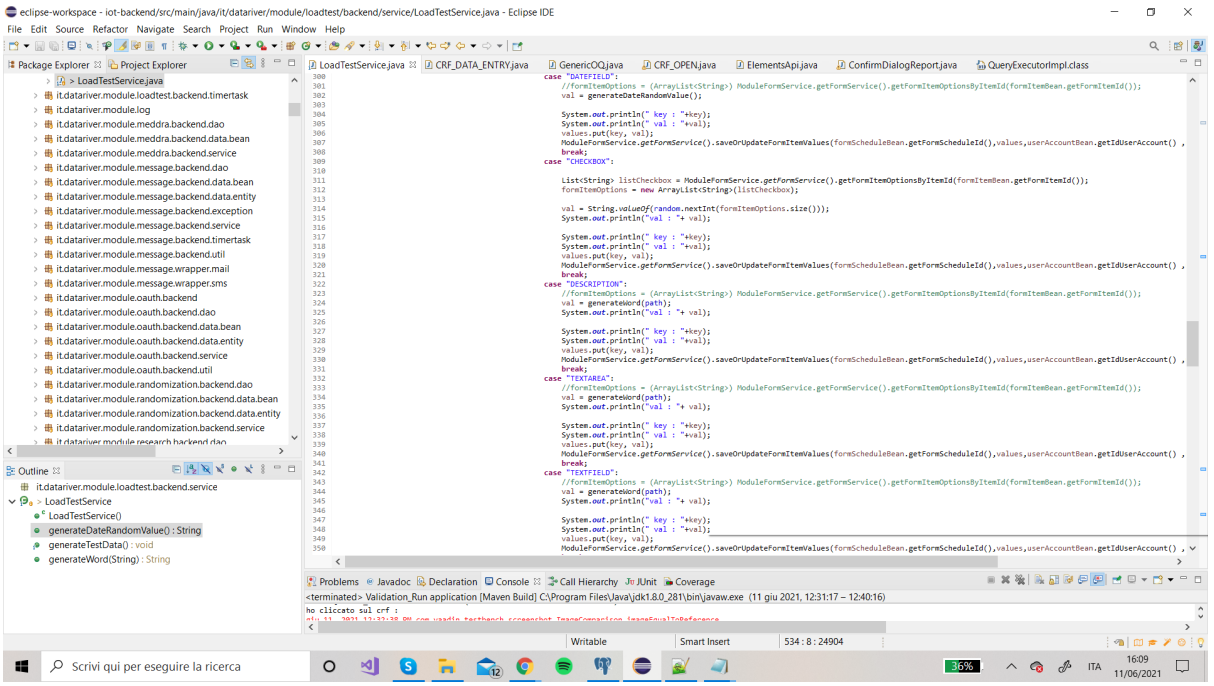
2) Creazione paziente



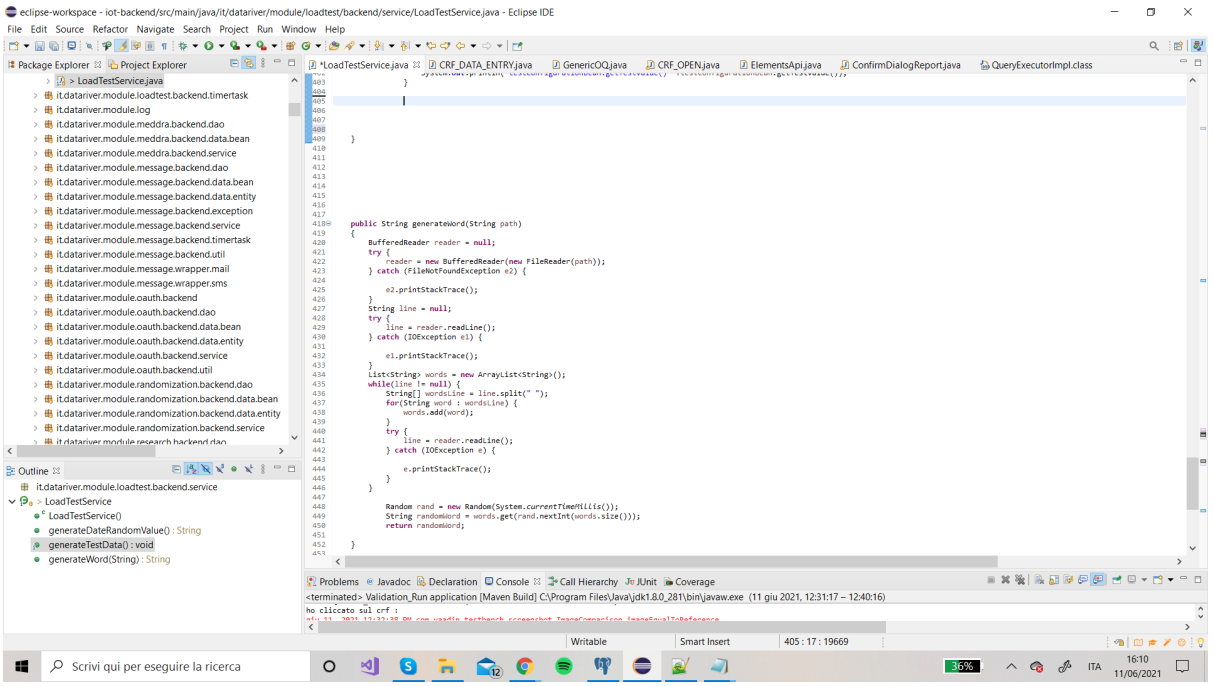
3) Creazione di phase, *phase_group,form_schedule,form_item_values* tutti associati al paziente creato nel punto precedente


eclipse-workspace - iot-backend/src/main/java/it/datariver/module/loadtest/backend/service/LoadTestService.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
Project Explorer
LoadTestService.java
CRF_DATA_ENTRY.java
GenericOQ.java
CRF_OPEN.java
ElementsApi.java
ConfirmDialogReport.java
QueryExecutorImpl.class
Outline
LoadTestService()
generateDateRandomValue() : String
generateTestData() : void
generateWord(String) : String
Problems
Javadoc
Declaration
Console
Call Hierarchy
JUnit
Coverage
Writable
Smart Insert
534 : 8 : 24904
Scrivi qui per eseguire la ricerca
ITA
16:08
11/06/2021


eclipse-workspace - iot-backend/src/main/java/it/datariver/module/loadtest/backend/service/LoadTestService.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
Project Explorer
LoadTestService.java
CRF_DATA_ENTRY.java
GenericOQ.java
CRF_OPEN.java
ElementsApi.java
ConfirmDialogReport.java
QueryExecutorImpl.class
Outline
LoadTestService()
generateDateRandomValue() : String
generateTestData() : void
generateWord(String) : String
Problems
Javadoc
Declaration
Console
Call Hierarchy
JUnit
Coverage
Writable
Smart Insert
534 : 8 : 24904
Scrivi qui per eseguire la ricerca
ITA
16:09
11/06/2021



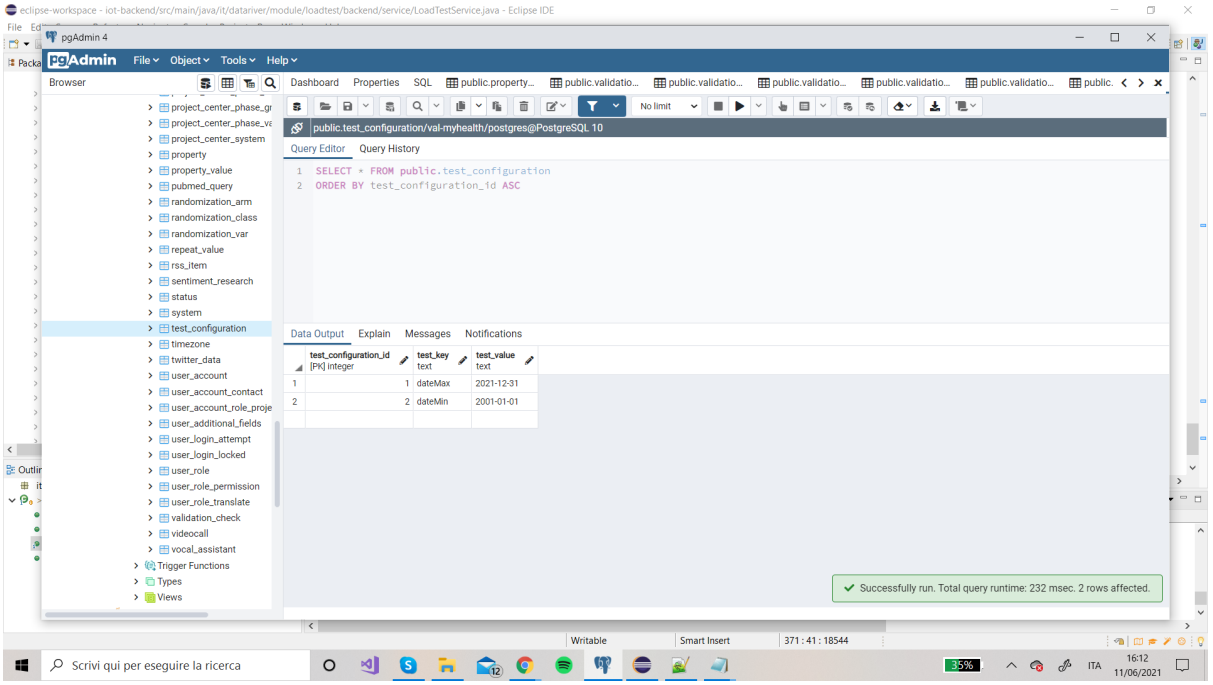
4) Funzioni di supporto



Dopo aver finito tutta la generazione di dati random e di acquisizione dati dal db per creare nuovi pazienti/form/ecc. bisognava passare alla creazione del db da cui prendere i dati.

Ma, come già anticipato in precedenza, dal momento in cui inseriamo tutti i dati da codice , l'unica cosa che al momento potrebbe servirci è la data; quindi andremo per comodità a creare semplicemente una tabella del tipo "key-value" dove andremo ad inserire il range di data che vogliamo utilizzare.

Questa soluzione è ottima perchè configurabile, ovvero, nel momento in cui io volessi fare altri test e mi dovesse servire prendere qualcosa da db, mi basterebbe aggiungere una colonna a questa tabella e modificare di conseguenza i file entity/bean e dao.



Dopo aver creato questa tabella nel db(CREATE TABLE test_configuration (test_configuration_id SERIAL PRIMARY KEY, date_min TEXT, date_max TEXT); ALTER TABLE test_configuration OWNER TO clinica;) allora possiamo passare alla creazione dei file entity/bean/dao.

4.3.3 Entity/Bean/Dao

PER ENTITY(prenderò come esempio quello di Center che è simile)

-il file si deve chiamare come la tabella

-aggiungere una chiave long (che nel db deve essere serial)

-poi i due attributi che ho nella tabella

-tasto destro, source, generate getter/setter

-dopo aggiungere ai get/set con la notazione solo sopra i get

PER BEAN(prendere come esempio quello di Center che è simile)

-Oltre ai get/set , ha dei metodi per prendere una mappa e trasformarla nel bean nella mappa cerchi il nome della chiave e la assegni (valueof)

PER DAO/SERVICE(prendere come esempio quello di Center che è simile)

-Ho le query che voglio fare (nel mio caso solo select all)

-get all properties

-Da fare quello normale(solo metodi dichiarati) e quello impl(fare un metodo list all dove appunto genero la query)

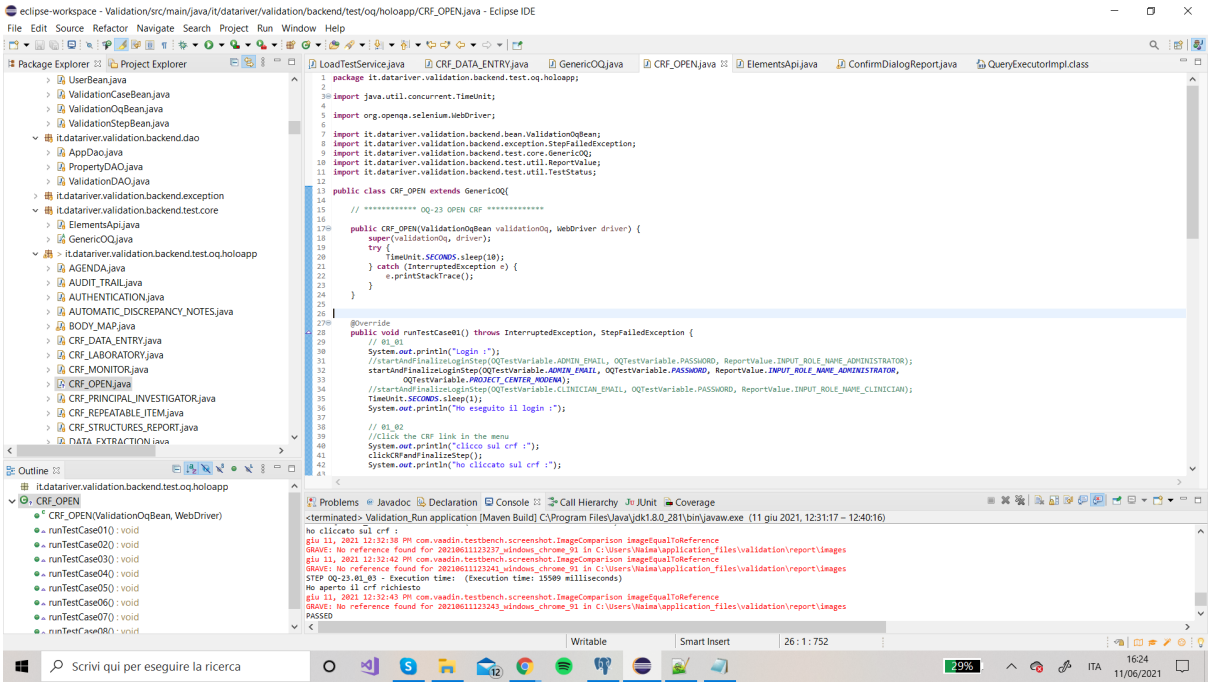
Dopo aver verificato che la generazione dei dati avvenga in modo corretto, allora dovrò ,come ultimo passo, mettere giù tutte le modifiche fatte al codice su Cygwin (git status, git branch -a, git fetch, ecc).

4.4 Validation

Dopo essere riuscita ad implementare una soluzione che permetta di generare dati casuali a partire dalla generazione del paziente, adesso posso focalizzarci sull'ultima parte dell'attività di tirocinio , ovvero la validazione con VaadinTestBanch.

Per fare questo i passi che ho seguito sono stati :

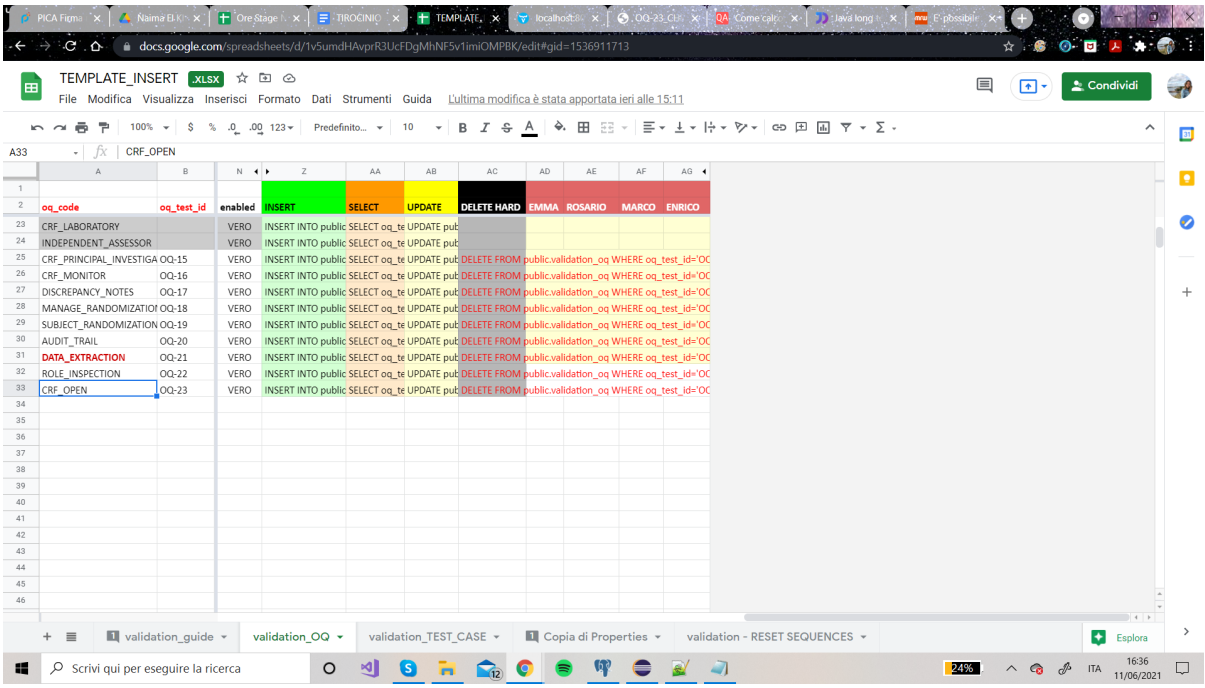
1) Creare un OQ :

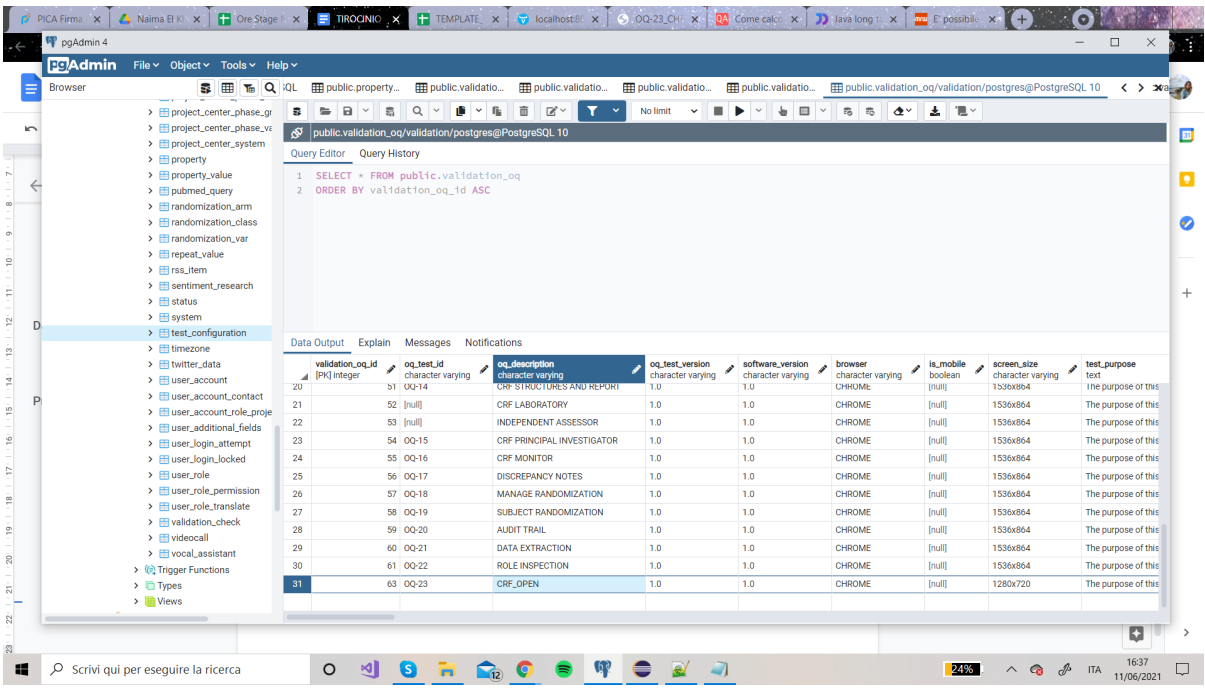


2) Basarsi su un OQ già creato oppure prendere d'esempio OQ-template

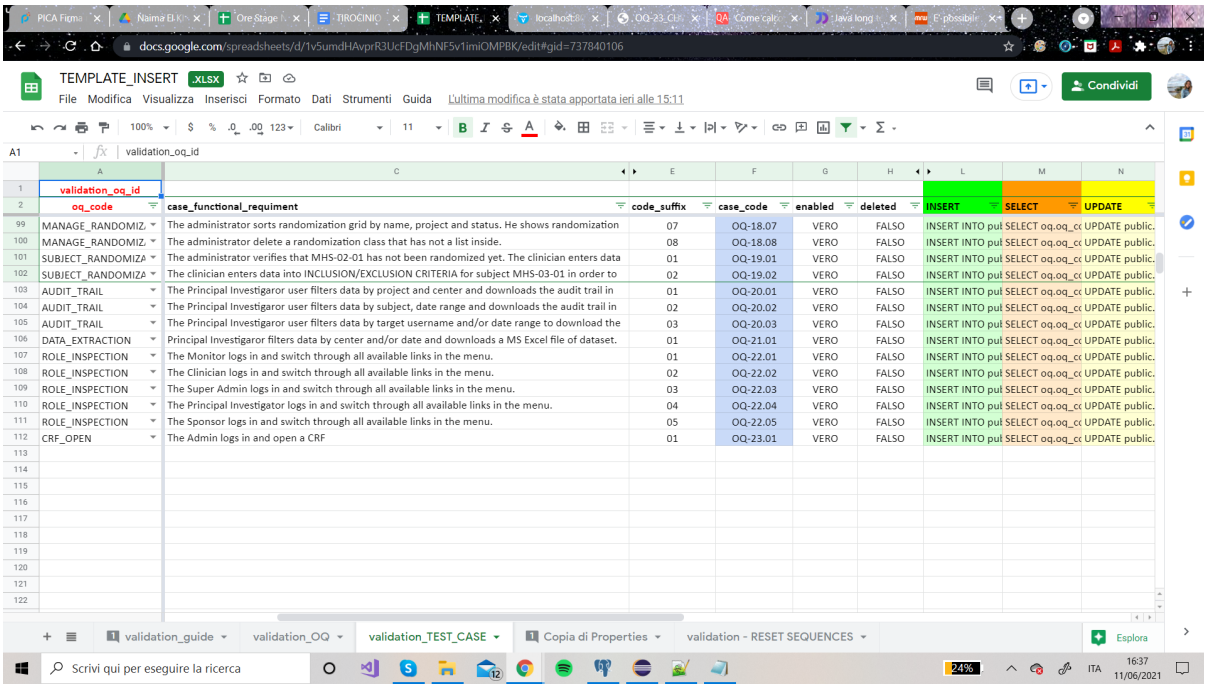
3) Cosa fondamentale è creare il *validation_oq* nel database

   a) farlo direttamente dall'excel che ci ha passato Emma e da lì copiare l'istruzione INSERT, autogenerata, nel nostro db di Validation.



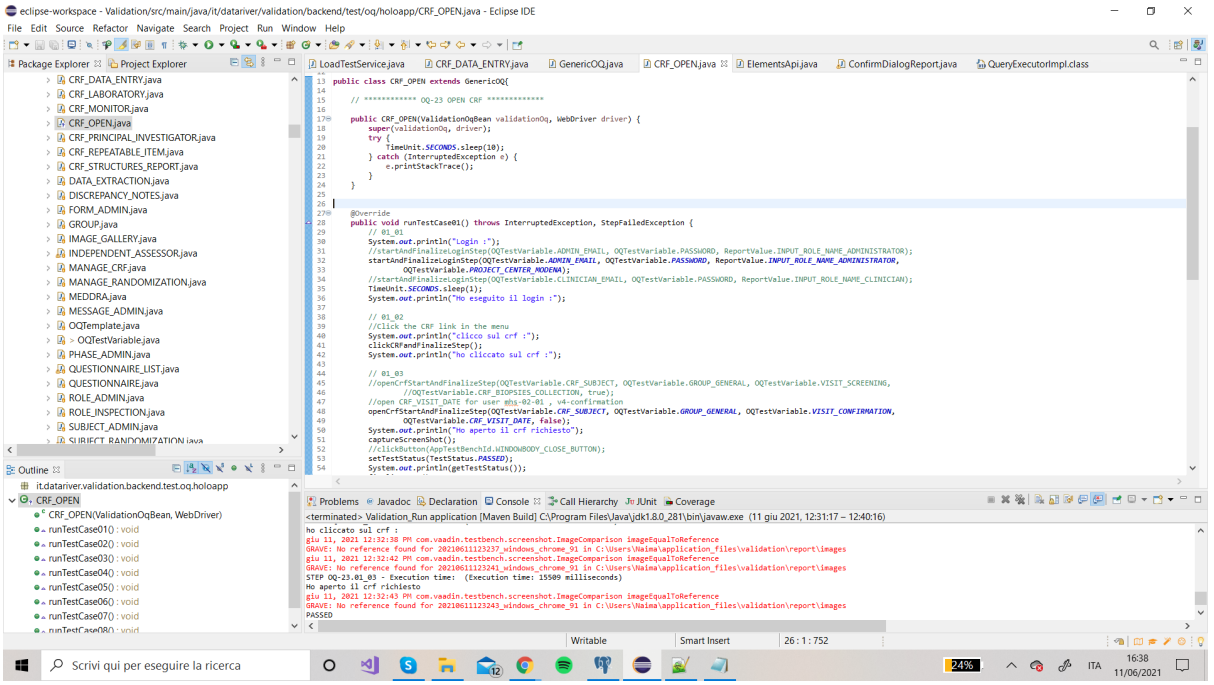


b) fare la stessa cosa per il test case.

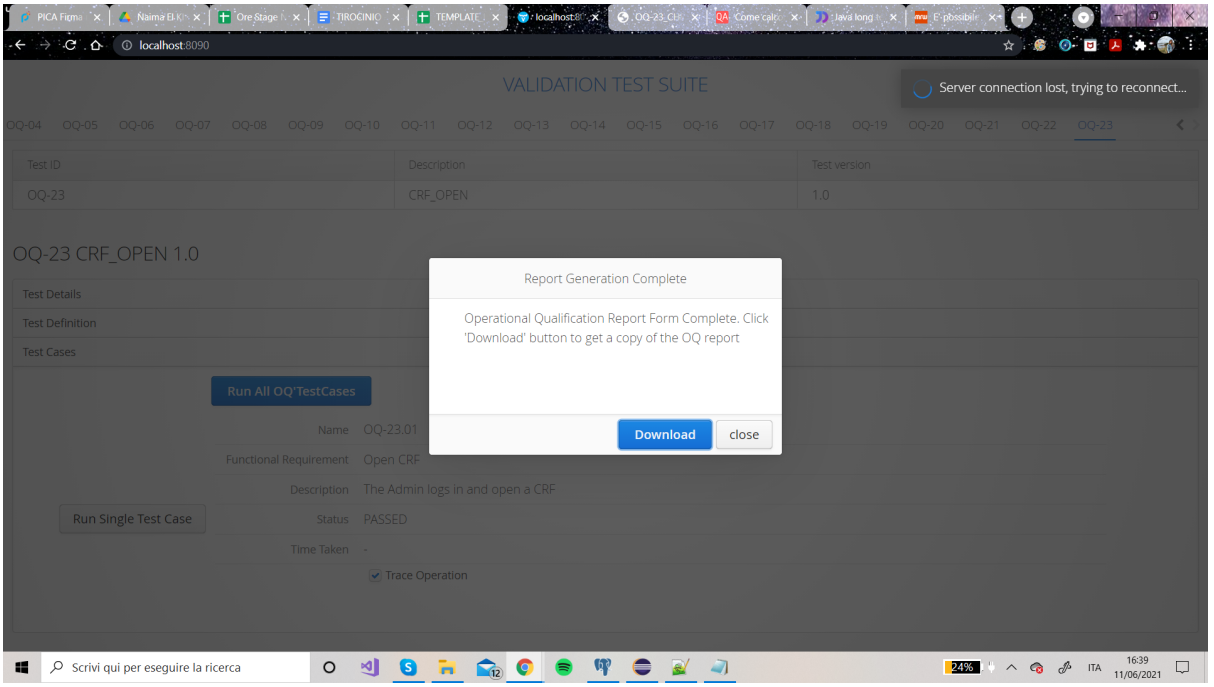


4) Possiamo passare alla scrittura del codice.
5) Nel mio caso ho dovuto sviluppare un solo Test Case ovvero quello dove svolgerò le operazioni di :

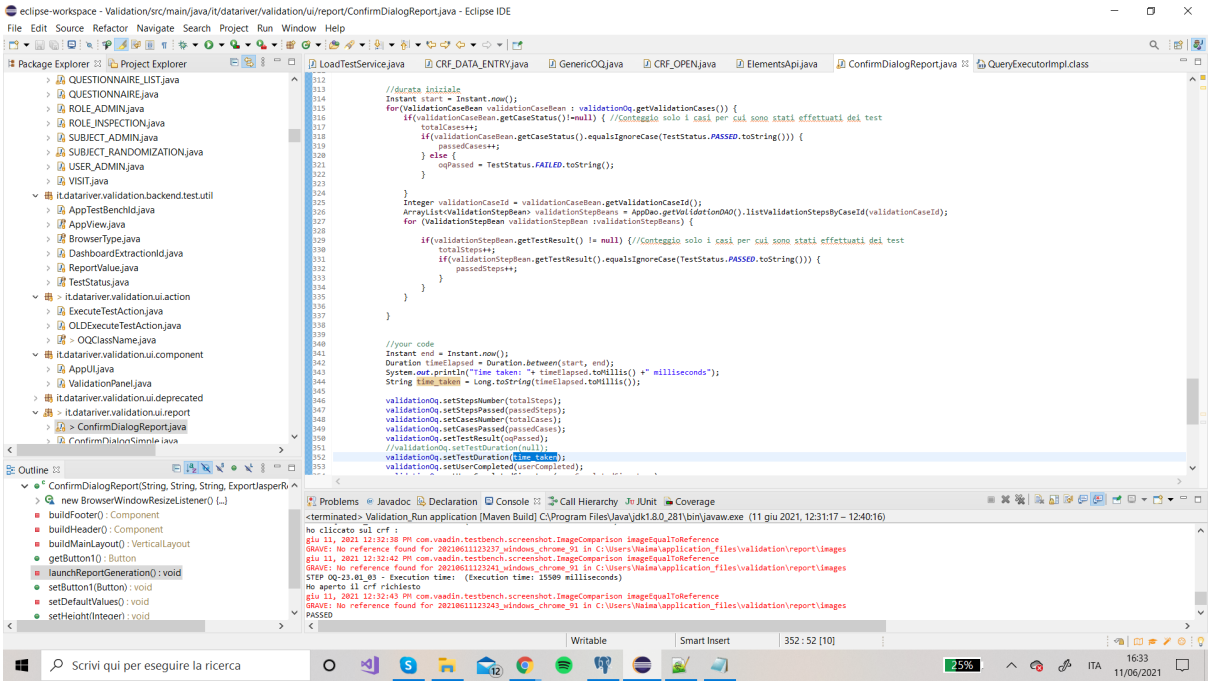
- login
- click del link CRF nel menu
- mi posiziono sul paziente al *phaseGroup* e *form* che ho deciso da codice e apro il *phaseGroup* V4 Confirmation Visit e il form VisitDate.

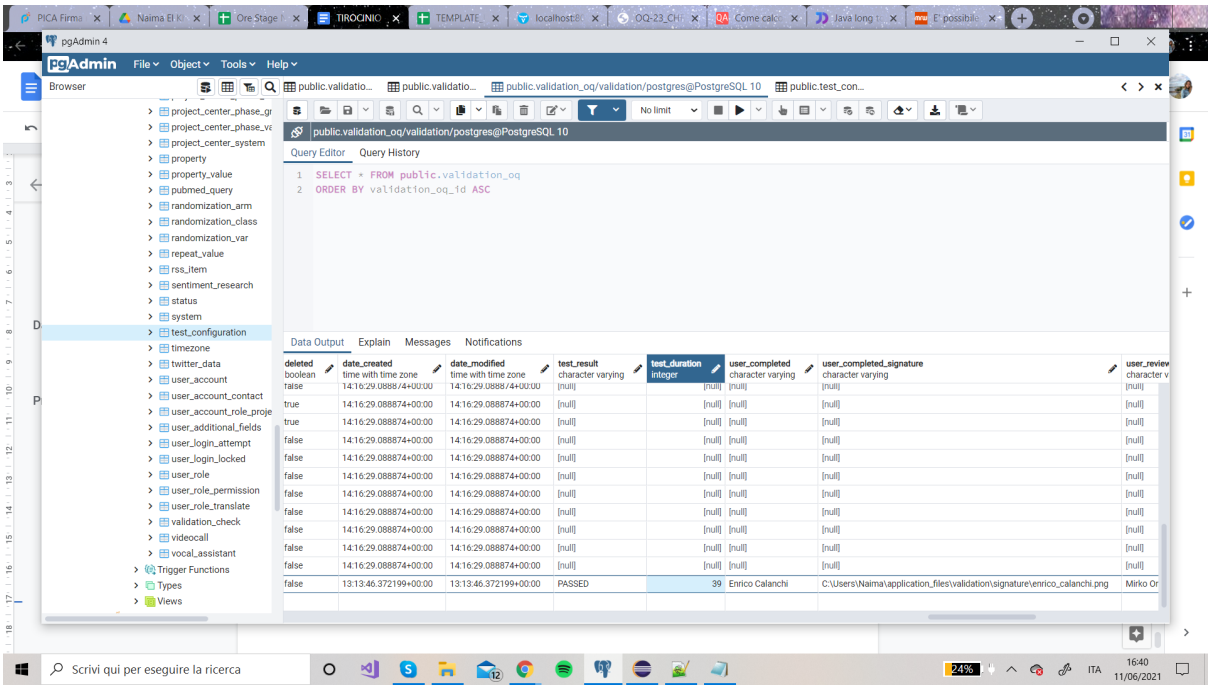


6) Tutto questo mi serve per effettuare dei test sul tempo impiegato a svolgere il Test Case e questo andrà poi riportato come output nel mio Report PDF.



7) Per aggiungere il tempo di esecuzione (che di norma è settato a null) è stato necessario andare ad agire sul file ConfirmDialogReport.java nei due for annidati e fare la differenza dei tempi impiegati all'inizio e alla fine del for e poi settarli a *test_duration*.





8) Per aggiungere il tempo impiegato che appunto corrisponde al *test_duration*, è stato necessario agire sul file *repor_template_conclusions.jrxlm* e in particolare utilizzare lo strumento *iReport*.
9) Purtroppo in questo modo si riusciva a vedere solo il tempo totale e non il tempo per ogni test case; quest'ultimo è utile per capire qual è la fase che dà maggiori problemi nei 2 casi. Per esempio, se con 10 pazienti per far tutto impiego 10 secondi di cui 3 solo per il secondo test case, mentre con 100 pazienti ne impiego 25 di cui 15 solo per il secondo test case allora abbiamo così individuato un possibile problema nel secondo test case.

   Proprio per questo motivo, ho cambiato approccio non agendo più su i due for annidati ma sul file *GenericOQ* dove si trovano tutte le funzionalità che richiamo nel mio *CRF_OPEN*.

   Qui utilizzo la funzione *finalizeStep()* che viene chiamata alla fine di ogni funzione della CRF e qui calcolo la differenza tra lo *StartTime* e il tempo corrente e con un *appendOutput* riporto il tempo di esecuzione del singolo step nel report; per il tempo totale ho agito similmente ma sul *CRF_OPEN* alla fine dei miei step.

   In questo modo nel Report avrò nella pagina riassuntiva, per ogni step del test case, oltre agli screenshots, anche il tempo impiegato per ogni singolo step.

   Questo, come detto in precedenza, ci permetterà di valutare la criticità dello step/operazione in entrambe le situazioni (es. 100 pazienti vs 1000 pazienti), e di poter agire di conseguenza sapendo a priori su cosa concentrarsi.

4.4.1 Test Result

Il test che ho eseguito è stato quello di lanciare l'applicazione di validazione ,con il test case realizzato (che ho esposto al punto 9 del capitolo 4.4), in due casi :

- ➔ selezionando il paziente 1
- ➔ selezionando il paziente 305.

Il test ha prodotto questi due report :

- Selezionando il paziente 1, i tempi sono stati :
    - 14667 secondi (PRIMO STEP)
    - 14720 secondi (SECONDO STEP)
    - 139886 secondi (TERZO STEP)

  Per un totale di : 169,273 secondi, cioè 2,821216667 minuti.

DataRiver
open source data management

Operational Qualification Report Form

Test ID	Description	Test Version
OQ-23	CRF_OPEN	1.0

Validation Steps

Step	Test Step	Expected Result	Pass/Fail	Reference
OQ-23.01_01	Insert administrator credentials, select a center and log in	Result: access with role administrator to project center MHS - Modena	PASSED	--INPUT-- email: admin@datariver.it password: Datariver2020 Project center: MHS - Modena --OUTPUT-- Login successful Execution time: 14667 seconds --SCREENSHOT-- OQ-23.01_01_IMG_1 OQ-23.01_01_IMG_2 OQ-23.01_01_IMG_3
OQ-23.01_02	Click the CRF link in the menu.	Result: access to the CRF page	PASSED	--INPUT-- Click on CRF --OUTPUT-- Access to the CRF page Execution time: 14720 seconds --SCREENSHOT-- OQ-23.01_02_IMG_1
OQ-23.01_03	Open Visit Date CRF of V4 Confirmation visit.	Result: Visit Date is opened in view mode.	PASSED	--INPUT-- CRF selected: Visit Date --OUTPUT-- CRF opened in view mode. Execution time: 139886 seconds TOTAL TIME :169273 --SCREENSHOT-- OQ-23.01_03_IMG_1 OQ-23.01_03_IMG_2 OQ-23.01_03_IMG_3

- Selezionando il paziente 305, invece, i tempi sono stati :

- 13206 secondi (PRIMO STEP)
- 16192 secondi (SECONDO STEP)
- 262565 secondi (TERZO STEP)

Per un totale di 291,963 secondi, cioè 4,86605 minuti.

DataRiver
open source data management

Operational Qualification Report Form

Test ID	Description	Test Version
OQ-23	CRF_OPEN	1.0

Validation Steps

Step	Test Step	Expected Result	Pass/Fail	Reference
OQ-23.01_01	Insert administrator credentials, select a center and log in	Result: access with role administrator to project center MHS - Modena	PASSED	--INPUT-- email: admin@datariver.it password: Datariver2020 Project center: MHS - Modena --OUTPUT-- Login successful Execution time: 13206 seconds --SCREENSHOT-- OQ-23.01_01_IMG_1 OQ-23.01_01_IMG_2 OQ-23.01_01_IMG_3
OQ-23.01_02	Click the CRF link in the menu.	Result: access to the CRF page	PASSED	--INPUT-- Click on CRF --OUTPUT-- Access to the CRF page Execution time: 16192 seconds --SCREENSHOT-- OQ-23.01_02_IMG_1
OQ-23.01_03	Open Visit Date CRF of V4 Confirmation visit.	Result: Visit Date is opened in view mode.	PASSED	--INPUT-- CRF selected: Visit Date --OUTPUT-- CRF opened in view mode. Execution time: 262565 seconds TOTAL TIME :291963 --SCREENSHOT-- OQ-23.01_03_IMG_1 OQ-23.01_03_IMG_2 OQ-23.01_03_IMG_3

I tempi a confronto:

Paziente 1	*Paziente 305*
14667	13206
14720	16192
139886 (2,331433333 minuti)	262565 (4,376083333 minuti)

La cosa che salta più all'occhio è quella che accade nel TERZO STEP;
come si legge dalla tabella, con il terzo paziente si sono impiegati 4,376083333 dei 4,86605 minuti solo per quest'ultimo step, con una differenza di circa 2 minuti in più rispetto al primo test con il paziente 1.
Questo ci mostra che le operazioni svolte nel terzo step non sono performanti in presenza di più soggetti arruolati e quindi indica uno dei potenziali problemi su cui concentrarsi in futuro per poter migliorare le prestazioni e la fluidità della piattaforma.

5. Conclusioni e possibili estensioni future

La soluzione è stata implementata in modo flessibile per permettere future estensioni per migliorare le prestazioni, i tempi e i dati utilizzati.

Ad esempio: potrei voler generare solo pazienti con determinati dati/caratteristiche o solo alcuni tipi di moduli, ecc.

Come ho accennato in precedenza, il codice è molto flessibile e permette di fare solo poche modifiche in termini di codice per adattarlo agli usi futuri; e l'aiuto della tabella *test_configuration* permette che ciò accada.

POSSIBILI ESTENSIONI FUTURE :

- aggiungere più test case in base alle esigenze della compagnia
- gestire più centri
- multilingua

6. Sitografia

(1) https://vaadin.com/docs/v14/tools/testbench/overview
(2) https://it.wikipedia.org/wiki/Git_(software)
(3) https://www.postgresql.org/about/
(4) https://it.wikipedia.org/wiki/Cygwin
(5) https://it.wikipedia.org/wiki/Eclipse_(informatica)
(6) https://wiki.eclipse.org/FAQ_What_is_a_wizard%3F
(7) https://www.html.it/pag/16729/tomcat-e-apache/
(8) http://tomcat.apache.org/
(9) http://www.354353.com/computer/Software/1492227.html
(10) http://www.di.uniba.it/~ig/animazione/thread.htm#:~:text=Un%20programma%20multithread%20contiene%20due,un%20percorso%20separato%20di%20esecuzione
(11) https://www.datariver.it/chi-siamo/