# Progettazione e pattern architetturali usati

Il sistema è stato progettato utilizzando le tecniche di modellazione ad oggetti. Dal punto di vista architetturale, ci siamo ispirati al **pattern MVC**. Durante la fase di sviluppo del codice abbiamo optato poi ad una variante di quest’ultimo, aggregando model e controllore in un’unica classe. La struttura dei dati è definita e gestita dalla tecnologia MySQL e la manipolazione dei dati viene gestita all’interno del controllore tramite le query. Ciò ha permesso di separare nettamente e comodamente la componente grafica dalla componente strutturale di gestione dei dati grazie anche all’implementazione nativa di Java FX e il suo relativo formato FXML.

• **Modello-Controllore**: parte del sistema che riguarda i dati e le informazioni memorizzate. I dati vengono mossi all’interno del sistema grazie alle query sql. Questa parte del sistema definisce anche la logica applicativa, ovvero il comportamento del sistema a fronte degli stimoli esterni governati dai listener implementati nativamente da Java FX.

• **Vista**: parte del sistema che rappresenta visivamente il modello-controllore, e quindi, i dati del sistema. Si è utilizzata la tecnologia di Java FX, basata sulla piattaforma Java.

Le azioni dell’utente saranno catturate dai listener, implementati nativamente da Java FX e controllati da relative funzioni ad hoc. Ogni listener, e quindi funzione, è istruito a reagire in determinati modi in base alle esigenze e alle richieste viste a inizio progetto, andando a modificare le informazioni contenute nella base di dati e quindi ad aggiornare la vista, che rappresenta il modello.

Seguono un semplice schema dell’architettura del sistema ed i diagrammi UML delle classi del Modello e della Vista-Controllore. Di seguito una rappresentazione grafica del pattern MVC, nel nostro caso Controller e Model sono uniti in una stessa classe che si interfaccia alla struttura dei dati tramite le query Sql.

Immagine che contiene testo, schermata, diagramma, design

Descrizione generata automaticamente

# Implementazione e design pattern usati

Immagine che contiene testo, diagramma, Carattere, schermata

Descrizione generata automaticamentePer l’implementazione dell’applicazione ci siamo serviti del Design Pattern Singleton. Questo pattern è stato utilizzato all’interno della classe “UserSession”, per gestire le sessioni dell’utente. La classe mantiene all’interno il riferimento all’unica istanza Singleton della classe. Questa classe contiene poi tutti i metodi, le proprietà e gli attributi tipici dell’astrazione per cui è stata concepita. Le classi Singleton implementano un metodo statico per assicurare che nessuna altra istanza venga creata oltre la prima, restituendo contemporaneamente un riferimento all’unica esistente. Di seguito una rappresentazione del funzionamento della classe Singleton, si è deciso per comodità di rappresentare come “Client” tutte le classi con le relative funzioni che generano un’istanza Singleton.

# Diagrammi di sequenza

Seguono diagrammi di sequenza che mostrano le dinamiche di alcune interazioni tra classi particolarmente complesse od interessanti. Sono stati rappresentati soltanto gli scambi salienti.

Immagine che contiene diagramma, testo, Parallelo, Disegno tecnico

Descrizione generata automaticamente

loginCitizen()

enter codice fiscale/password()

session.getInstance()

:session

:login

:database

cittadino

Cciicccc

Alt

:Reason Listener

:Disponibility Listener

[OK]

Choose reason from checkbox

:Request Controller

**[last request.status != “opened”, “closed”]**

Error message()

cittadino

update previous request (day appoint., time, status=”processing”)

: onClickConfirmButton ()

Select office, day, time

Select office, day, time

**[last request.status == “opened”]**

:database

update previous request (day appoint., time, status=”processing”)

: onClickConfirmButton ()

: onClickConfirmButton ()

Add new request (codice fiscale, date now, reason, status=”opened”)

**[last request.status == “closed”]**

:Request Listener

: onClickRequestButton ()

Read requests ()

Retrieve requests ()

Click on request button ()

Personale

:Disponibility Controller

:Disponibility Listener

Choose office from checkbox

loop

Add new disponibility (office, day, time, typology)

**[for each slot in slots]**

: onClickDisponibilityButton ()

Choose #slots from checkbox

Choose time from checkbox

Choose typology from checkbox

Choose day from checkbox

: onClickDisponibilityButton ()

Click on disponibility button ()

:Disponibility Listener

:database

Read disponibility ()

Retrieve disponibility ()