***UNIBite – Architettura e design***

Università degli studi di Bergamo  
Ingegneria Informatica

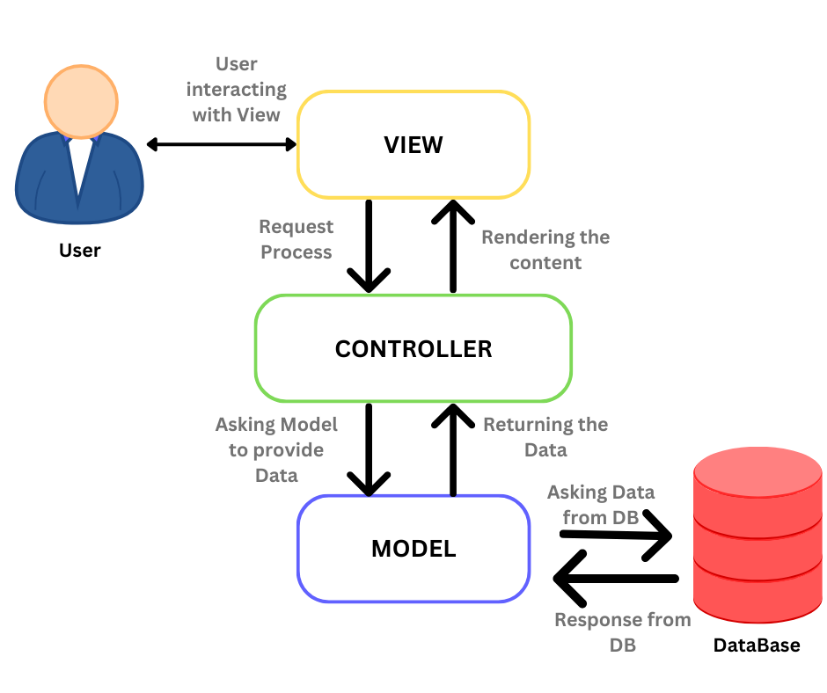
Beccarelli Raissa Matr. 1086785  
Locatelli Giacomo Matr. 1086262  
Valceschini Marco Matr. 1086356

***Immagine che contiene testo, Carattere, Elementi grafici, simbolo

Descrizione generata automaticamente***

**1 Software architecture**

L’architettura del progetto è basata su uno stile di tipo MVC (model-view-controller), secondo la quale il progetto si suddivide nei seguenti pacchetti:

* Model, in cui è organizzata la funzione logica del progetto per gestire i dati tramite la connessione al database;
* Controller, in cui si gestiscono le classi che servono da intermediarie tra l’utente e il database, gestendo gli input dell’utente e trasferendoli al model;
* View, nel nostro progetto chiamato GUI, in cui è implementata la parte che si interfaccia con l’utente mostrandogli l’output tramite delle viste sviluppate con Vaadin.

Tramite lo stile MVC l’utente si interfaccia con la view, la quale comunica con il controller che passa l’input dell’utente al model. Il model con le informazioni ricevute è in grado di aggiornare il database effettuando modifiche e di ottenere informazioni che poi saranno passate nuovamente prima al controller e poi come output all’utente.  
Utilizziamo questo stile tramite componenti e connettori di tipo:

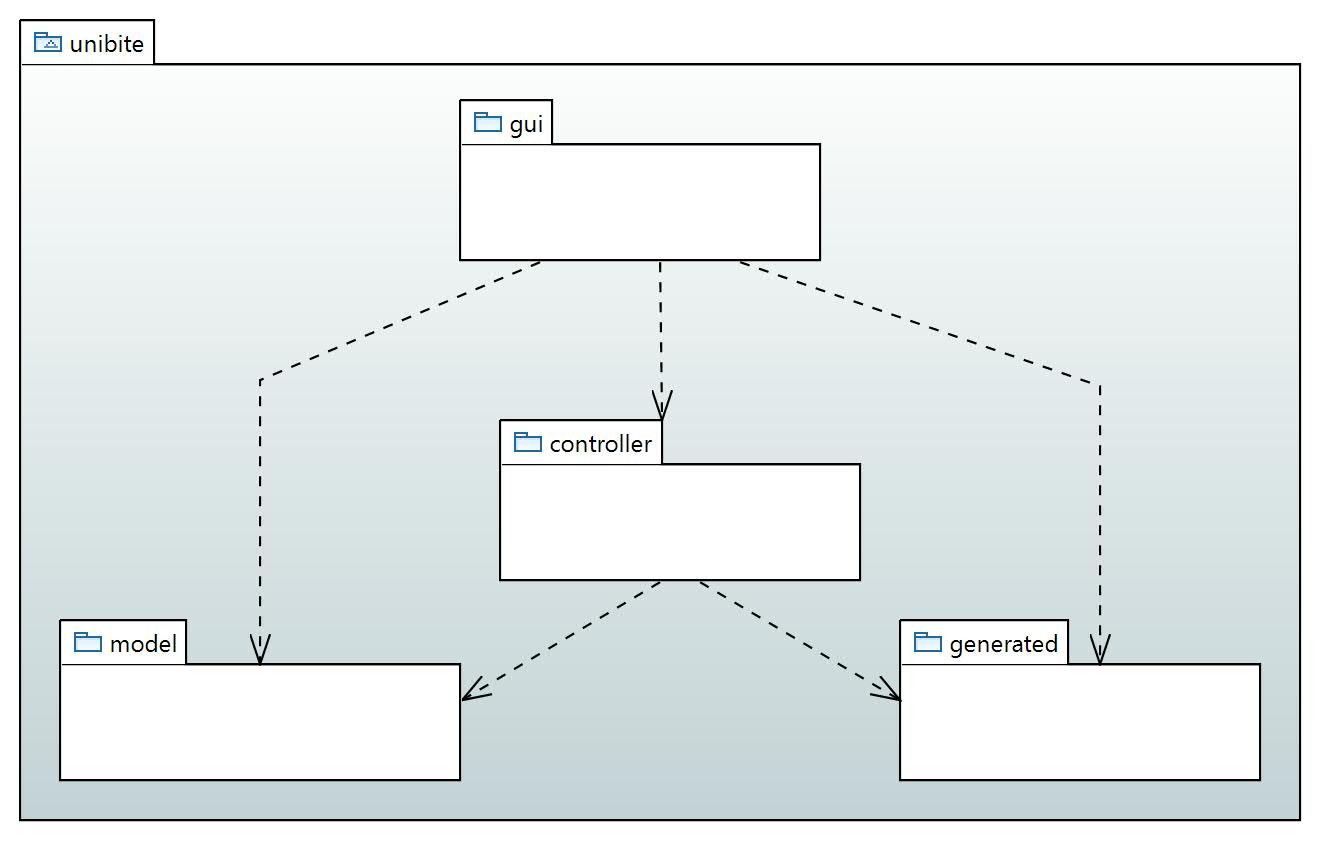
* Componenti: raccolte di procedure;
* Connettori: chiamate di procedure, che utilizzano un unico thread di controllo tra chiamante e chiamato.

Per descrivere l’architettura software utilizziamo i punti di vista:

* Punto di vista del modulo, che ci permette di avere una visione statica del programma. I moduli che abbiamo sono 4:
* Modulo per l’autenticazione, che gestisce il login e la verifica delle credenziali;
* Modulo carrello, che contiene la logica per la gestione degli ordini;
* Modulo database, che gestisce dati degli utenti, dei pasti e delle prenotazioni;
* Modulo interfaccia grafica (Vaadin), che interagisce con l’utente.

Il punto di vista del modulo che abbiamo usato è stato quello dell’Uso, per far sì che, se un elemento viene modificato allora anche tutti gli elementi che lo utilizzano reagiscano di conseguenza modificandosi a loro volta.

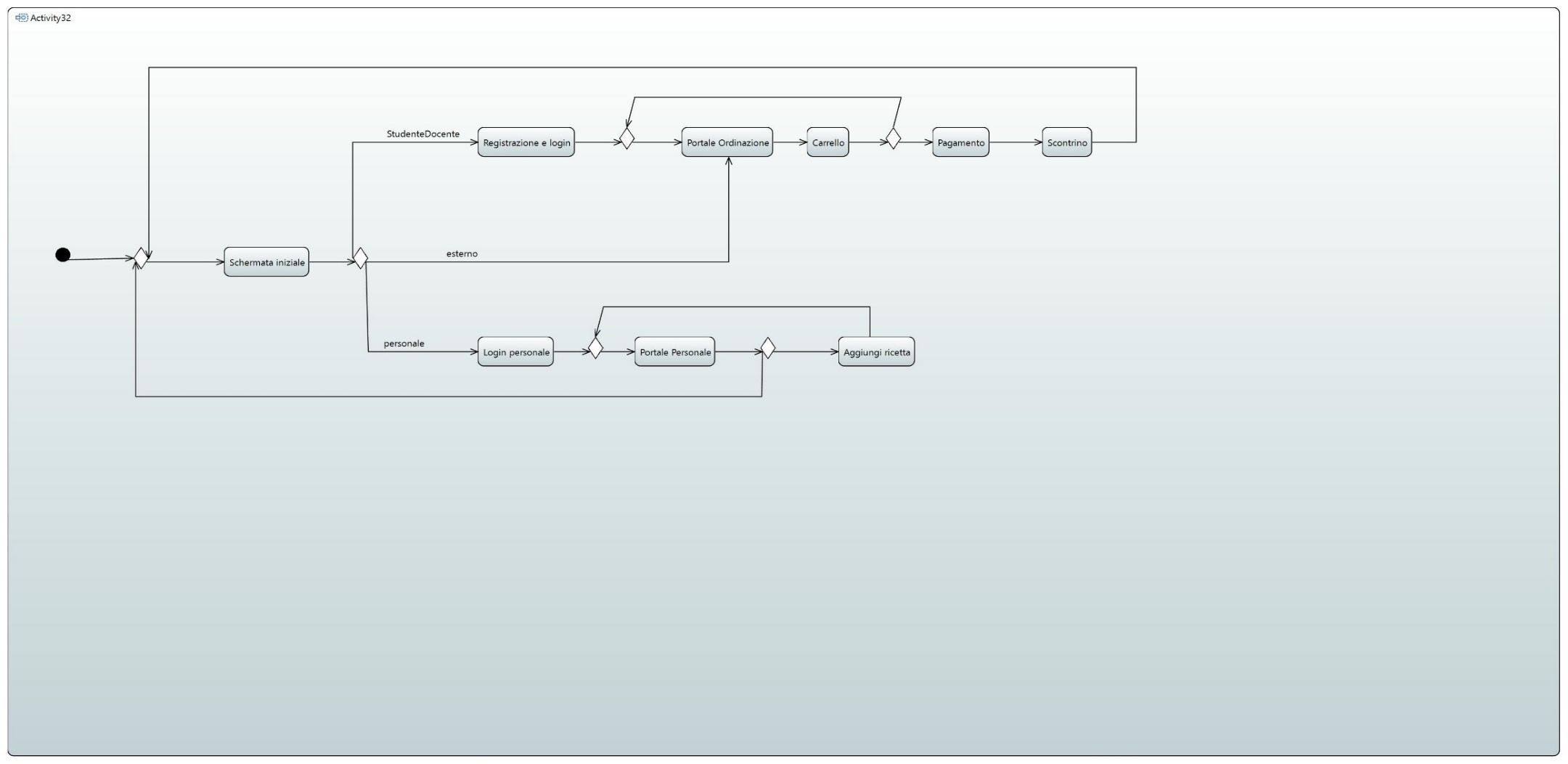
* Punto di vista dei componenti e dei connettori, per poter descrivere come si comporta il sistema mentre viene eseguito.
* Processi, in quanto il programma è formato da una serie di processi che comunicano tra loro;
* Dati condivisi, ovvero per descrivere come vengono utilizzati ed immagazzinati i dati persistenti.
* Punto di vista dell’allocazione, che descrive la relazione tra il programma e l’ambiente in cui è sviluppato.
* Incarico di lavoro, permette di capire chi sta facendo il lavoro e di cosa si sta occupando. Nel nostro programma i tre membri lavorano sia singolarmente che in gruppo, assegnandosi incarichi a seconda delle conoscenze.
* Distribuzione, ovvero in che modo il programma si interfaccia e comunica con un database.

**Diagramma dei package:**

**Diagramma dei componenti:**

Inoltre, all’interno del nostro progetto abbiamo utilizzato diverse librerie qui sottoelencate:

* Vaadin: Framework per la creazione di interfacce grafiche web in java
* SQLite JDBC Driver: Driver JDBC per interfacciarsi con database SQLite
* JOOQ Core: Libreria per il mapping SQL a codice java, lo abbiamo utilizzato per semplificare le query SQL
* JOOQ Codegen: Strumento che abbiamo utilizzato per generare automaticamente le classi java corrispondenti alle tabelle del database
* Apache Log4j: Framework per la gestione dei log, lo abbiamo utilizzato per le fasi di debugging, per registrare informazioni utili al funzionamento dell’applicazione.

**2 Software Design**

**2.1 Principi di progettazione**

**2.1.1 Analisi statica del codice**

L’analisi statica consiste nel verificare il codice senza alcuna esecuzione, con lo scopo di individuare bug, vulnerabilità problemi di conformità agli standard di codifica, complessità e altri difetti che potrebbero portare a malfunzionamenti o scarsa manutenibilità. Nel nostro progetto abbiamo deciso di utilizzare SonarLint e PMD, due tool che analizzano il codice ed evidenziano i potenziali problemi. Essi si basano su un set di regole che coprono diverse aree:

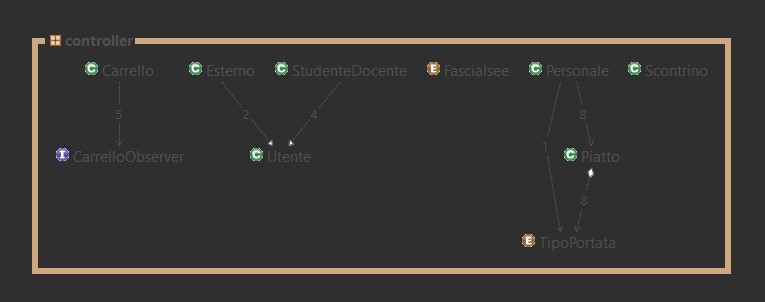
* Bug: Errori logici o difetti che possono causare comportamenti indesiderati.
* Code Smells: Problemi di design o scelte di implementazione che rendono il codice meno leggibile o manutenibile.
* Vulnerabilità di sicurezza: Potenziali rischi di attacchi o falle nella sicurezza.
* Debt (debito tecnico): Aree del codice che necessitano di miglioramenti per ridurre la complessità.

Nel nostro caso l’utilizzo di questi tool ci è stato utile, per esempio, per identificare SQL vulnerabili, segnalare codice duplicato o metodi troppo complessi e ci hanno fornito suggerimenti per migliorare la manutenibilità del codice.

**2.1.2 Analisi strutturale del codice**

L’analisi strutturale del codice mira a esaminare l’organizzazione interna del software. Essa aiuta a rilevare accoppiamenti stretti, cicli di dipendenza, loop inutili, funzioni non utilizzate, metodi ridondanti e verifica che il codice rispetti l’architettura prevista. StanIde è uno strumento specifico per l’analisi strutturale del codice, esso ci ha permesso di:

* Analizzare il codice sorgente: Identifica le relazioni tra classi, moduli e librerie.
* Costruire diagrammi strutturali: Visualizza graficamente la struttura del codice, evidenziando dipendenze o violazioni.
* Identificare problemi strutturali: Come cicli di dipendenza, violazioni dell'architettura o codice spaghetti.

Questo è il grafico che rappresenta le dipendenze tra le classi, interfacce ed enumerativi all’interno del package controller:

Alcune metriche che abbiamo osservato dal nostro codice sono per esempio:

* Nella classe Carrello abbiamo:
* Coupling afferente: 3
* Coupling efferente: 4
* Instabilità: 0.57
* Numero metodi: 12
* CBO: 4
* Nella classe StudenteDocente abbiamo:
* Coupling afferente: 4
* Coupling efferente: 3
* Instabilità: 0.43
* Numero metodi: 12
* CBO: 2
* Nel pacchetto Controller abbiamo:
* Coupling afferente: 8
* Coupling efferente: 4
* Instabilità: 0.33

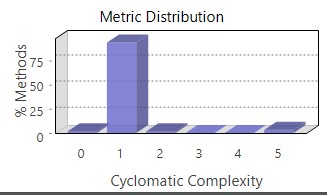
**2.1.3 Complessità ciclomatica di McCabe**

La complessità ciclomatica è una metrica che misura la complessità del codice sorgente. Essa rappresenta il numero di percorsi indipendenti attraverso un programma e fornisce un'indicazione del livello di complessità e manutenibilità di un modulo software.

Andiamo ad analizzare e calcolare la complessità ciclomatica di questa funzione che si trova all’interno della classe Carrello tramite l’utilizzo di StanIde:



Complessità ciclomatica = 5

Per quanto riguarda il pacchetto controller la complessità ciclomatica è pari a 1.13, il grafico sottostante rappresenta la distribuzione di complessità sul numero di metodi:

**2.2 Design Patterns**

Nel progetto sono stati utilizzati due pattern: il singleton pattern e il observer-observable pattern.

Il primo pattern è stato implementato per la gestione della connessione al database. Questo ha permesso di creare una singola istanza per gestire le chiamate al database, che ci sono servite sia per inserire dati nei record sia per riutilizzare informazioni nel programma, per fare calcoli o per mostrarli a schermo. Questo è stato creato tramite l’implementazione di una classe Connessione (all’interno del package model) nella quale abbiamo inserito un’istanza privata, un costruttore privato che garantisca l’esistenza di una sola istanza e un metodo getInstance() per poter richiamare tra le varie classi la connessione al database.

Il secondo pattern è stato utilizzato per la gestione delle notifiche all’interno del portale di ordinazione. Ogni volta che viene aggiunto un piatto al carrello tramite il pulsante “aggiungi piatto”, viene visualizzata a schermo una notifica. Questo meccanismo di comunicazione si basa sull'utilizzo di un'interfaccia denominata CarrelloObserver, che agisce come intermediario tra il portale di ordinazione e la classe Carrello. Quando l'utente preme il pulsante sul portale di ordinazione, l'observer, implementato dal portale stesso, comunica con il Carrello tramite l'interfaccia. A sua volta, il Carrello notifica il sistema, che provvede a mostrare una notifica a schermo.