



Backend per gestionale di fatture realizzato con Java EE 7

Alberto Merciai, Simone Ricci

Laurea Magistrale in Ingegneria Informatica
Software Architecture and Methodologies

Introduzione

- **Idea chiave:**
 - il progetto nasce dall'idea di avere un sistema che consenta ai proprietari di un'azienda di poter gestire le proprie fatture.

Obiettivo e Fasi sviluppo

- **Obiettivo:**
 - Realizzazione della logica per un sistema gestionale di fatture
- **Fasi di sviluppo:**
 - Analisi dei requisiti
 - Progettazione e Sviluppo Software
 - Test

Analisi dei Requisiti

- **fasi:**
 1. Descrizione informale e stesura dei Requisiti
 2. Modello Concettuale
 3. Casi d'uso
 4. Template
 5. Page Navigation Diagram
 6. Mockup

Descrizione Informale e stesura dei Requisiti

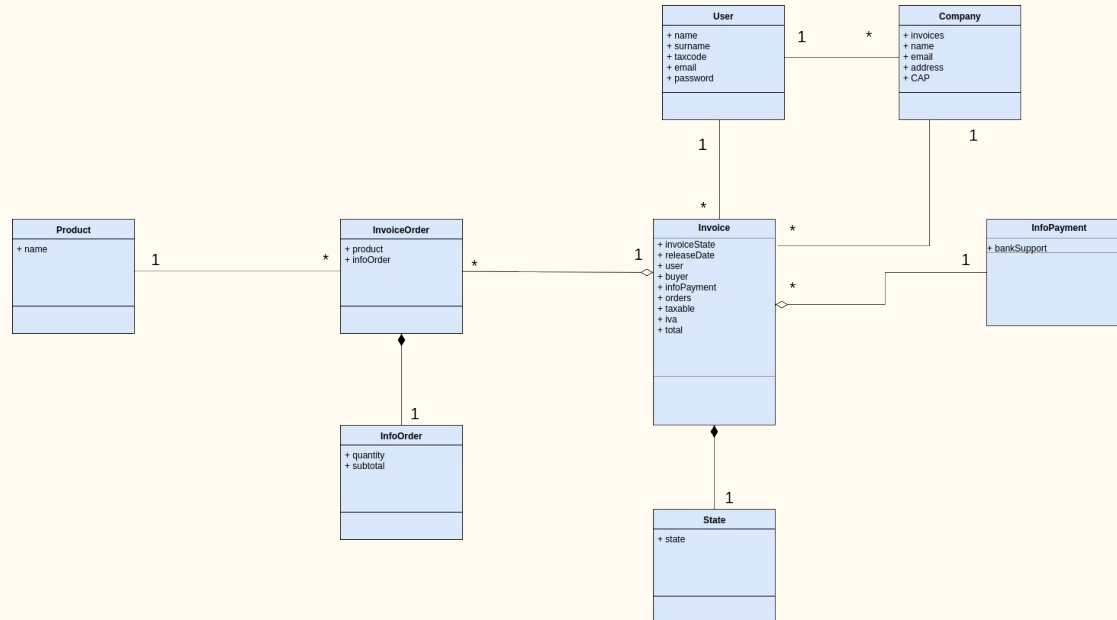
- durante questa fase a partire da una descrizione informale sono estratti:
 - **Data Requirements:** requisiti che il sistema deve possedere per soddisfare le necessità legate alla memorizzazione e la manipolazione dei dati
 - **Functional Requirements:** requisiti legati alle funzionalità che il programma deve offrire per soddisfare le esigenze dell'utente

- Un Emittente è composto da:
 - dati relativi all'Emittente
 - nome
 - sede (CAP via...)
 - partita iva
 - breve descrizione
 - nome e cognome
 - codice fiscale
 - e-mail

- Il sistema deve poter permettere all'Emittente di autenticarsi
 - se già registrato immettere user e psw
 - se non registrato riempie i campi legati a se stesso

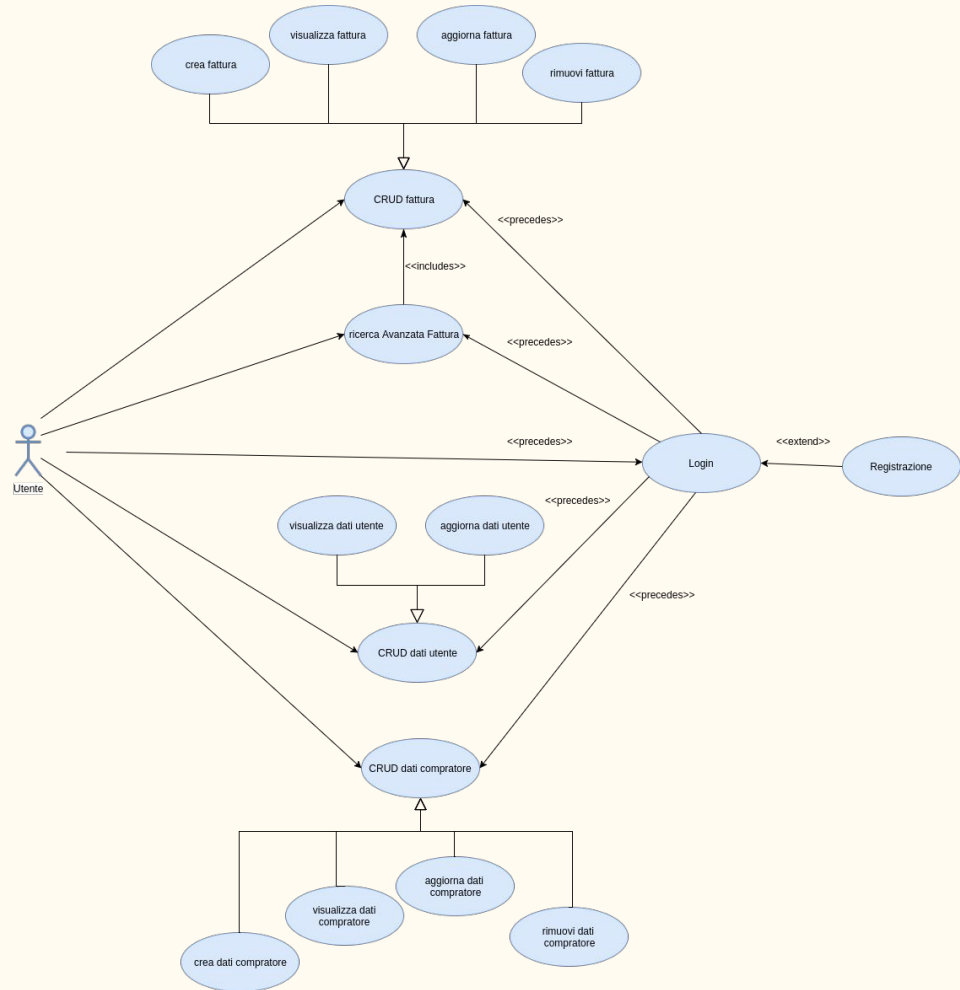
Modello Concettuale

- dai requisiti sono estratte quelle che sono le Entità coinvolte nel sistema:
 - **Modello Concettuale:** costruito tramite class diagram ponendo enfasi sulle entità coinvolte piuttosto che i dettagli di design del software



Casi d'uso

- i casi d'uso sono guidati dai requisiti funzionali, successivamente riportati in un diagramma UML.



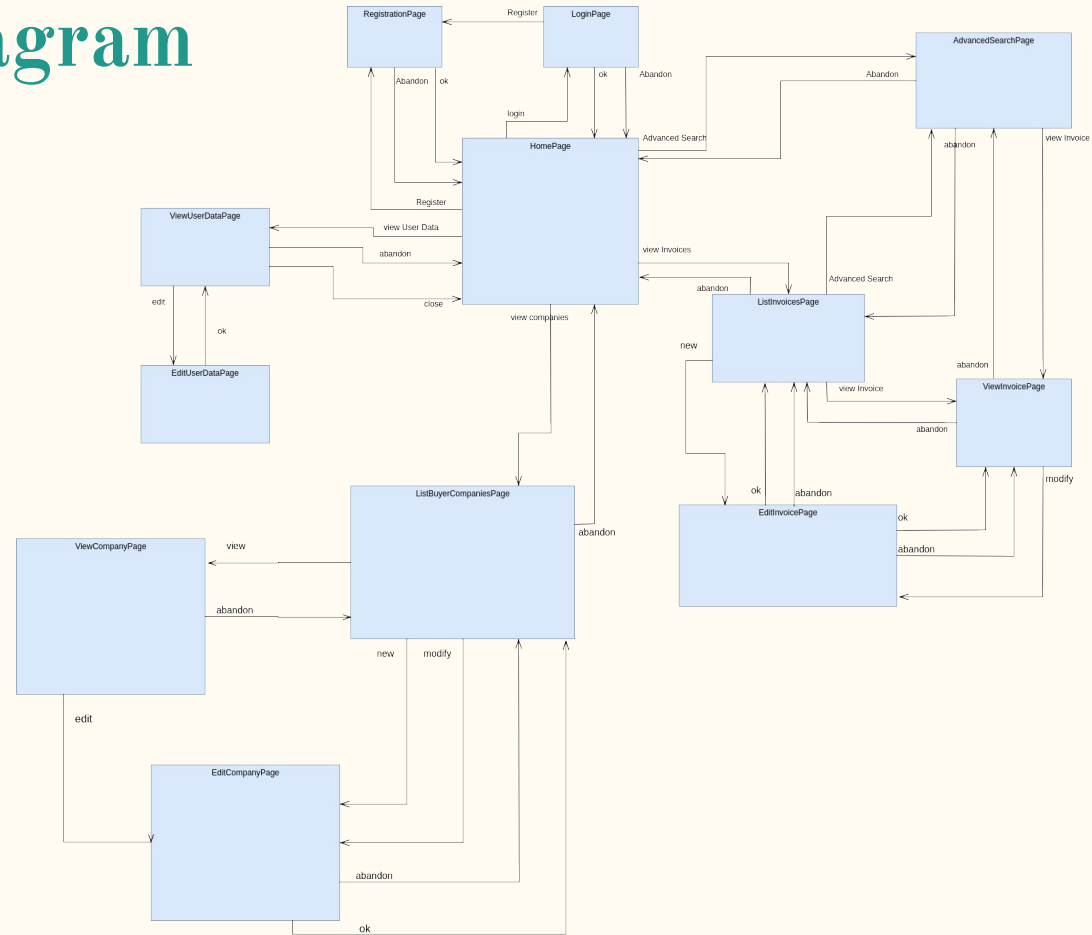
Template

- Ogni caso d'uso estratto è documentato attraverso un template:

Use Case: Login
Name: Login
Description: L'utente inserisce le credenziali per accedere al servizio
Primary Actors: Utente
Level: User Goal
Preconditions: L'utente deve essere già registrato nel sistema, altrimenti procede col caso d'uso Registrazione
Main Flow: 1. L'utente seleziona il bottone LOGIN dalla pagina HomePage ed è rediretto alla pagina LoginPage 2. L'utente inserisce i campi per effettuare il login 3. l'utente seleziona il bottone LOGIN
Postconditions:
Alternative Flows: 1. se l'utente non è in possesso delle credenziali occorre effettuare la registrazione premendo dalla pagina LoginPage il bottone REGISTER

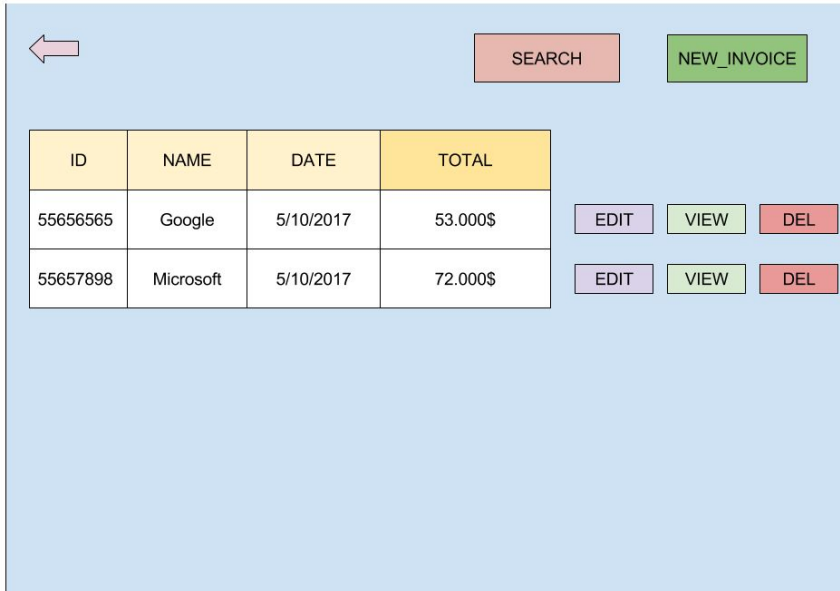
Page Navigation diagram

- Dagli use case è ricavato il **page navigation diagram** che permette di visualizzare quali sono le pagine presenti nell'applicativo e come queste interagiscono tra loro.



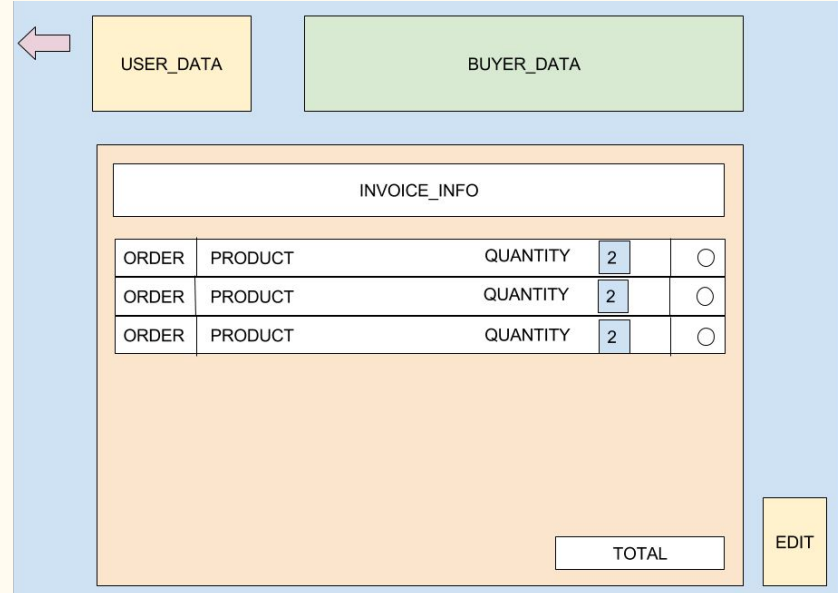
Mockups

- consentono di avere un'idea meno astratta dell'interfaccia del nostro sistema.
- facilitano la realizzazione dei **controller** nella fase implementativa.



Mockup of an invoice list interface. It features a back arrow, a 'SEARCH' button, and a 'NEW_INVOICE' button. Below these is a table with columns: ID, NAME, DATE, and TOTAL. To the right of the table are 'EDIT', 'VIEW', and 'DEL' buttons for each row.

ID	NAME	DATE	TOTAL
55656565	Google	5/10/2017	53.000\$
55657898	Microsoft	5/10/2017	72.000\$



Mockup of an invoice details interface. It features a back arrow, 'USER_DATA' and 'BUYER_DATA' sections, and an 'INVOICE_INFO' section. The 'INVOICE_INFO' section contains a table with columns: ORDER, PRODUCT, QUANTITY, and a radio button. Below the table is a 'TOTAL' label and an 'EDIT' button.

INVOICE_INFO			
ORDER	PRODUCT	QUANTITY	2 <input type="radio"/>
ORDER	PRODUCT	QUANTITY	2 <input type="radio"/>
ORDER	PRODUCT	QUANTITY	2 <input type="radio"/>

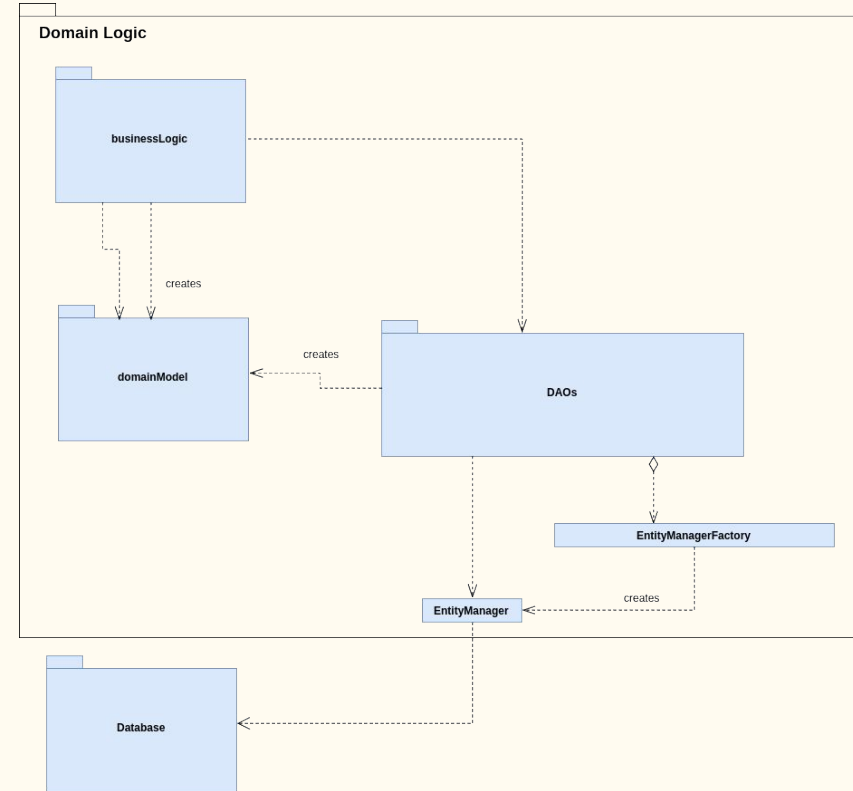
TOTAL

Progettazione e sviluppo

- **fasi:**
 1. Architettura
 2. Modello di dominio
 3. DAO
 4. Business logic

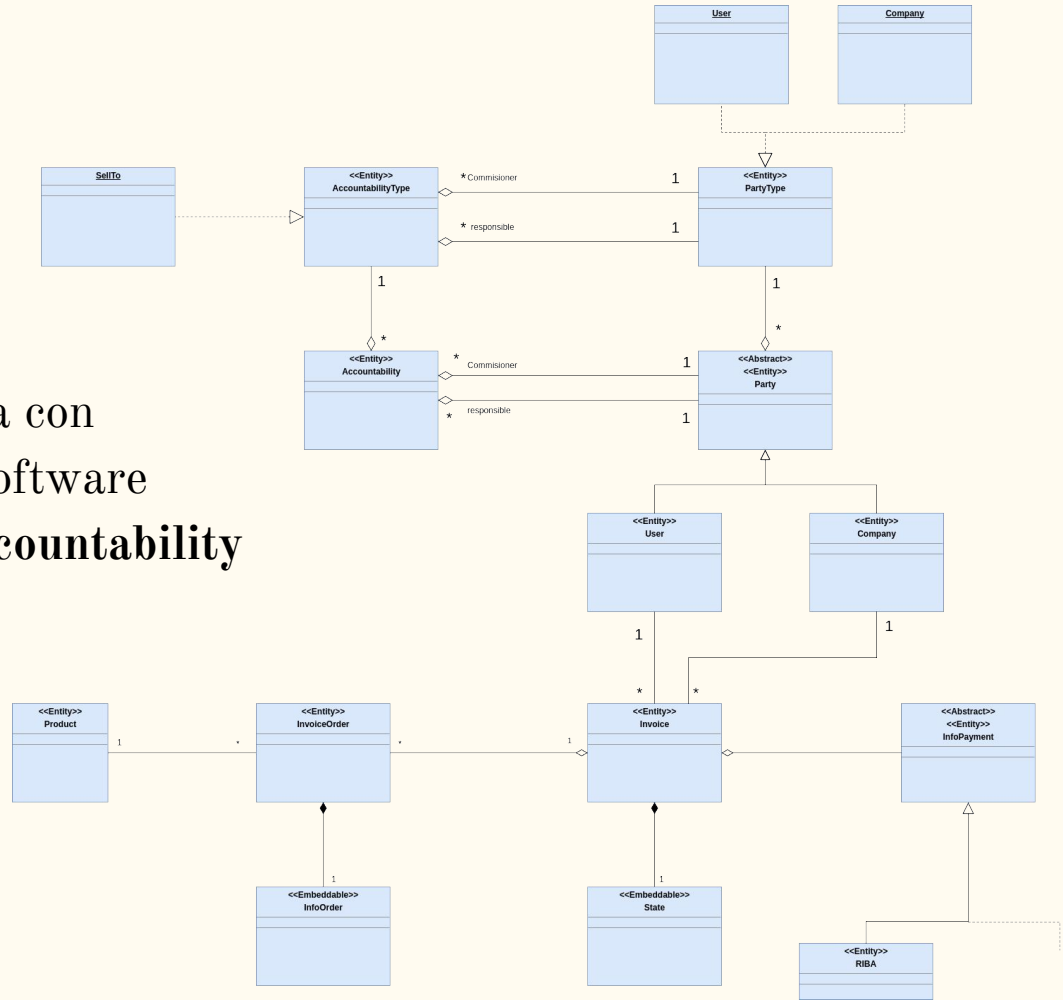
Architettura

- Tipologia: **3-tier**
 - **Presentation Layer** (non implementata)
 - **Domain Logic**
 - Domain model
 - Data Access Object
 - Business logic
 - **Database**



Modello di Dominio

- simile al modello concettuale, ma con maggiore enfasi sul design del software
- comprende pattern di analisi **Accountability**
- inserita **BaseEntity**
- **Java + annotazioni JPA**



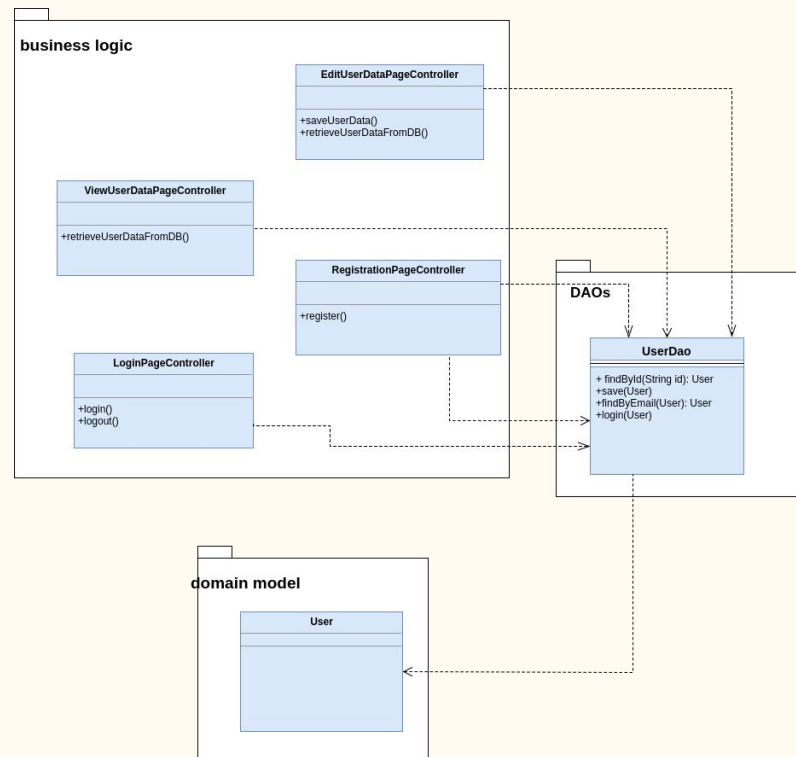
Data Access Object (DAO)

- **DAO:**
 - aumenta il livello di astrazione tra domain model e business logic
 - disaccoppia la business logic dalle operazioni di persistenza
- **Base DAO:**
 - immagazzina le operazioni comuni a tutti i dao
 - permette di non duplicare codice
- **Java + operazioni JPA + annotazioni CDI**

```
public class BaseDao<E> {  
  
    private final Class<E> typeClass;  
  
    @PersistenceContext  
    protected EntityManager entityManager;  
  
    public BaseDao(Class<E> typeClass) {  
        this.typeClass = typeClass;  
    }  
  
    public void save(E entity) {  
        entityManager.persist(entity);  
    }  
  
    public void remove(E entity) {  
        entityManager.remove(entity);  
    }  
  
    public E findById(Long typeId) {  
        return entityManager.find(typeClass, typeId);  
    }  
}
```

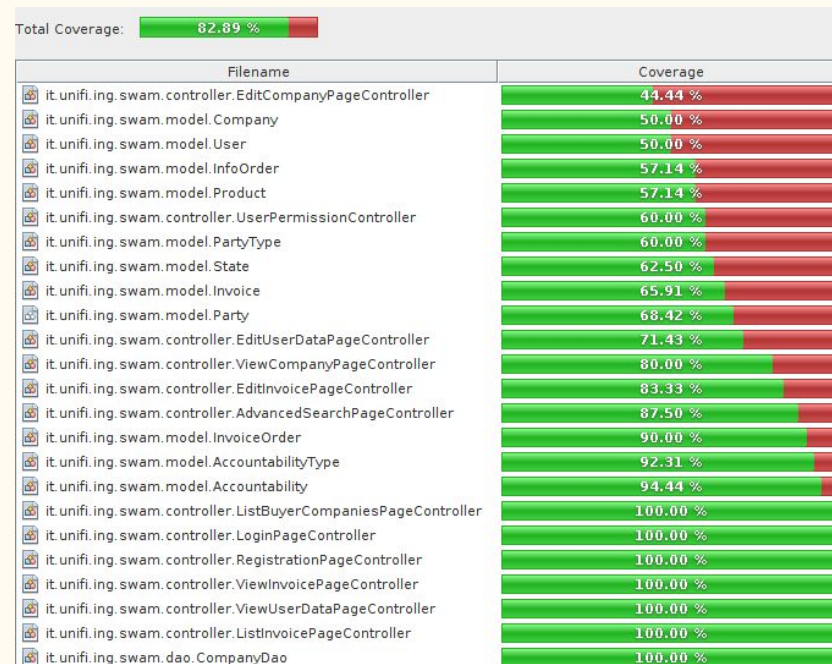
Business Logic

- guidata dai casi d'uso
- reagisce agli ingressi implementando le corrispondenti azioni
- comprende i tutti **controller** che reagiscono alle pagine
- **Java + annotazioni CDI**



Test

- fase eseguita iterativamente durante tutta l'implementazione del software
- i test eseguiti sono di tipo **Unit Test**
 - in questi sono testate le classi Java
- le tecnologie utilizzate sono i framework **JUnit** e **Mockito**.
- nella fase finale di testing è stato eseguito il programma **Jacoco** per valutare la coverage dei test.
 - copertura **82.89%**



Conclusioni e futuri sviluppi

- Il backend è sviluppato seguendo le tecniche viste a lezione e testato garantendo la copertura indicata precedentemente questo garantisce una certa qualità
- come futuri sviluppi sarebbe interessante costruire un'interfaccia da integrare al software descritto ed effettuare così usability test del sistema.