O.D.D.

Object Design Document

Team:

NOME	MATRICOLA
Raffaele Casaburi	0512103796
Giuseppe Ranieri	0512103616
Antonio Piacentini	0512103552

1. Introduzione

Definiti i precedenti documenti RAD e SSD, questo documento ha lo scopo di produrre un modello capace di integrare in modo coerente e preciso tutte le diverse funzionalità individuate nelle fasi precedenti.

In particolare si descriveranno i trade-offs generali realizzati dagli sviluppatori, le linee guida sulla documentazione delle interfacce e le convenzioni di codifica, le interfacce delle classi, le operazioni, gli argomenti e la signature dei sottosistemi definiti nel System Design.

1.1 Object Design Trade-offs

Comprensibilità vs costi

Al fine di rendere il codice più comprensibile sia per i membri del progetto che non sono stati coinvolti in una determinata porzione del progetto, sia per professionisti esterni che non hanno collaborato alla progettazione del sistema, verranno aggiunti commenti chiari e mirati così da migliorare anche una possibile manutenzione. Si preferisce dunque, aggiungere costi relativi alle ore/uomo dedicate per la documentazione.

Prestazioni vs Costi

Per la progettazione del sistema si ha a disposizione un budget non proprio eccessivo, si utilizzano dunque componenti free e/o open source, il necessario affinché possa essere garantito, per quanto possibile, un normale utilizzo.

Costi vs Mantenimento

L'utilizzo di componenti open source e di linguaggi dinamici "HTML, XML, CSS, JAVASCRIPT, etc..", permette di aggiungere facilmente nuove funzioni, aggiornamenti e manutenzione.

Interfaccia vs Easy-use

L'interfaccia è pensata affinché le funzionalità del sistema, siano semplici ed intuitive anche agli occhi di utenti non esperti di sistemi informatici (Easy-use).

Memoria vs efficienza

Mediante l'utilizzo di Cookie viene resa possibile la memorizzazione dei dati con accesso più frequente. Il tutto per evitare un rallentamento delle prestazione del sistema, dovuto ad un crescente numero di visitatori ognuno dei quali libero di effettuare una o più query al sistema.

Sicurezza vs costi

La indisponibilità di un budget alto non permette agli sviluppatori di utilizzare componenti in grado di garantire la massima sicurezza. Tuttavia, per quanto possibile, verrà garantito un grado di sicurezza soddisfacente utilizzando le regole dei vari linguaggi utilizzati.

Interfacce vs tempo di risposta

Il tempo di risposta tra server e interfaccia sono più che sufficienti a soddisfare le richieste da parte dell'utente.

1.2 Linee Guida per la Documentazione delle Interfacce

Gli sviluppatori seguiranno alcune linee guida per la scrittura del codice:
Naming convention

- E' buona norma utilizzare nomi:

1. Descrittivi
2. Pronunciabili
3. Di uso comune

4. Di lunghezza medio-corta
5. Non abbreviati
6. Evitando la notazione ungherese
7. Utilizzando solo caratteri consentiti (a-z, A-Z, 0-9)

Variabili:
● I nomi delle variabili devono cominciare con una lettera minuscola, e le parole seguenti con la lettera maiuscola. Quest'ultime devono essere dichiarate ad inizio blocco, solamente una per riga e devono essere tutte allineate e facilitarne la leggibilità. Esse possono essere annotate con dei commenti.

● E' inoltre possibile, in alcuni casi, utilizzare il carattere underscore ("_") per la definizione del nome.

Metodi:
● I nomi dei metodi devono cominciare con una lettera minuscola, e le parole seguenti con la lettera maiuscola. Il nome del metodo tipicamente consiste in un verbo che identifica una azione, seguito dal nome di un oggetto.

I nomi dei metodi per l'accesso e la modifica delle variabili dovranno essere del tipo getNomeVariabile() e SetNomeVariabile().
● I commenti dei metodi devono essere raggruppati in base alla loro funzionalità, la descrizione dei metodi deve apparire prima di ogni dichiarazione di metodo, e deve descriverne lo scopo. Deve includere anche informazioni sugli argomenti, sul valore di ritorno, e se applicabile, sulle eccezioni.

Classi e pagine:
● I nomi delle classi e delle pagine devono cominciare con una lettera maiuscola, e anche le parole seguenti all'interno del nome devono cominciare con una lettera maiuscola. I nomi di quest'ultime devono fornire informazioni sul loro scopo.

● La dichiarazione di classe deve essere caratterizzata da:

1. Dichiarazione della classe pubblica
2. Dichiarazioni di costanti
3. Dichiarazioni di variabili di classe
4. Dichiarazione di variabili d'istanza
5. Costruttore
6. Commento e dichiarazione dei metodi

1.3 Definizioni, acronimi e abbreviazioni

- **RAD :** Requirements Analysis Document
- **SDD** : System Design Document
- **ODD :** Object Design Document

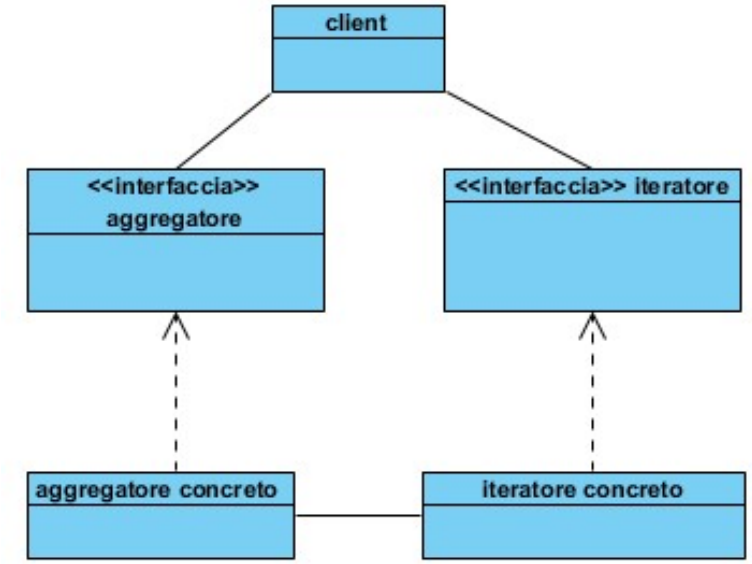
1.4 Riferimenti

- **B.Bruegge, A. H. Dutoit, Object Oriented Software Engineering - Using UML, Pattern and Java, Prentice Hail, 3rd edition, 2009.**
- Documento RAD del progetto El Tamborito.
- Documento SDD del progetto El Tamborito.

2. Design pattern

Iterator Pattern

L'iterator risolve diversi problemi relativi all'accesso e alla navigazione attraverso gli elementi, come una struttura di dati contenitrice senza esporre i dettagli dell'implementazione e della struttura interna del contenitore. L'oggetto principale su cui si base questo design pattern è l'iteratore. Una classe contenitrice dovrebbe consentire l'accesso e la navigazione attraverso l'insieme degli elementi che contiene. Nel nostro caso viene utilizzato per accedere al database:



-L'interfaccia iteratore espone i metodi di accesso alla struttura dati.

-L'aggregatore definisce l'interfaccia per creare un oggetto di tipo iteratore.

-L'aggregatore concreto implementa l'interfaccia di creazione di un oggetto.

3. Packages

Il nostro sistema presenta una suddivisione basta su tre livelli (three-tier):

- Presentation Layer
- Application Layer
- Storage Layer

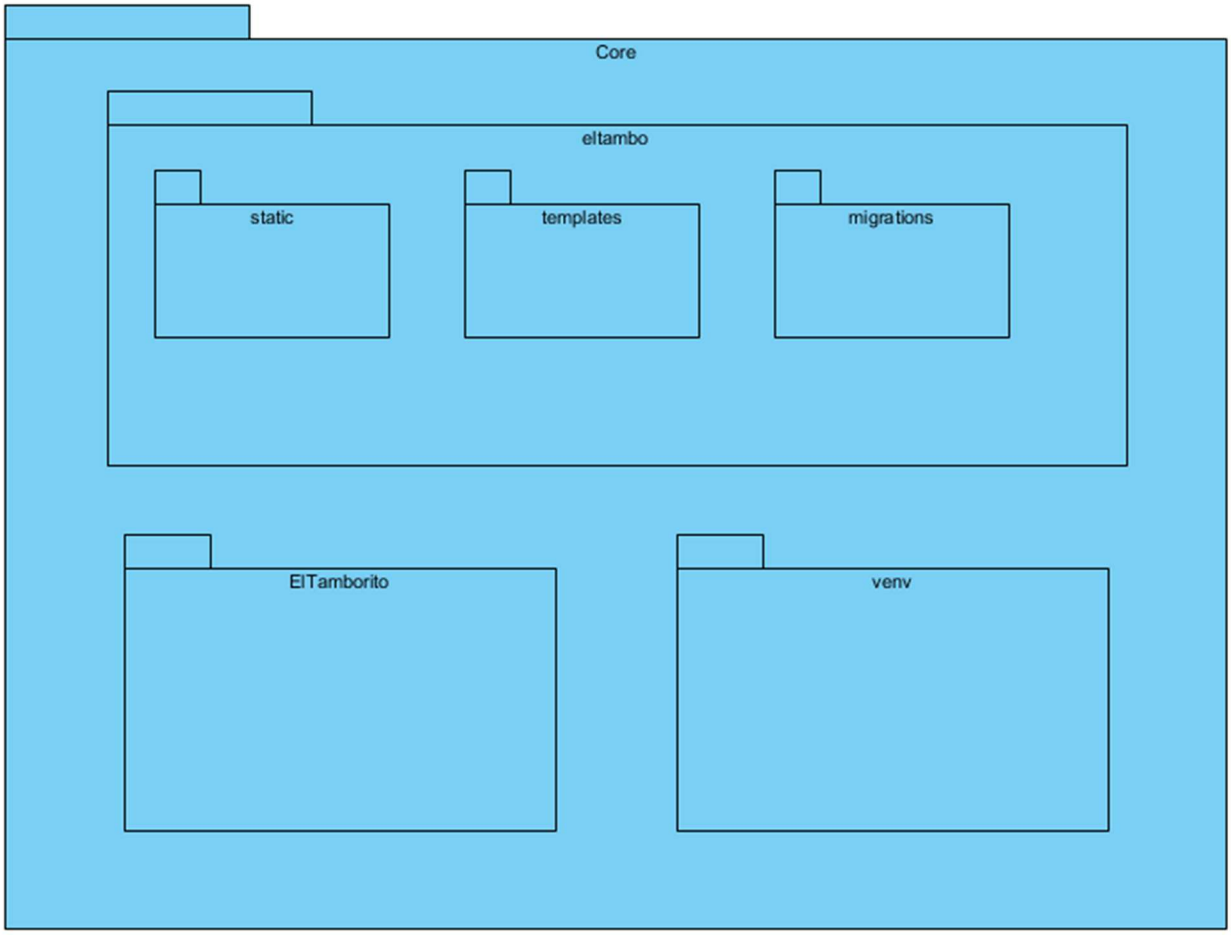
Il package El Tamborito contiene sottopackage che a loro volta inglobano classi atte alla gestione delle richieste utente. Le classi contenute nel package svolgono il ruolo di gestore logico del sistema.

Presentation Layer	Rappresenta l'interfaccia del sistema, ed offre la possibilità all'utente di interagire con il esso offrendo sia la possibilità di inviare in input, che di visualizzare in output dati.

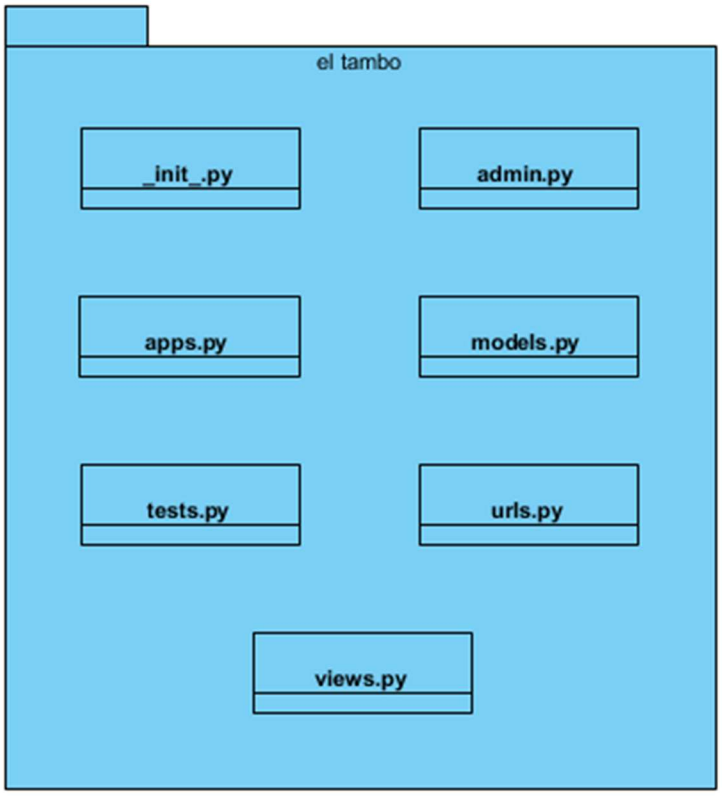
Application Layer	Ha il compito di elaborare i dati da inviare al client, e spesso grazie a delle richieste fatte al database, tramite lo Storage Layer, accede ai dati persistenti. Si occupa di varie gestioni quali: - Gestione Profilo - Gestione Acquisti - Gestione Profili - Gestione Carrello - Gestione Prodotti

Storage Layer	Ha il compito di memorizzare i dati sensibili del sistema e di ricevere le varie richieste dall'Application Layer e restituendo i dati richiesti.

3.1 Package core



3.2 Package eltambo

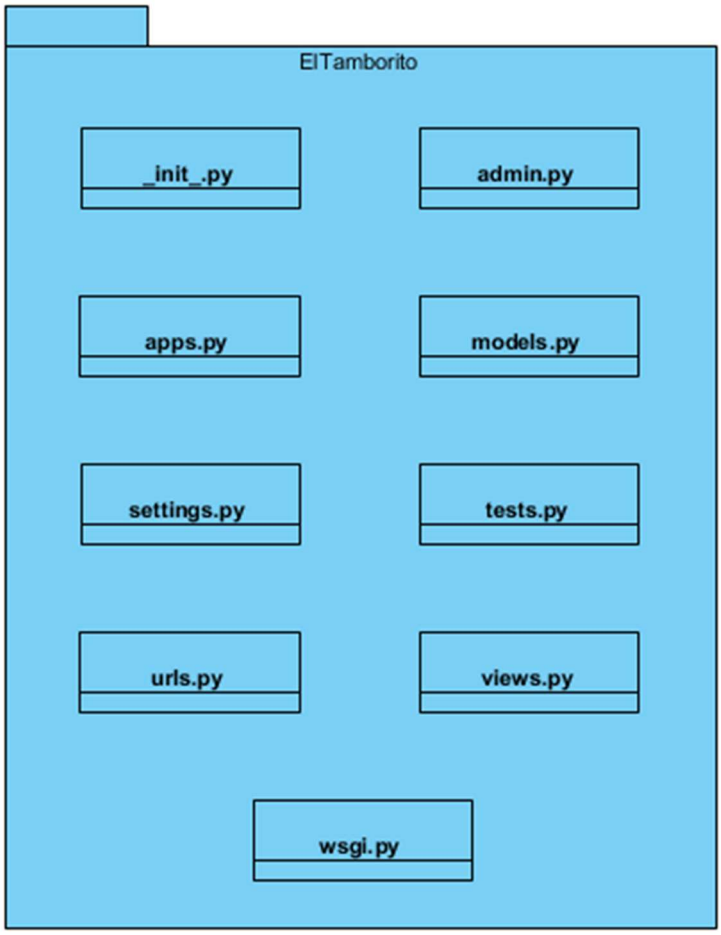


init.py	Pagina di configurazione python. (creata automaticamente dal framework)
admin.py	Rappresenta l'amministratore.
apps.py	Serve all'amministratore per configurare l'applicazione. (creata automaticamente dal framework)
models.py	Contiene tutti i model.
urls.py	Permette di far comunicare tutte le pagine.
views.py	Contiene tutte le view.
tests.py	Viene utilizzato per effettuare test nelle classi.

Il package eltambo contiene a sua volta tre sottopackage:

- static: contiene tutti i file statici come css, fonts, immagini)
- migrations: contiene i file necessari a tener traccia dei cambiamenti che vengono fatti nei model)
- templates: contiene tutte le pagine html della nostra applicazione)

3.3 Package ElTamborito



init.py	Pagina di configurazione python. (creata automaticamente dal framework)
admin.py	Rappresenta l'amministratore.
apps.py	Serve all'amministratore per configurare l'applicazione. (creata automaticamente dal framework)
models.py	Contiene tutti i model.
urls.py	Permette per far comunicare tutte le pagine.
views.py	Contiene tutte le view.
tests.py	Viene utilizzato per effettuare test nelle classi.
settings.py	Vi sono tutte le impostazioni essenziali all'applicazione.
wsgi.py	Serve all'applicazione per permettere all'amministratore di eseguire lo startup.

4. Interfacce delle classi

4.1 Models.py

I model presenti nell'implementazione del codice sono la fonte definitiva di informazione dei dati, ciò significa che essi contengono i comportamenti essenziali dei dati che si stanno memorizzando. Generalmente ogni modello si associa a una singola tabella di database, riportiamo un esempio (presente nel codice sorgente):

```
class Indirizzo (models.Model):
    nazione = models.CharField(max_length=50)
    citta = models.CharField(max_length=50)
    via = models.CharField(max_length=100)
    provincia = models.CharField(max_length=50)
    CAP = models.CharField(max_length=5)


class Utente (models.Model):
    email = models.EmailField(max_length=100, primary_key=True)
    nome = models.CharField(max_length=50)
    cognome = models.CharField(max_length=50)
    password = models.CharField(max_length=100)
    indirizzo = models.ManyToManyField(Indirizzo)
```

4.2 Interfacce di gestore

Per quanto riguarda le interfacce dedicate al gestore queste vengono generate in maniera automatica. Una delle più importanti caratteristiche di django è l'interfaccia di amministrazione automatica. Il framework legge i metadati definiti nei modelli e per fornire un'interfaccia rapida e centrata dove il gestore può appunto gestire i contenuti del suo sito. Ovviamente non sarà possibile modificare la parte front-end da parte del gestore.

Tutta via questi metodi pur essendo generati automaticamente necessitano di vincoli i quali saranno espressi con linguaggio naturale.

Visualizzazione lista utenti/singolo utente:

-pre: almeno un utente è presente nella tabelli utenti

-post: almeno un utente è visualizzato

Inserimento prodotto:

-pre: la forma proposta per l'inserimento di un prodotto contiene campi non nulli e corretti

-post: la tabella prodotti ha una nuova tupla con un prodotto disponibile alla vendita

Rimozione prodotto:

-pre: almeno un prodotto è presente nella tabella prodotti

-post: almeno un prodotto è stato eliminato (si intende quello selezionato)

Modifica prodotto:

-pre: almeno un prodotto è presente nella tabella dei prodotti, i campi da modificare sono non nulli e modificati correttamente

-post:almeno un prodotto è stato modificato (si intende quello selezionato)

Visualizza incassi:

-pre:/

-post: sono mostrati gli incassi

4.3 Views.py

In view.py sono presenti i metodi per caricare i template, in questa pagina risiede la logica dell'applicazione. La view richiede informazioni dal model e le passa ad un template (i metodi sono dichiarati con la notazione "def").

Homepage

- **Index:** visualizza il template della homepage

class ProductView

- **get_context_data:** visualizza tutti i prodotti della tabella prodotto al template.
  pre: prodotti->exists(p|p.ID!=null);
  post: prodotti->select(p|p.ID!=null);
- **InsertCart:** permette di inserire i prodotti al carrello
  pre: prodotti->exist(p|p.ID!=null)
  post: un prodotto è stato inserito nel carrello.

class ProductDetail

- **get_context_data:** visualizza I dettagli di un singolo prodotto al template.
  pre: prodotto**->** exists(p|(p.ID!=null) && p.caratteristiche!=null && p.tipologia!=null && p.modello!=null && p.data_prodotto!=null && p.prezzo!=null && p.disponibilità!=null);
  post: prodotto**->** select(p|(p.ID!=null) && p.caratteristiche!=null && p.tipologia!=null && p.modello!=null && p.data_prodotto!=null && p.prezzo!=null && p.disponibilità!=null);
- **cart:** visualizza i prodotti inseriti nel carrello.
  pre: /
  post: sono mostrati i prodotti inseriti nel carrello
- **contact:** visualizza il template della pagina relativa ai contatti
  pre:/
  post: è mostrata la pagina relativa ai contratti

class UserDetail

- **login:** permette l'accesso tremite un form da compilare, con email e password
  pre: email=!null && password=!null;
  post: Utente->exist(u|u.email == username && u.password == password);
- **signin:** permette la registrazione ad un utente non registrato
  pre: utente!=null && utente.email!=null && utente.nome!=null && utente.cognome !=null && cliente.password!=null;
  post: utenti->include(utente)

- **modificapass:** permette ad un utente registrato di modificare la password
  pre: utenti.password !=nul;
  post: utenti->update(utente.password);
- **modifimail:** permette ad un utente registrato di modificare la password
  pre: utenti.mail !=nul;
  post: utenti->update(utente.mail);

  acquisto: permette all'utente di acquistare i prodotti presenti nel carrello. Inserendo dettagli relativi all'indirizzo di spedizione.
  pre: indirizzo!=null && indirizzo.CAP !=null && indirizzo.via!=null && indirizzo.città=!null && indirizzo.provincia!=null && indirizzo.nazione !=null
  post: indirizzi->update(indirizzo)

5. Glossario

RAD: Documento di Analisi dei Requisiti.
DBMS: Sistema di gestione di basi di dati.
SDD: Documento di System Design.
ODD: Documento di Object Design.
Database: Insieme organizzato di dati persistenti.
Query: Termine utilizzato per indicare l'interrogazione da parte di un utente di un database Utente Registrato: Il termine identifica l'utente che ha effettuato la registrazione sul sistema.