Università degli Studi di Salerno

Corso di Ingegneria del Software

Ufficio Postale
Object Design Document

Versione 1.3

Data: 07/12/2017

Coordinatore del progetto:

Nome	Matricola
Andrea De Lucia	
Rita Francese	

Partecipanti:

Nome	Matricola
Sara Borriello	0512103468
Pasquale Scudieri	0512103702

Scritto da:	Sara Borriello e Pasquale Scudieri

Revision History

Data	Versione	Descrizione	Autore
07/12/2017	1.0	Prima stesura	Sara Borriello Pasquale Scudieri
21/12/2017	1.1	Modifiche	Pasquale Scudieri
28/12/2017	1.2	Modifiche	Sara Borriello
02/01/2018	1.3	Revisione	Sara Borriello Pasquale Scudieri

Indice

1. INTRODUZIONE

Dopo aver stilato il documento di Requirements Analysis e il documento di System Design in cui vi è una descrizione sommaria di ciò che sarà il nostro sistema, definendo i nostri obiettivi ma tralasciando gli aspetti implementativi, andiamo ora a stilare il documento di Object Design che ha come obiettivo quello di produrre un modello che sia in grado di integrare in modo coerente e preciso tutte le funzionalità individuate nelle fasi precedenti. In particolar modo, in tale documento si definiscono le interfacce delle classi, le operazioni, i tipi, gli argomenti e la signature dei sottosistemi definiti nel System Design.

Comprensibilità vs Tempo:
Il codice del sistema deve essere comprensibile il più possibile, in modo da facilitare testing e possibili modifiche da apportare in futuro. Per rispettare ciò il codice sarà ovviamente accompagnato da commenti volti a semplificare il tutto. Questo comporterà un aumento di tempo di sviluppo del nostro progetto.

Interfaccia vs Usabilità:
L'interfaccia grafica è stata realizzata in maniera molto semplice, chiara e concisa, vengono utilizzati i form e pulsanti con lo scopo di rendere semplice l'utilizzo del sistema da parte dell'utente finale.

Sicurezza vs Efficienza:
La sicurezza, come descritto nei requisiti non funzionali del Requirements Analysis, rappresenta uno degli aspetti importanti del sistema. Tuttavia, dati i tempi di sviluppo molto limitati, ci limiteremo ad implementare sistemi di sicurezza basati su username e password degli utenti.

1.1. Linee guida per la documentazione delle interfacce

Gli sviluppatori dovranno seguire determinate linee guida per la stesura del codice:

Convenzione nominativi:

È buona norma utilizzare nominativi:

- Descrittivi
- Pronunciabili
- Di lunghezza medio-corta
- Abbreviazioni non esagerate
- Utilizzare caratteri consentiti

Variabili:

- I nomi delle variabili devono iniziare con la lettera minuscola, e le parole successive con quella maiuscola. La dichiarazione delle variabili deve essere effettuata ad inizio blocco; ogni riga presenterà una sola variabile per migliorare la comprensione e per lo stesso motivo c'è bisogno di una corretta indentazione.

Metodi:

- I nomi dei metodi devono iniziare con la lettera minuscola, e le parole successive con la lettera maiuscola. Di solito il nome del metodo è costituito da un verbo che identifica un'azione, seguito dal nome di un oggetto.
- I nomi dei metodi per l'accesso e la modifica delle variabili dovranno essere del tipo getNome(), setNome().
- Ai metodi va aggiunta una descrizione, la quale deve essere posizionata prima della dichiarazione del metodo, e deve descriverne lo scopo.

Classi e pagine:

- I nomi delle classi e delle pagine devono iniziare con la lettera maiuscola, e anche le parole successive all'interno del nome devono iniziare con lettera maiuscola. I nomi delle classi e delle pagine devono essere evocativi, in modo da fornire informazioni

sullo scopo di quest'ultime.

- La dichiarazione di una classe è caratterizzata da:
    - Dichiarazione di costanti
    - Dichiarazione di variabili di classe
    - Dichiarazione di variabili d'istanza
    - Costruttore
    - Commento e dichiarazione metodi
    - Le variabili non vedono una descrizione in quanto nessuna variabile sarà visibile all'esterno, tutte le nostre variabili sono tenute interne alla classe.

1.2. Definizioni, acronimi e abbreviazioni

Acronimi:

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

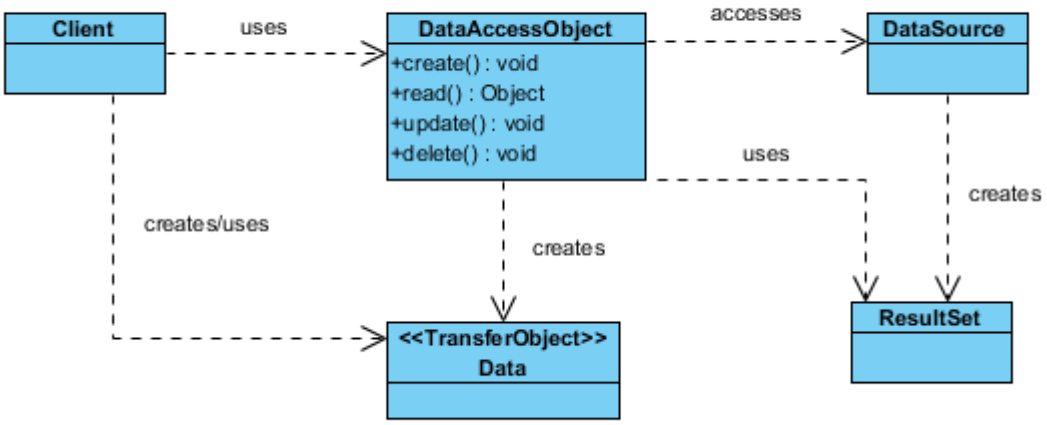
Abbreviazioni:

- **DB:** DataBase

1.3. Riferimenti

- B.Bruegge, A. H. Dutoit, Object Oriented Software Engineering – Using UML, Pattern and Java, Prentice Hail, 3rd edition, 2009.
- Documento RAD_Ufficio Postale.odt del progetto Ufficio Postale.
- Documento DatiPersistenti_UfficioPostale.odt del progetto Ufficio Postale.

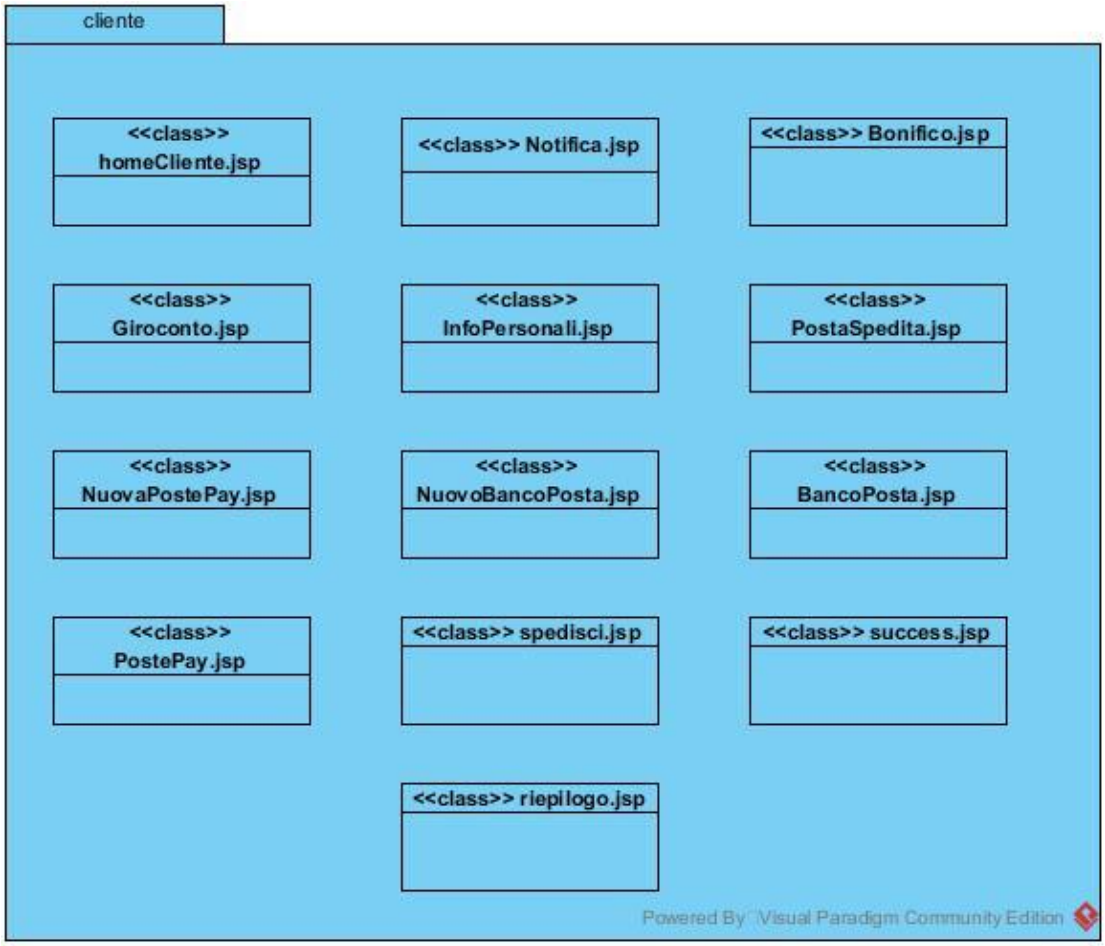
2. DESIGN PATTERN



Ufficio Postale fa uso del DAO Pattern. Si tratta di un pattern architetturale per la gestione della persistenza: fondamentalmente, è una classe con relativi metodi che rappresenta un'entità tabellare di un RDBMS, usata per stratificare e isolare l'accesso ad una tabella tramite query (poste all'interno dei metodi della classe) ovvero al data layer da parte della business logic creando un maggiore livello di astrazione e una facile manutenibilità. I metodi del DAO con le rispettive query verranno così richiamati dalle classi della business logic.
Il seguente design pattern è stato utilizzato nell'implementazione dei vari Model, dunque nelle seguenti classi: *BancoPostaModel, ClienteModel, ContoModel, DipendentiModel, NotificaModel, OperazioniModel, PacchiModel, PostaModel, PostePayModel, UtenteModel,* dove sono presenti dunque i vari metodi con le query per interrogare e riportare informazioni presenti nel database.

3. PACKAGE COMPONENTS

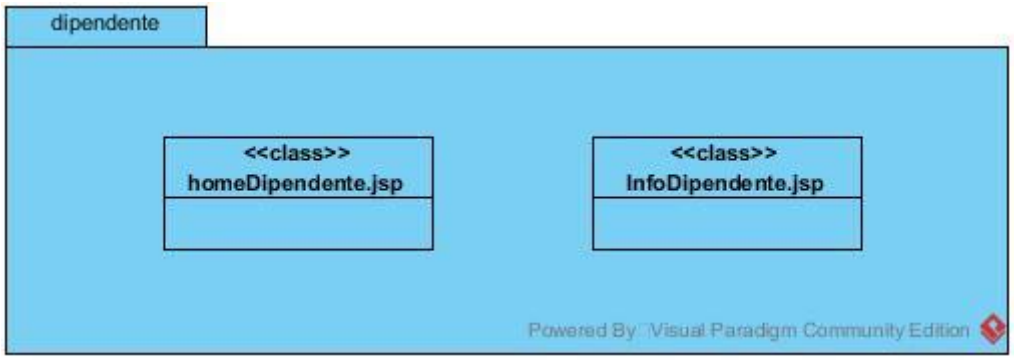
3.1. Package cliente



Classe:	Descrizione:
homeCliente.jsp	Si occupa di generare la pagina html che sarà la home page del cliente
Notifica.jsp	Si occupa di generare la pagina html per la visualizzazione di una notifica
Bonifico.jsp	Si occupa di generare la pagina html contenente la form per effettuare un bonifico. Permette, inoltre, di visualizzare i messaggi d'errore relativi al bonifico che si sta effettuando o il messaggio di conferma che l'operazione è stata portata a termine.
Girocnto.jsp	Si occupa di generare la pagina html contenente la form per effettuare un giroconto. Permette, inoltre, di visualizzare i messaggi d'errore relativi al giroconto che si sta tentando di effettuare o il messaggio di conferma che l'operazione è stata portata a termine.
InfoPersonali.jsp	Si occupa di costruire la pagina html che contiene le informazioni personali del cliente.
PostaSpedita.jsp	Si occupa di permettere al cliente di visualizzare la lista

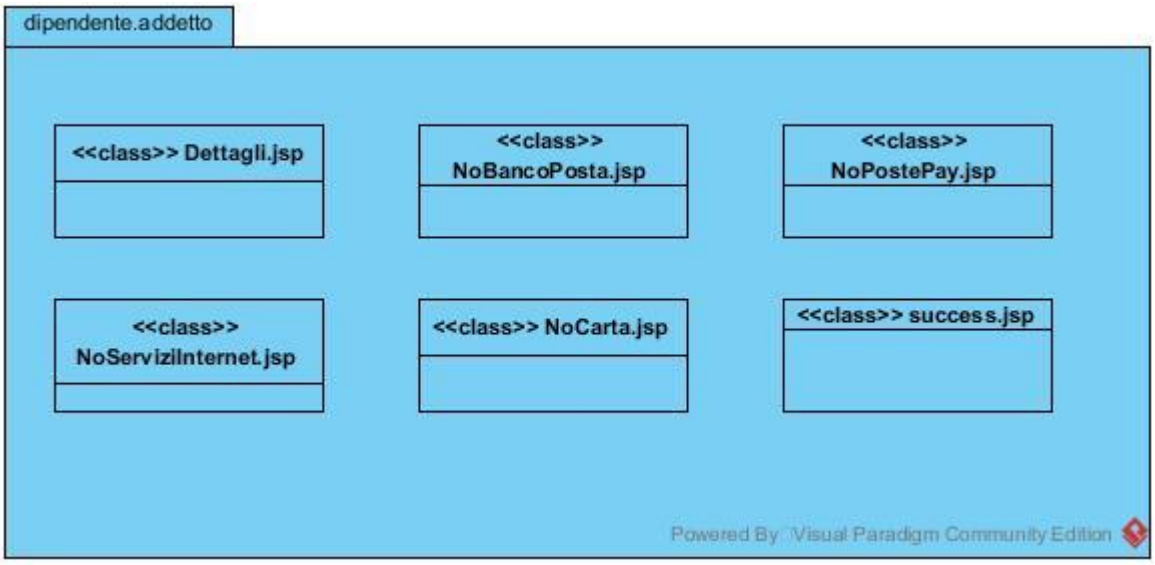
	contenente i dettagli sulla posta spedita
NuovaPostePay.jsp	Si occupa di generare la pagina html contenente la form per l'apertura di una nuova PostePay, di permettere la visualizzazione di un messaggio d'errore relativo all'errata compilazione della form
NuovoBancoPosta.jsp	Si occupa di generare la pagina html contenente la form per l'apertura di un nuovo conto BancoPosta, di permettere la visualizzazione di un messaggio d'errore relativo all'errata compilazione della form
BancoPosta.jsp	Si occupa di generare la pagina html che contiene i dati relativi ad un conto BancoPosta. Nel caso in cui ci siano servizi non attivi sul conto la pagina generata conterrà i bottoni per attivarli
PostePay.jsp	Si occupa di generare la pagina html che contiene i dati relativi ad una carta PostePay
Spedisci.jsp	Si occupa di generare la pagina html che contiene la form per permettere l'inserimento dei dati relativi ad una spedizione. Permette, inoltre di visualizzare i messaggi d'errore relativi all'errata compilazione della form.
Riepilogo.jsp	Si occupa di visualizzare l'elenco delle informazioni appena inserite.
Success.jsp	Si occupa di notificare al cliente che l'azione appena compiuta è stata effettuata con successo

3.2. Package dipendente



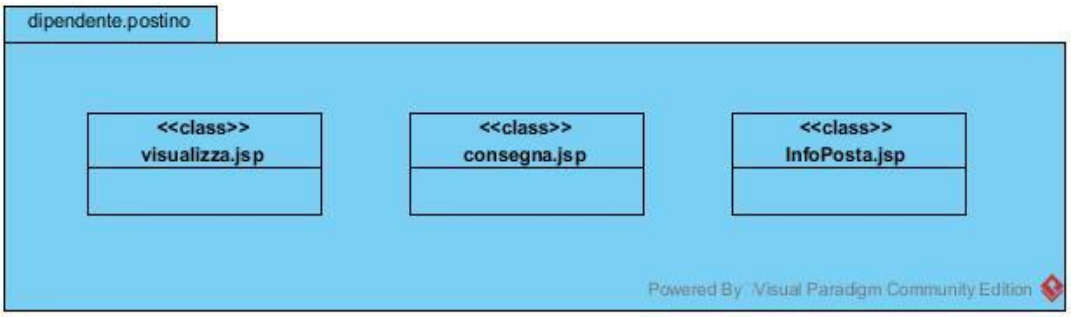
Classe:	Descrizione:
homeDipendente.jsp	Si occupa di visualizzare la home page di un dipendente sia esso un addetto allo sportello o un postino.
infoDipendente.jsp	Si occupa di generare la pagina html contenente le informazioni personali del dipendente.

3.3. Package dipendente.addetto



Classe:	Descrizione:
NoBancoPosta.jsp	Si occupa di generare una pagina html che contiene l'elenco dei clienti che non hanno un conto BancoPosta. Per ogni cliente presenta due bottoni, uno per inviare direttamente la notifica e uno per visualizzarne i dettagli.
NoPostePay.jsp	Si occupa di generare una pagina html che contiene l'elenco dei clienti che non hanno una PostePay. Per ogni cliente presenta due bottoni, uno per inviare direttamente la notifica e uno per visualizzarne i dettagli.
NoServiziInternet.jsp	Si occupa di generare una pagina html che contiene l'elenco dei clienti che hanno un conto BancoPosta senza i servizi internet attivi. Per ogni cliente presenta due bottoni, uno per inviare direttamente la e uno per visualizzarne i dettagli.
NoCarta.jsp	Si occupa di generare una pagina html che contiene l'elenco dei clienti che hanno un conto BancoPosta cui non è associata alcuna carta bancomat. Per ogni cliente presenta due bottoni, uno per inviare direttamente la notifica e uno per visualizzarne i dettagli.
Dettagli.jsp	Si occupa della visualizzazione delle informazioni relative ad un cliente. Presenta, inoltre, un bottone che permette l'invio di una notifica.
Success.jsp	Si occupa di visualizzare un messaggio che indica che la notifica è stata effettuata con successo.

3.4. Package dipendente.postino

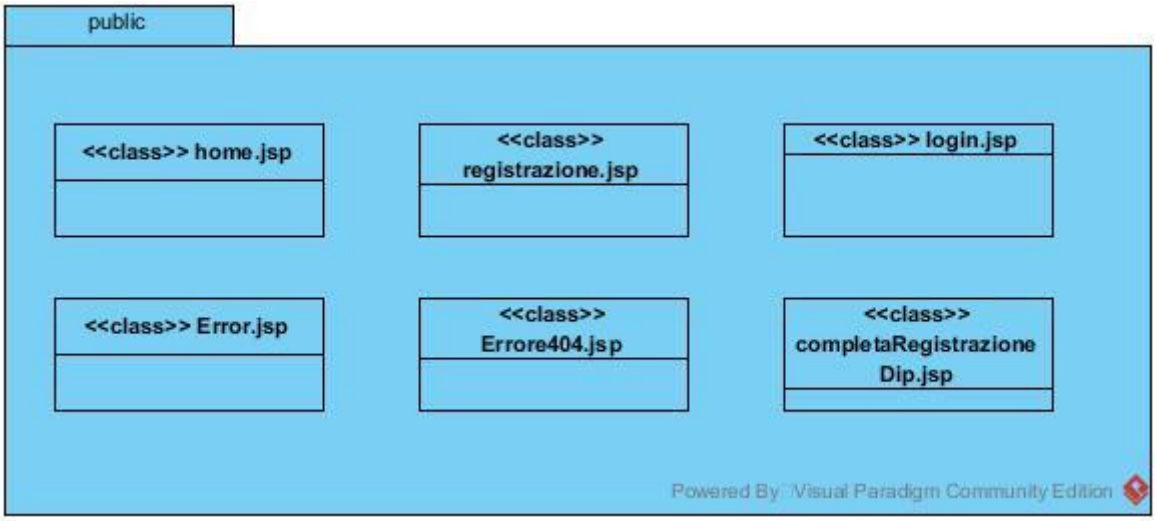


Classe:	Descrizione:
visualizza.jsp	Si occupa di generare la pagina html che contiene la form per l'inserimento del codice della posta di cui si vogliono conoscere i dettagli. Inoltre, permette la visualizzazione di un messaggio d'errore nel caso in cui il codice immesso non sia valido.
Consegna.jsp	Si occupa di generare la pagina html che contiene la form per l'inserimento del codice della posta consegnata. Permette la visualizzazione di messaggi d'errore nel caso in cui il codice sia non valido.
InfoPosta.jsp	Si occupa di visualizzare i dettagli di una data spedizione.

3.5. Package gestore

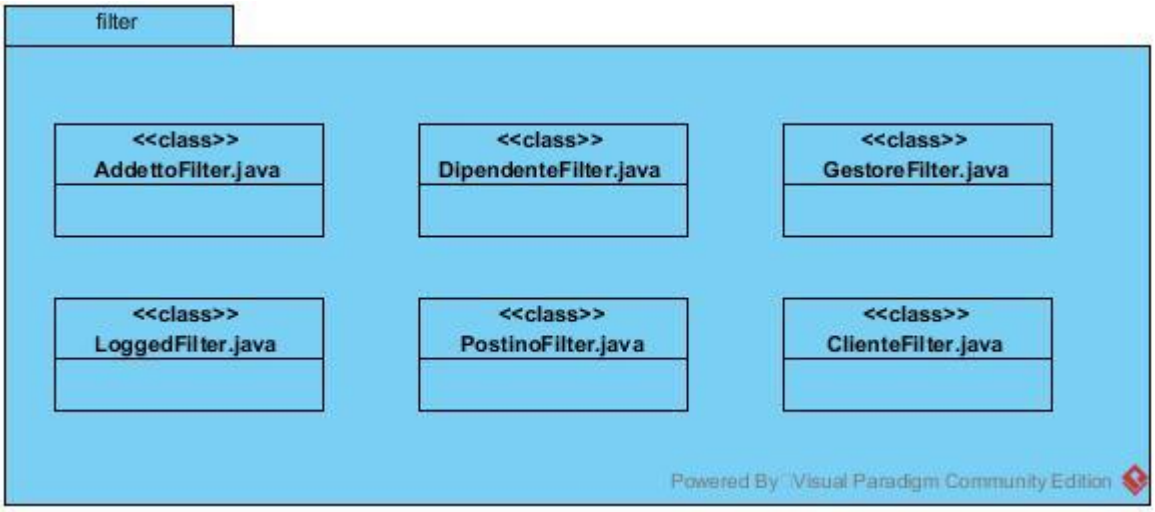
Classe:	Descrizione:
homeGestore.jsp	Si occupa della visualizzazione della home page del gestore.
RegistrazioneDipendenti.jsp	Si occupa di generare la pagina html che contiene la form per la registrazione di un nuovo dipendente. Permette, inoltre di visualizzare messaggi d'errore nel caso in cui i dati inseriti no siano corretti
RiepilogoInfo.jsp	Si occupa di visualizzare l'elenco delle informazioni di un dipendente appena inserito

3.6. Package pubblico



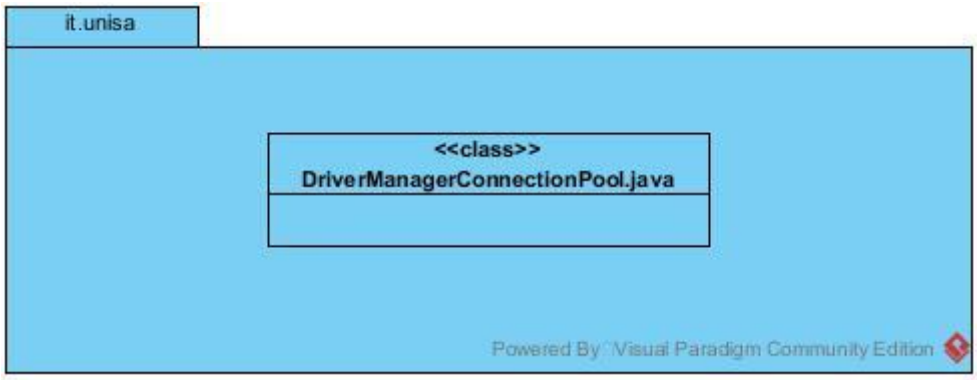
Classe:	Descrizione:
home.jsp	Si occupa di generare la home page del sito UfficioPostale.
Login.jsp	Si occupa della visualizzazione della pagina contenente la form per effettuare il login. Permette, inoltre, la visualizzazione di messaggi d'errore nel caso in cui i dati inseriti siano scorretti.
Registrazione.jsp	Si occupa di generare la pagina html contenente la form per effettuare la registrazione come cliente. Consente anche di visualizzare messaggi d'errore nel caso in cui i valori inseriti non siano ammissibili.
Error.jsp	Si occupa di visualizzare un messaggio d'errore generico per informare l'utente che si è verificata una condizione in attesa
Errore404.jsp	Si occupa di generare la pagina html associata all'errore 404.
completaRegistrazioneDip.jsp	Si occupa di generare la pagina html contenente la form per permettere al dipendente di completare la registrazione

3.7. Package filter



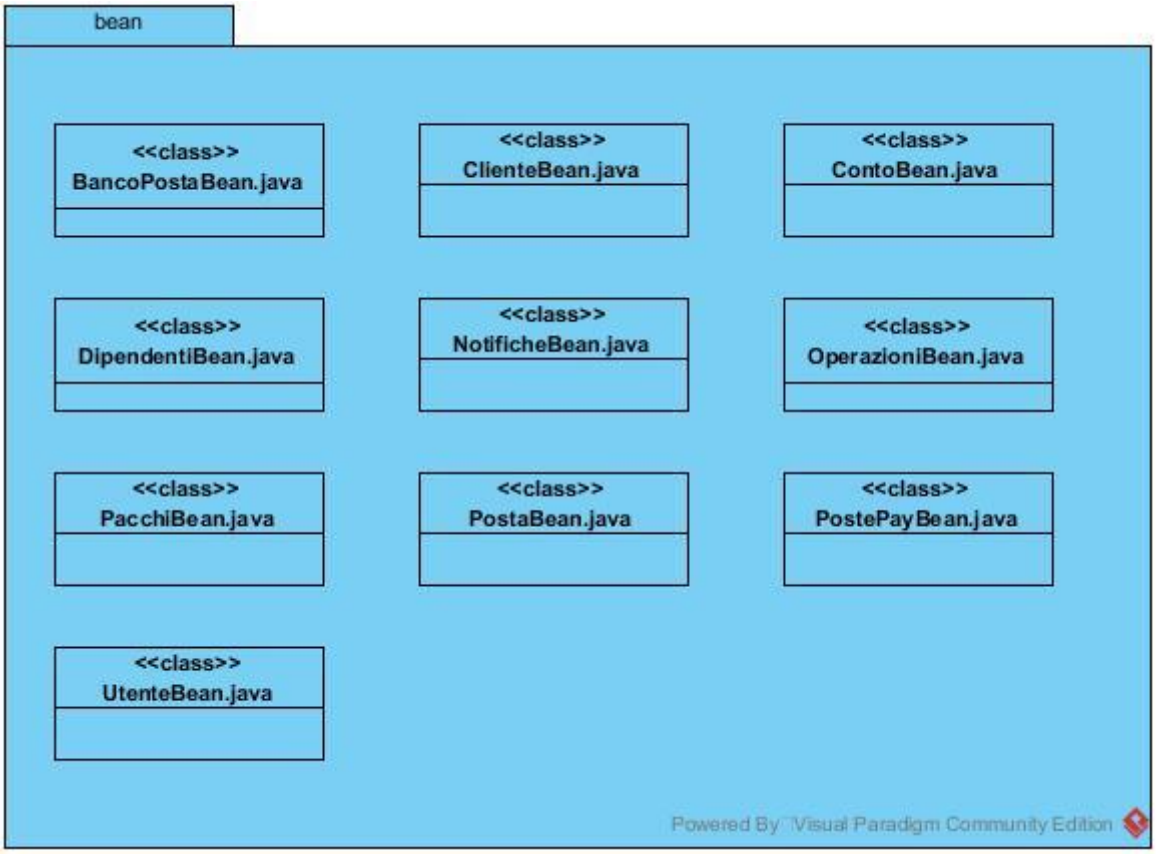
Classe:	Descrizione:
AddettoFilter.java	Questo filtro si occupa di assicurare che solo gli utenti registrati come addetti allo sportello accedano alle funzionalità loro dedicate.
DipendenteFilter.java	Questo filtro fa in modo che solo gli utenti registrati come dipendenti possano accedere all'area loro dedicata. Per quanto riguarda, invece, la distinzione tra postini e addetti allo sportello si rimanda alla descrizione di PostinoFilter e AddettoFilter
GestoreFilter.java	Questo filtro fa in modo che solo il gestore possa accedere all'area a lui dedicata.
LoggedFilter.java	Questo filtro si occupa di fare in modo che un utente non loggato acceda a pagine per le quali è necessaria l'autenticazione
PostinoFilter.java	Questo filtro fa in modo che solo i dipendenti registrati come postini possano accedere all'area dedicata dai postini
ClienteFilter.java	Questo filtro si occupa di fare in modo che solo gli utenti registrati come clienti posano accedere alle pagine dell'area personale dei clienti.

3.8. Package it.unisa



Classe:	Descrizione:
DriverManagerConnectionPool.java	Questa classe si occupa di gestire una pool di connessioni al database.

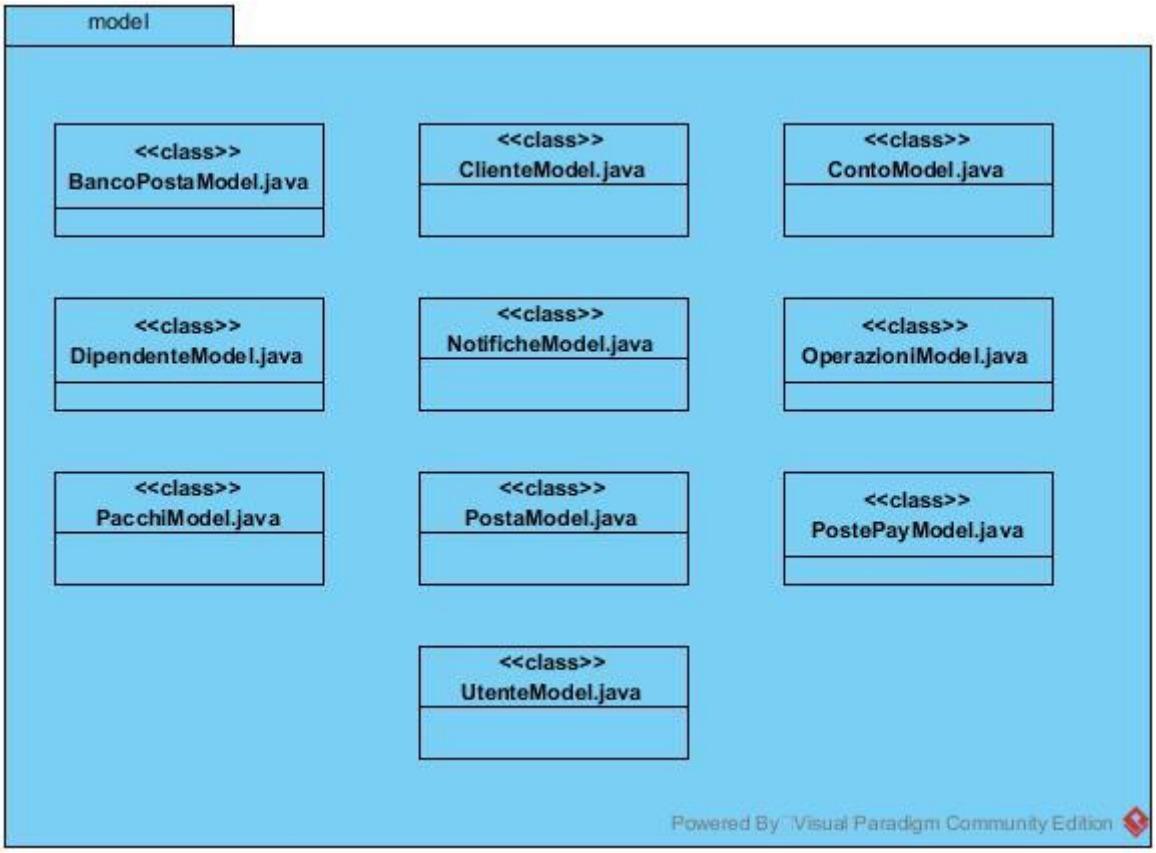
3.9. Package bean



Classe:	Descrizione:
BamcoPostaBean.java	Questa classe rappresenta un conto BancoPosta
ClienteBean.java	Questa classe rappresenta un cliente
ContoBean.java	Questa classe rappresenta un conto
DipendentiBean.java	Questa classe rappresenta un dipendente

NotificheBean.java	Questa classe rappresenta una notifica
OperazioniBean.java	Questa classe rappresenta un'operazione
PacchiBean.java	Questa classe rappresenta un pacco
PostaBean.java	Questa classe rappresenta la posta
PostePayBean.java	Questa classe rappresenta un conto PostePay
UtenteBean.java	Questa classe rappresenta un utente registrato

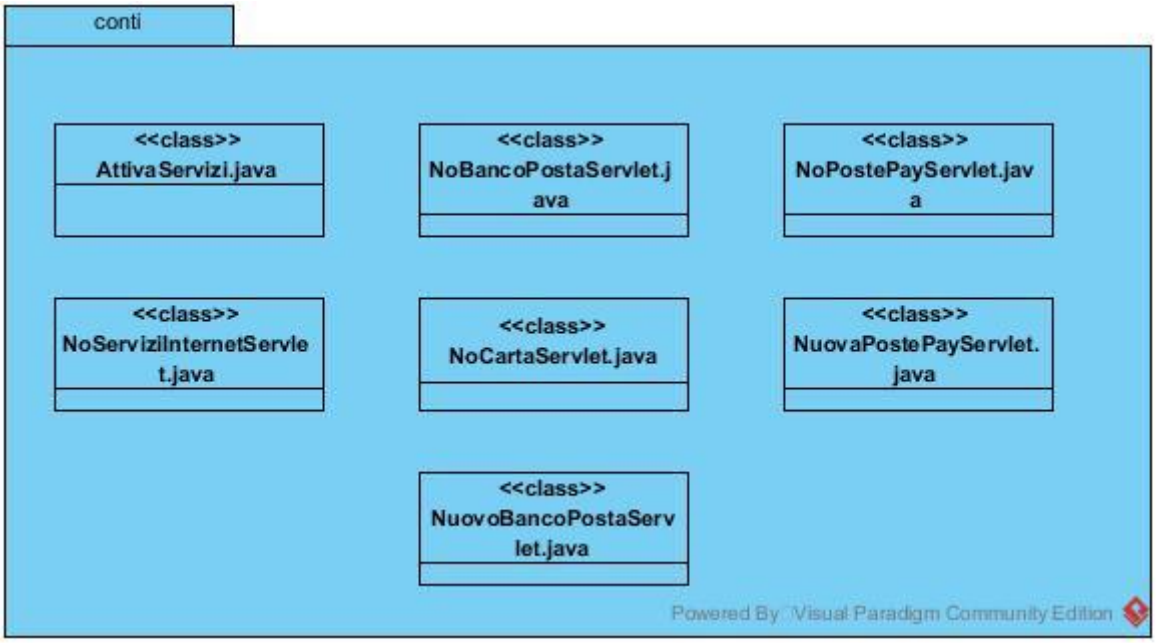
3.10. Package model



Classe:	Descrizione:
BamcoPostaModel.java	Questa classe contiene metodi che permettono di effettuare inserimenti, ricerche e update riguardanti i conti BancoPosta.
ClienteModel.java	Questa classe contiene metodi che permettono di effettuare inserimenti, ricerche e update riguardanti i clienti
ContoModel.java	Questa classe contiene metodi che permettono di effettuare query sul database che riguardano i conti
DipendentiModel.java	Questa classe contiene metodi che permettono di effettuare query sul database che restituiscono un DipendenteBean o permettono l'inserimento di un nuovo dipendente
NotificheModel.java	Questa classe permette l'inserimento e la cancellazione di notifiche all'interno del database
OperazioniModel.java	Questa classe permette di inserire una nuova tupla nella

	tabella operazioni oppure di ottenere la lista delle operazioni effettuate su un conto
PacchiModel.java	Questa classe permette di effettuare l'inserimento di una nuova tupla nella tabella pacchi oppure di ottere informazioni su pacchi già presenti nel database
PostaModel.java	Questa classe contiene metodi per effettuare l'inserimento nel database delle informazioni sulla posta, farne l'update ad esempio per inserire le informazioni sulla consegna. Permette inoltre di ottenere i dati di una singola posta o l'elenco della posta spedita da un cliente
PostePayModel.java	Questa classe permette di effettuare l'inserimento dei dati riguardanti una PostePay o di ottenere l'elenco delle PostePay di un cliente
UtenteModel.java	Questa classe si occupa di effettuare inserimenti e ricerche sulla tabella UtenteRegistrato presente nel database

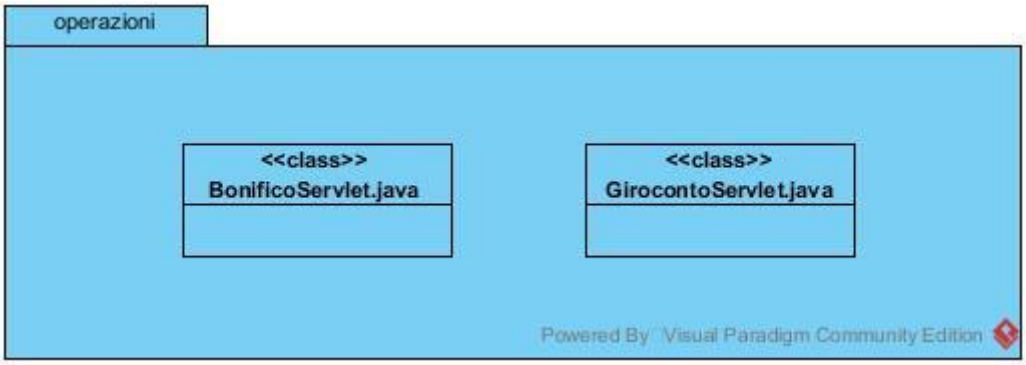
3.11. Package conti



Classe:	Descrizione:
NuovoBancoPostaServlet.java	Questa servlet riceve i dati dalla classe NuovoBancoPosta.jsp e li processa. Nel caso siano corretti utilizza i servizi di ContoModel e BancoPostaModel per effettuare l'inserimento.
NuovaPosrtePayServlet.java	Questa servlet riceve i dati dalla classe NuovaPostePayServlet.jsp e li processa. Nel caso siano corretti utilizza i servizi di ContoModel e PostePayModel per effettuare l'inserimento.
AttivaServizi.java	Questa servlet richiede i servizi di BancoPostaModel per associare una carta bancomat ad un conto o per effettuare l'attivazione dei servizi internet.
NoBancoPostaServlet.java	Questa servlet si occupa di mandare una notifica ad un

	cliente che non ha un BancoPosta usando i servizi di NotificaModel
NoPostePayServlet.java	Questa servlet viene invocata da un addetto allo sportello quando questi decide di mandare una notifica ad un cliente che non possiede una PostePay.
NoServiziInternetServlet.java	Questa servlet viene ha il compito di inviare una notifica ad un cliente che non ha i servizi internet attivi su un conto BancoPosta
NoCartaServlet.java	Questa servlet utilizza i servizi di NotificaModel per effettuare l'invio di una notifica ad un cliente che ha un conto Bancoposta a cui non è associata nessuna carta bancomat

3.12. Package operazioni



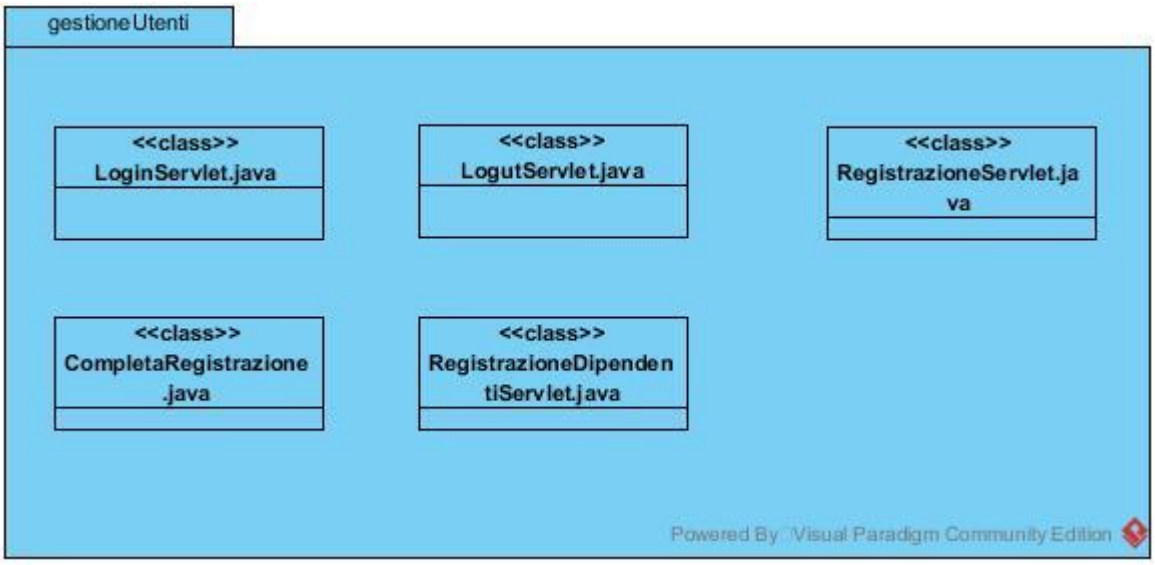
Classe:	Descrizione:
BonificoServlet.java	Questa servlet riceve i dati da Bonifico.jsp, li processa e nel caso i valori risultino ammissibili utilizza i servizi di OperazioniModel per effettare l'inserimento.
GirocontoServlet.java	Questa servlet riceve i dati da Giroconto.jsp, li processa e nel caso i valori risultino ammissibili effettua inserimento .

3.13. Package notifica

Classe:	Descrizione:
EliminaNotifica.java	Questa servlet si occupa di portare a termine l'eliminazione di una notifica una volta che questa è stata visualizzata da

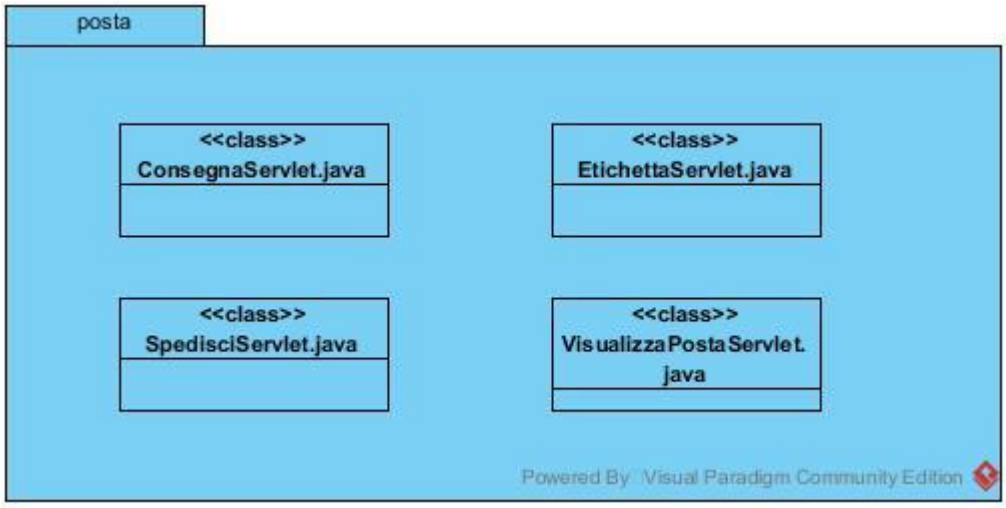
	un cliente. Per fare questo utilizza i servizi offerti dalla classe NotificaModel.

3.14. Package gestioneUtenti



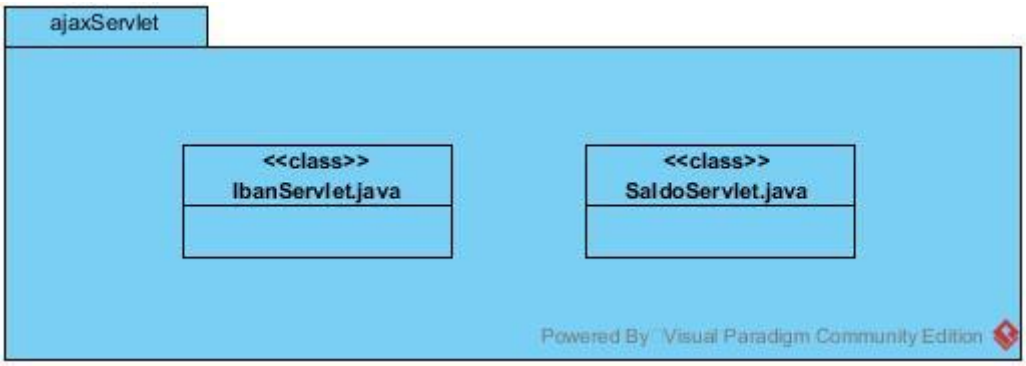
Classe:	Descrizione:
LoginServlet.java	Questa servlet si occupa di ricevere i dati dalla jsp di login elaborarli e decidere se consentire o no l'accesso all'area personale.
LogoutServlet.java	Questa servlet si occupa di invalidare la sessione per consentire il logut
RegistrazioneServlet.java	Questa servlet riceve di dati da Registrazione.jsp e li elabora. Se i dati sono corretti utilizza i servizi di UtenteModel e ClienteModel per effettuare la registrazione, altrimenti passa alla jsp un messaggio d'errore.
RegistrazioneDipendenteServlet.java	Questa servlet riceve di dati da RegistrazioneDipendenti.jsp e li elabora. Se i dati sono corretti utilizza i servizi di UtenteModel e DipendenteModel per effettuare la registrazione, altrimenti passa alla jsp un messaggio d'errore.
CompletaRegistrazione.java	Questa servlet riceve i dati da completaRegistrazioneDip.jsp e utilizza i servizi di utente model e dipendente model per completare la registrazione.

3.15. Package posta



Classe:	Descrizione:
ConsegnaServlet.java	Questa servlet riceve i dati da consegna.jsp, li processa e, nel caso risultino validi, richiede i servizi di PostaModel per indicare che la posta con quel codice è stata consegnata dal postino in una certa data.
EtichettaServlet.java	Questa servlet ha il compito di costruire il documento pdf che costituisce l'etichetta che il cliente dovrà stampare
SpedisciServlet.java	Questa servlet riceve i dati da spedisci.jsp e nel caso siano validi ne effettua l'inserimento usando i servizi di PostaModel e pacchiModel.
VisualizzaPostaServlet.java	Questa servlet riceve i dati visualizza.jsp, li processa e, nel caso risultino ammissibili, recupera i dati desiderati e li passa a infoPosta.jsp

3.16. Package ajax

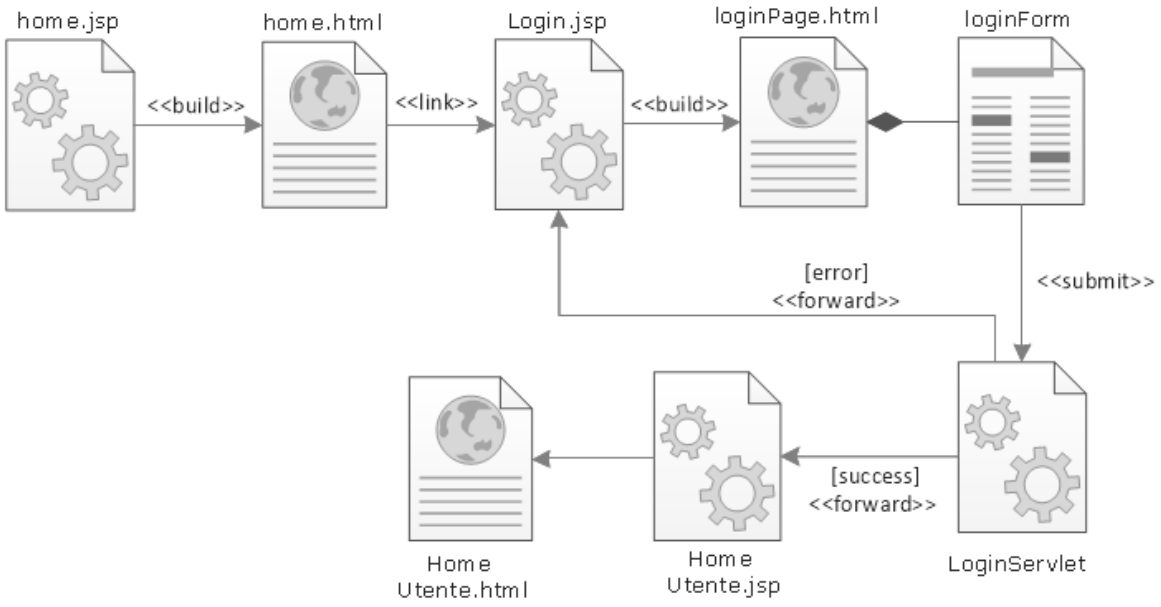


Classe:	Descrizione:
IbanServlet.java	Questa servlet serve a portare a termine la chiamata ajax che permette nella pagina di Giroconto di non avere l'iban selezionato nell'altra select
SaldoServlet.java	Questa servlet serve a portare a termine la chiamata ajax che permette nella pagina di Giroconto o Bancoposta di visualizzare il saldo corrispondente all'iban selezionato

4. DIAGRAMMI A RUN TIME

A completamento dei diagrammi a design time presenti nell'SDD si presentano qui i diagrammi a run time per rendere più chiara l'interazione tra le varie componenti.

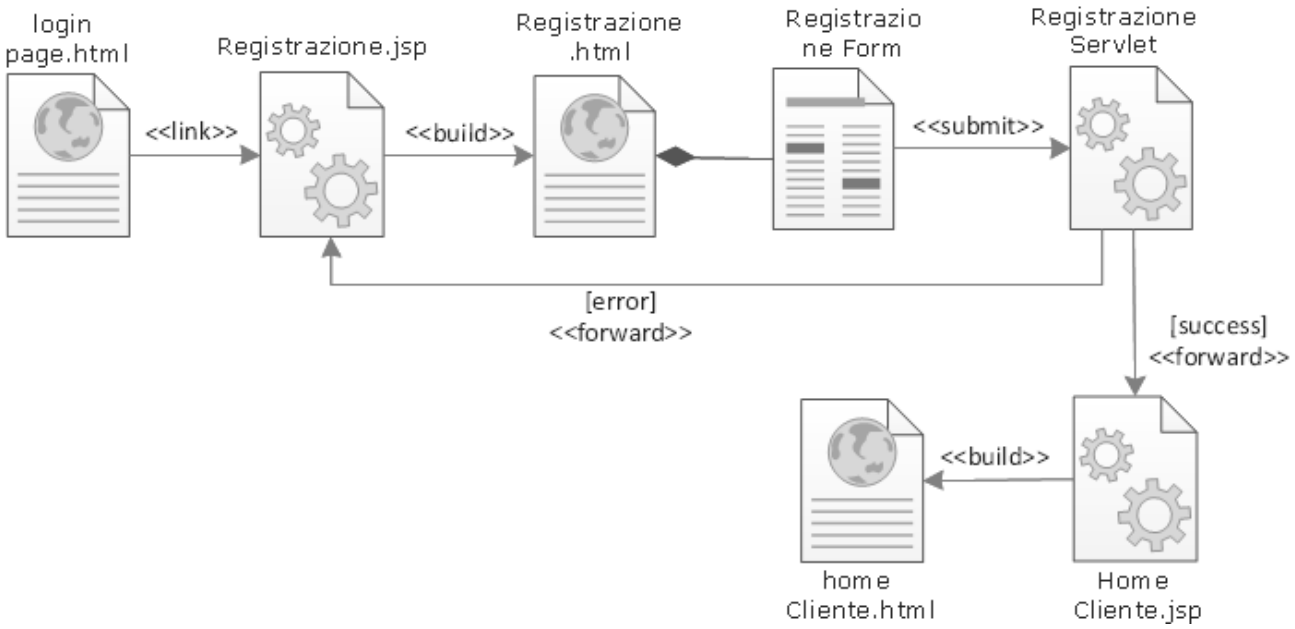
Login



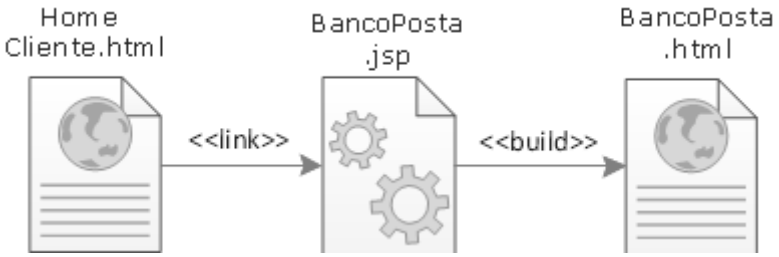
Per evitare di costruire un diagramma per ogni tipo di utente abbiamo utilizzato la notazione "Home utente.jsp". in realtà la jsp che viene utilizzata è differente per ogni utente che effettua il login al sito:

- homeCliente.jps per il cliente;
- homeDipendente per il dipendente;
- homeGestore per il gestore;

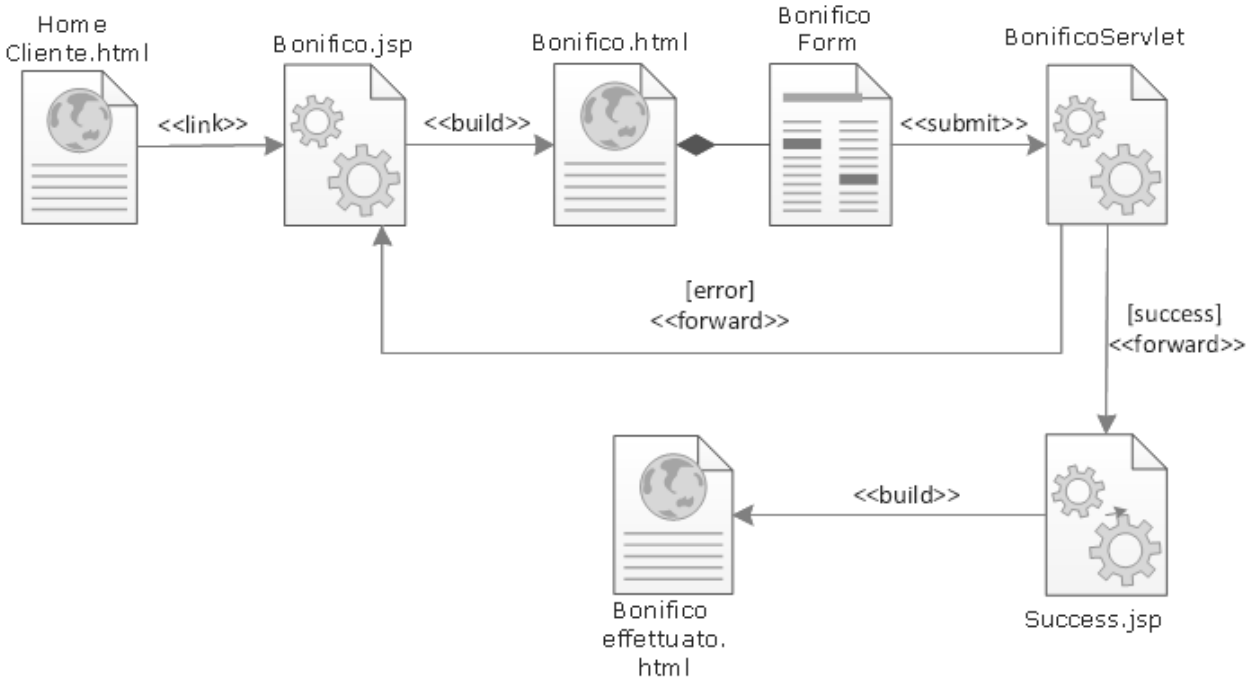
Registrazione cliente



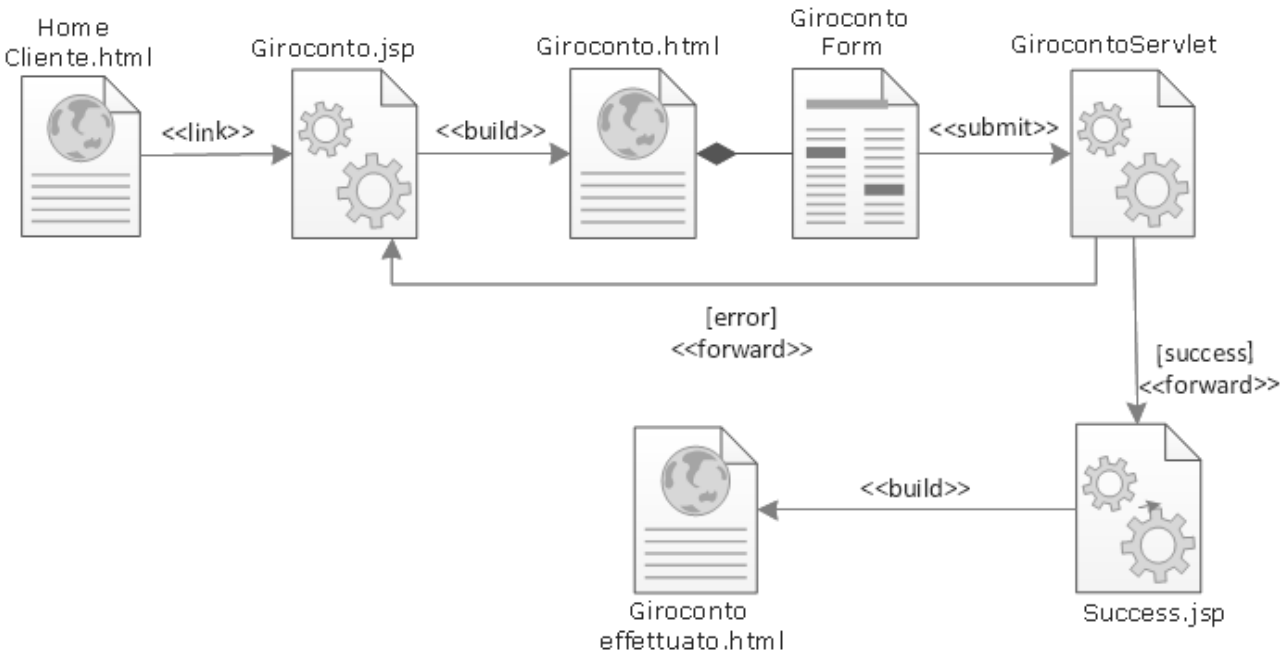
Visualizza informazioni su un BancoPosta



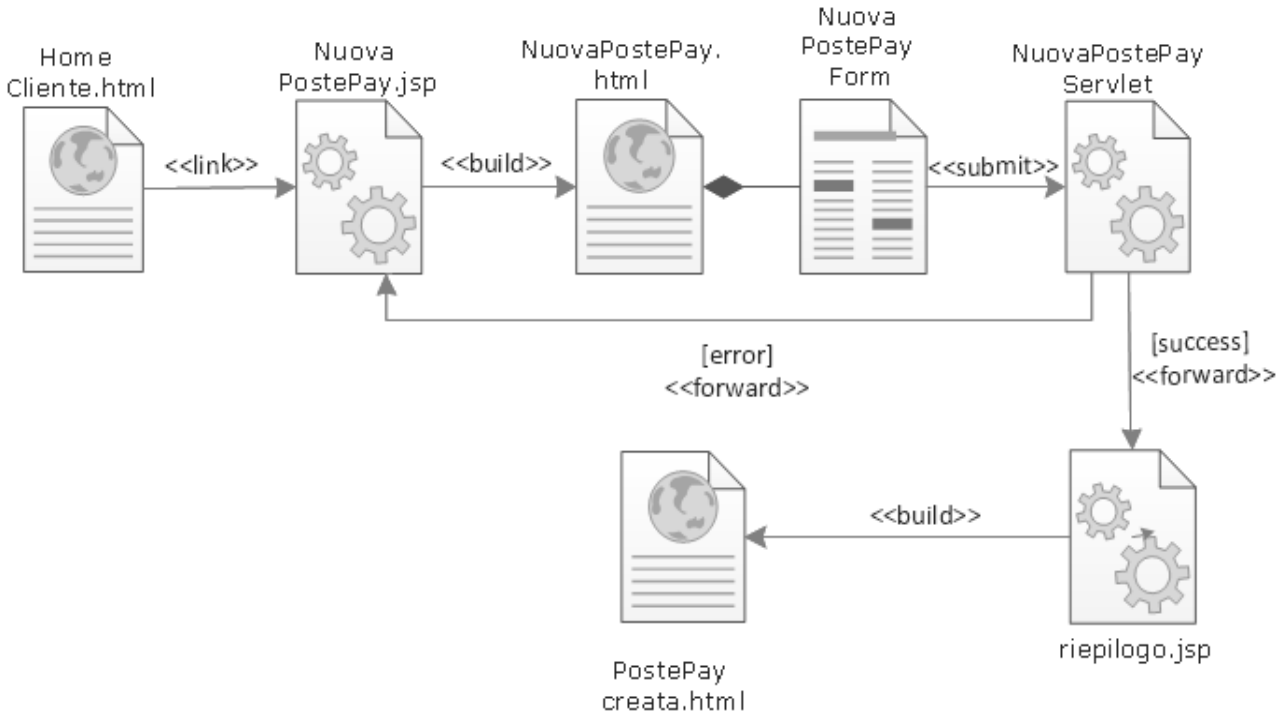
Bonifico



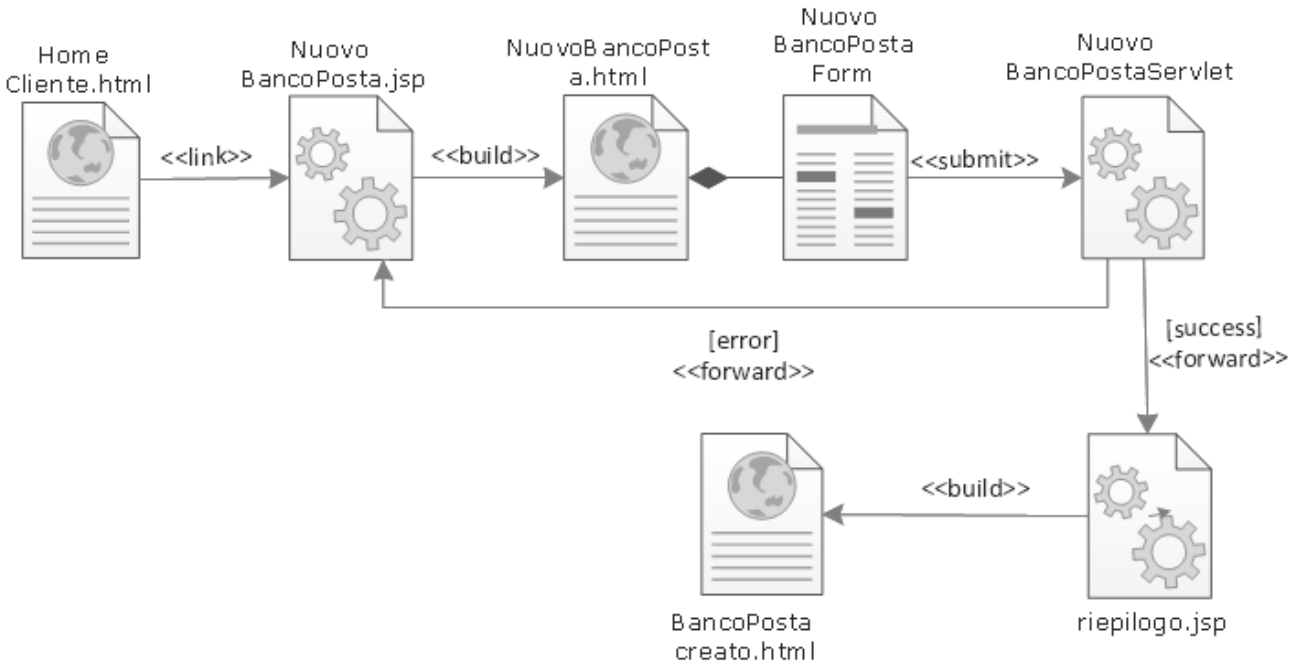
Giroconto



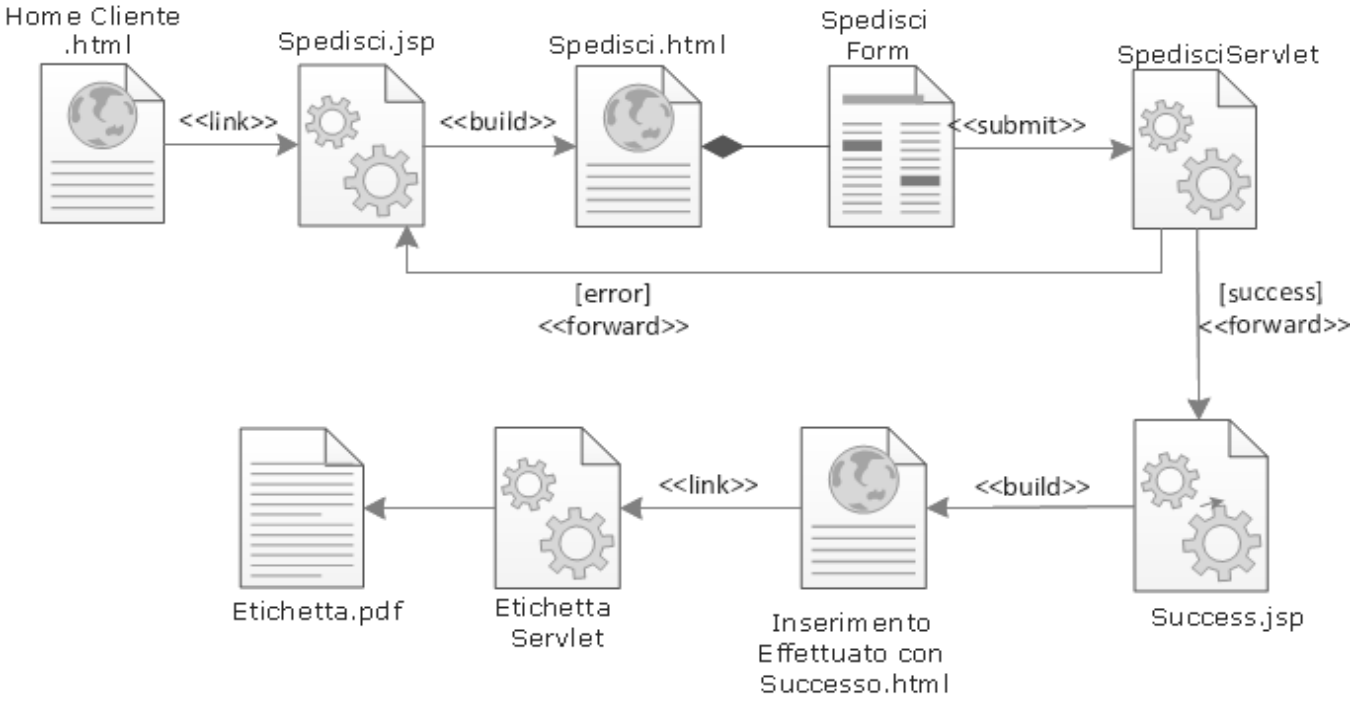
Nuova PostePay



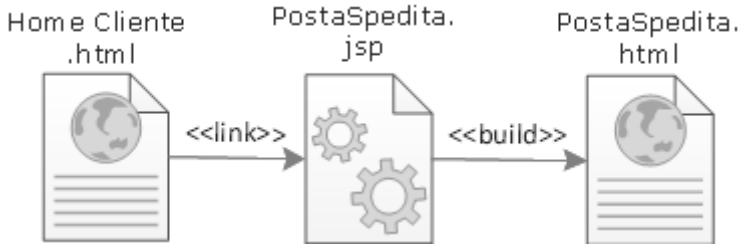
Nuovo BancoPosta



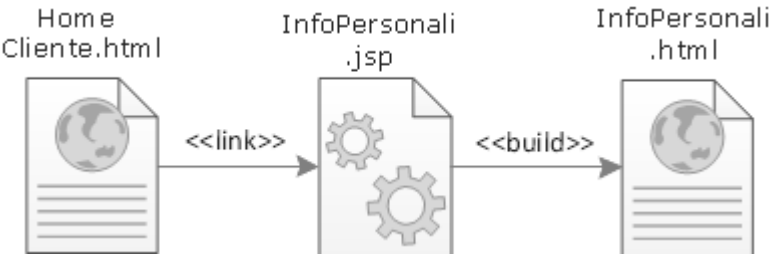
Inserimento delle informazioni su una spedizione



Visualizzazione della posta spedita

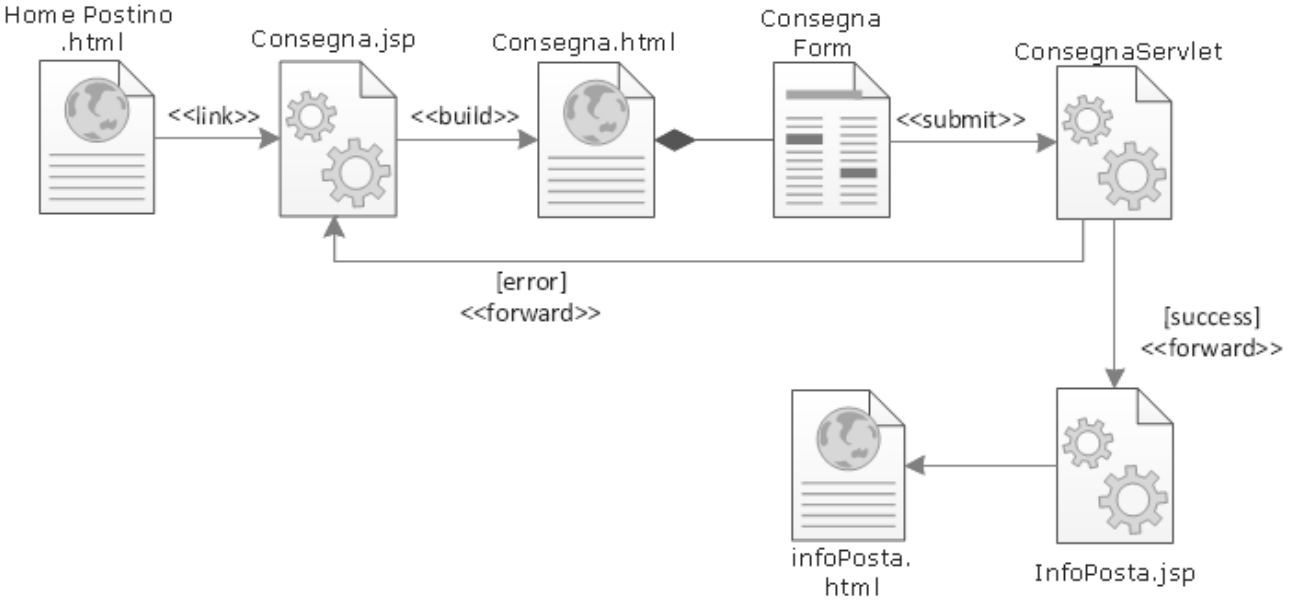


Visualizzazione informazioni personali

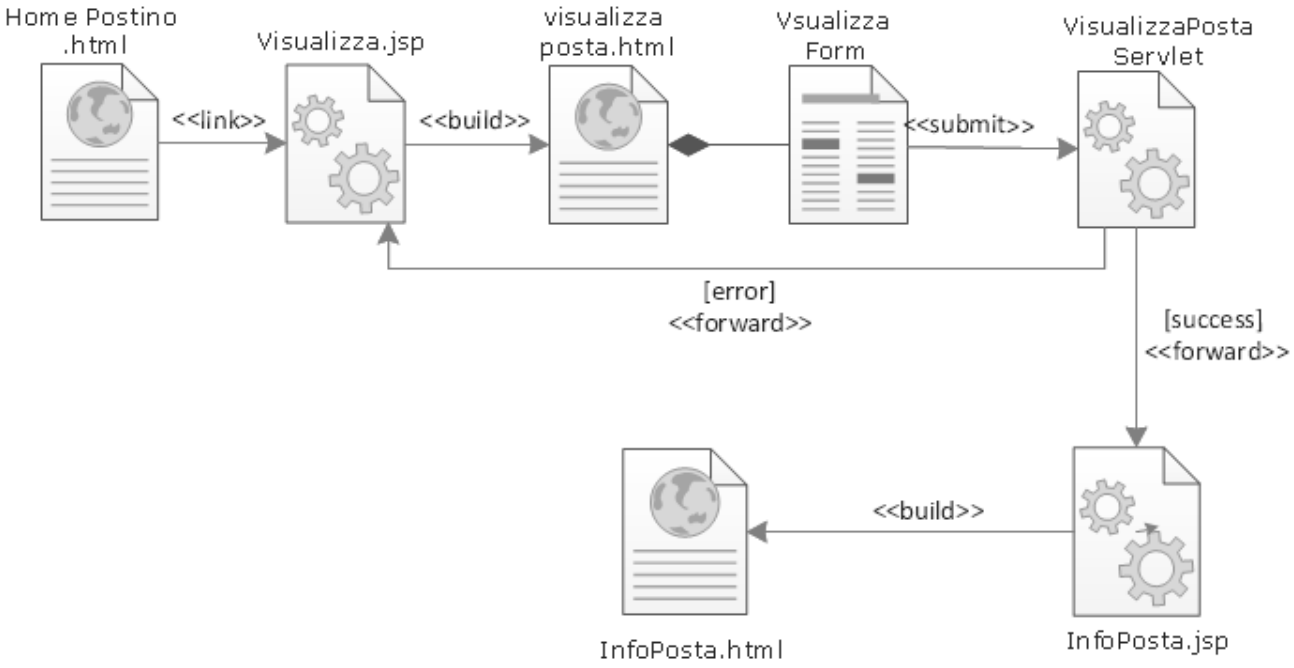


Abbiamo preso in considerazione solo il caso del cliente, ma per il resto degli utenti funziona nel medesimo modo, con la differenza di iniziare il tutto dalle rispettive home.

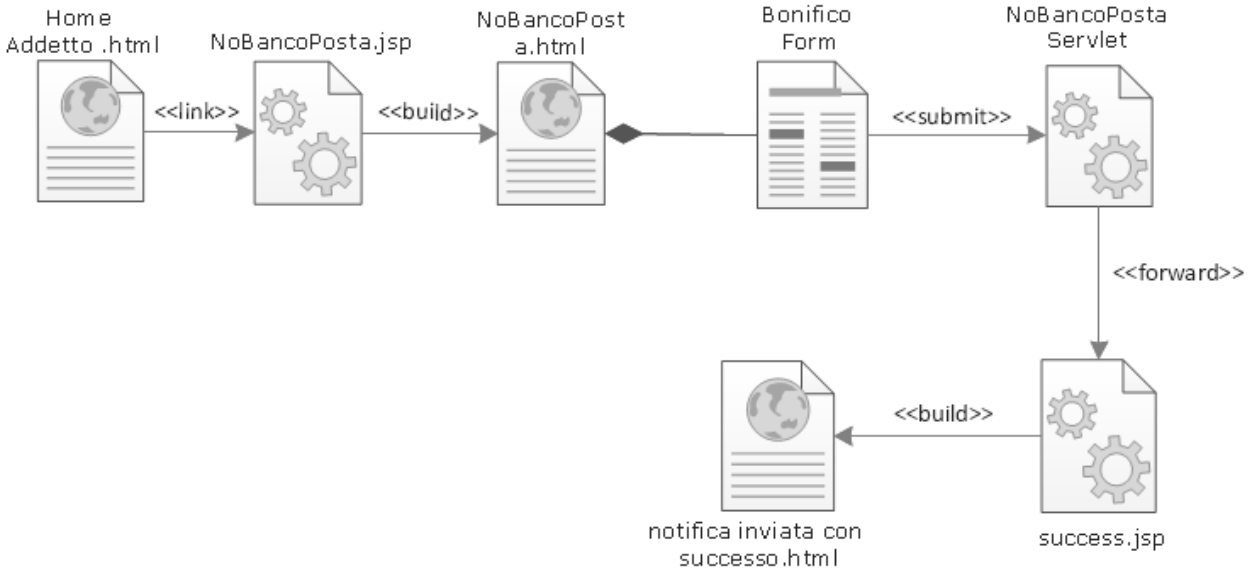
Consegna da parte di un postino



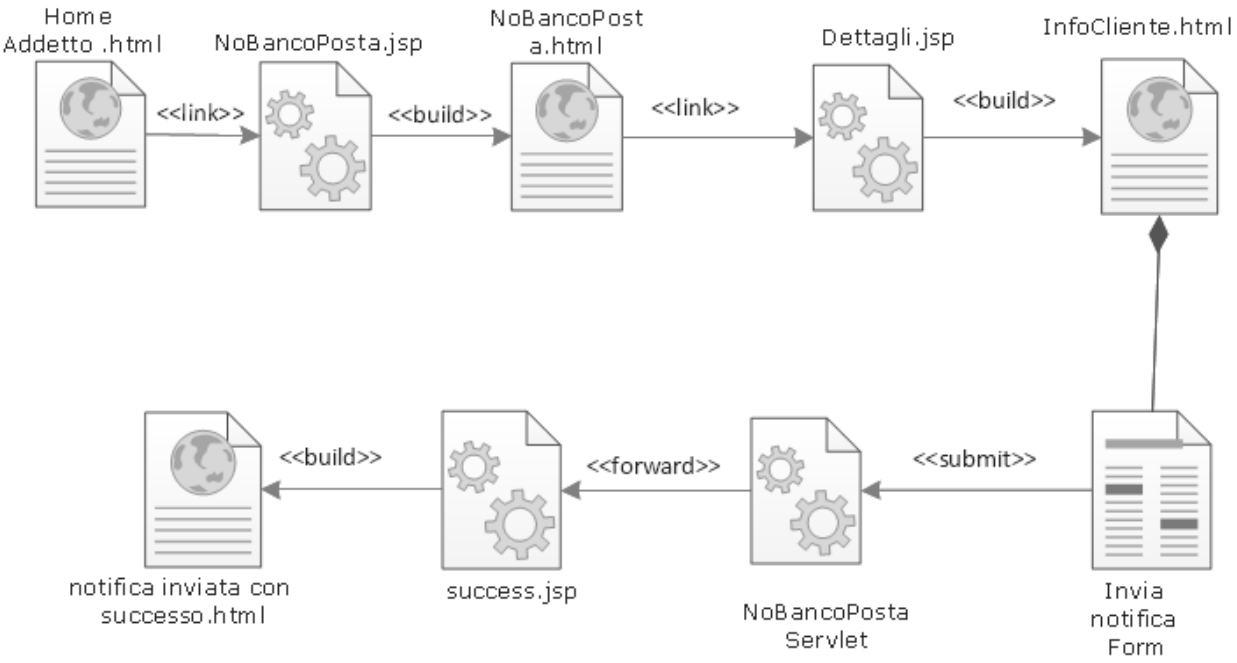
Visualizzazione delle informazioni sulla posta da parte di un postino



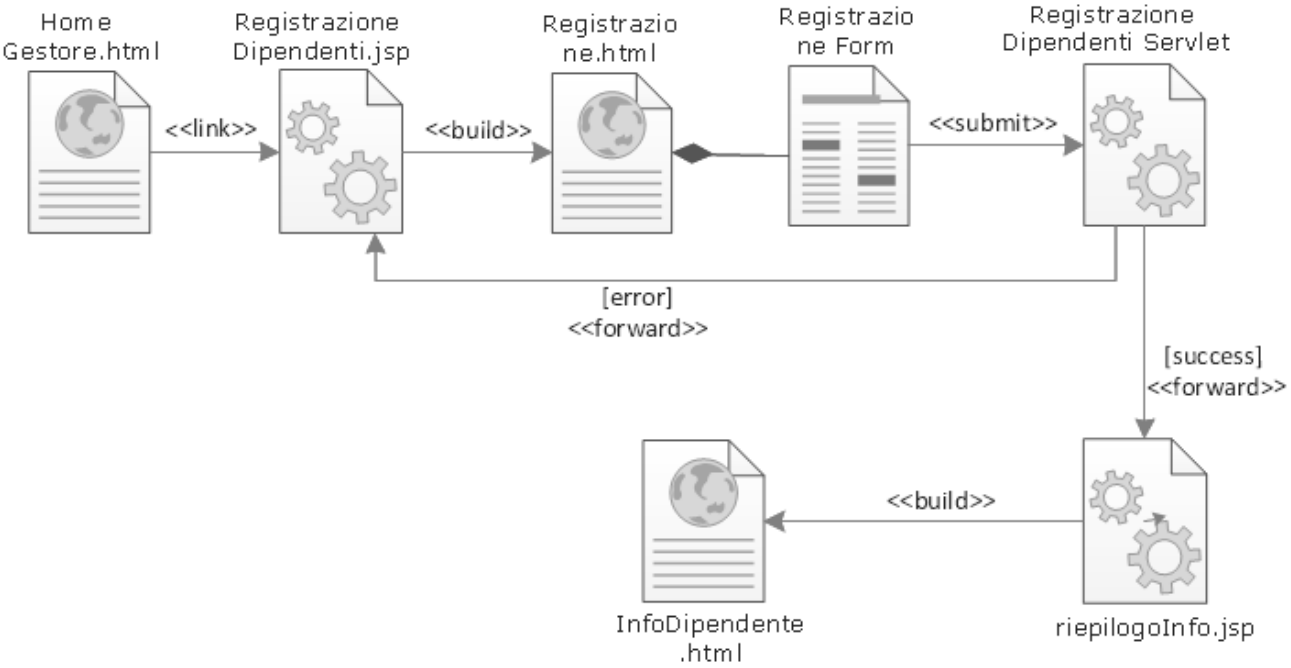
Addetto invia notifica



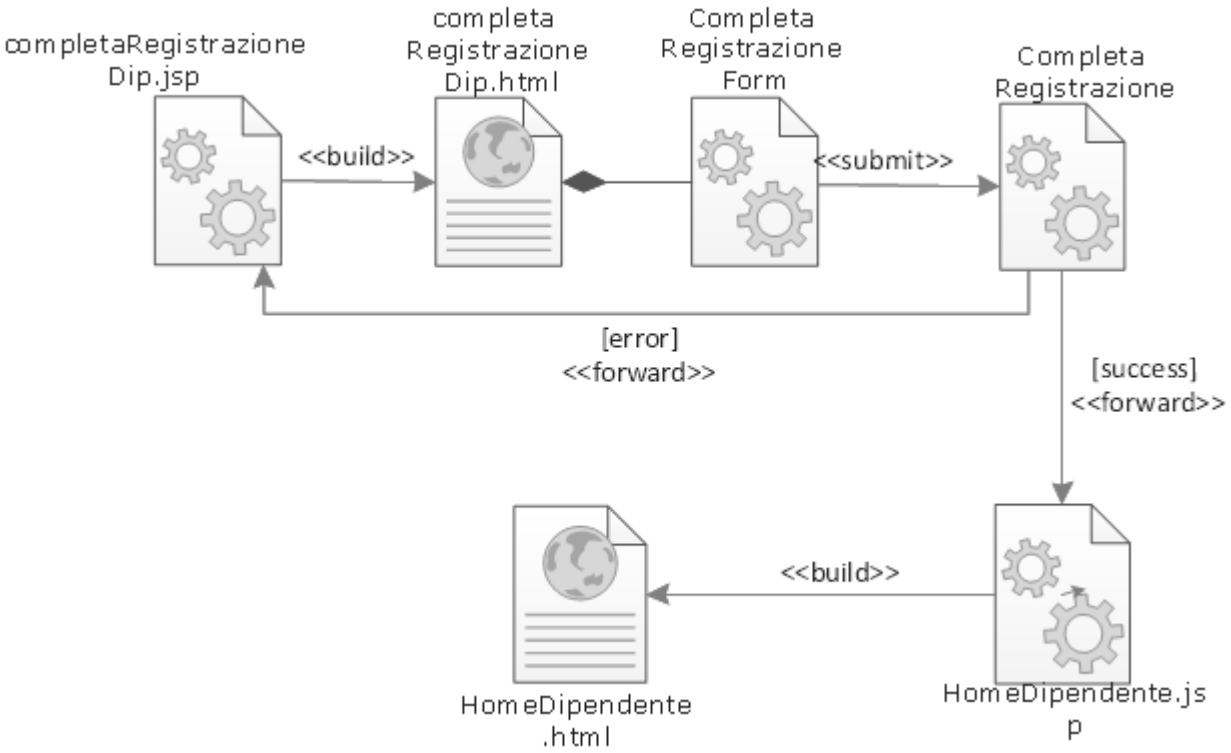
Addetto visualizza dettagli di un cliente prima di inviare la notifica



Registrazione di un dipendente



Conferma registrazione



5. CLASS INTERFACES

5.1. Cliente Model

ClienteModel	
doSave	**Context:** ClienteModel: doSave(cliente) **Pre:** cliente !=null && cliente.nome !=null && cliente.cognome !=null && clienteluogoNascita !=null&& cliente.datanascita !=null&&cliente.cf !=null && cliente.indirizzo != null &&clienti-> forAll(c| c.username!= cliente.username); **Post:** clienti->include(cliente)
cercaByCliente	**Context:** ClienteModel: cercaByCliente(clienteCF) **Pre:** clienteCF !=null && clienti->exists(cl| cl.CF==clienteCF) **Post:** cliente.CF=clienteCF
cercaPerNomeECognome	**Context:** ClienteModel: cercaPerNomeECognome(nome, cognome) **Pre:** nome != null && cognome!= null && clienti->exists(cl| cl.nome==nome && cl.cognome=cognome) **Post:** cliente.nome==nome && cliente.cognome==cognome
cercaNoPostepay	**Context:** ClienteModel: cercaNoPostepay() **Post:** cliente ->reject(conto->exists(co| co.Cf==cliente.CF&& PostePay->exists(pp| pp.IBAN=co.IBAN)))
cercaNoBancoposta	**Context:** ClienteModel: cercaNoBancoposta() **Post:** cliente ->reject(conto->exists(co| co.Cf==cliente.CF&& BancoPosta->exists(bp| bp.IBAN=co.IBAN)))
cercaNoServiziInternet	**Context:** ClienteModel: cercaNoServiziInternet() **Post:** cliente ->select(conto->exists(co| co.Cf==cliente.CF&& BancoPosta->exists(bp| bp.IBAN=co.IBAN && bp.ServiziInternet==’false’)))
cercaNoCarta	**Context:** ClienteModel: cercaNoCarta() **Post:** cliente ->select(conto->exists(co| co.Cf==cliente.CF&& BancoPosta->exists(bp| bp.IBAN=co.IBAN && bp.Carta==’false’)))

5.2. Conto Model

ContoModel	
cercaSaldo	**Context:** ContoModel: cercaSaldo(iban) **Pre:** iban != null && conto->exists(co| co.IBAN=iban) ; **Post:saldo:** conto->exists(co| co.IBAN=iban && co.saldo=saldo)
cercaConto	**Context:** ContoModel: cercaConto(iban) **Pre:** iban != null && conto->exists(co| co.IBAN=iban); **Post:** conto->exists(co| co.IBAN=iban)
creaConto	**Context:** ContoModel: creaConto(CF) **Pre:** CF!= null && conto->exists(co| co.CF=CF); **Post: iban:** conto->exists(co| co.IBAN=iban && co.CF==CF)
cercaSaldoPerCf	**Context:** ContoModel: cercaPerCf(CF) **Pre:** CF!= null && conto->exists(co| co.CF==CF); **Post:** saldo: conto->exists(co| co.CF==CF && co.saldo==saldo);

5.3. BancoPostaModel

BancoPostaModel	
cercaPerCf	**Context:** BancoPostaModel: cercaPerCf(CF) **Pre:** CF != null && conto->exists(co| co.CF==CF && BancoPosta->exists(bp| bp.IBAN=co.IBAN)); **Post:** BancoPosta->select(conto->exists(co| co.IBAN==bancoPosta.IBAN && co.CF==CF))
cercaByIban	**Context:** BancoPostaModel: cercaByIban(iban) **Pre:** iban !=null && BancoPosta->exists(bp| bp.IBAN=iban)); **Post**: bancoPosta: bancoPosta.IBAN==iban
creaBancoPosta	**Context:** BancoPostaModel: creaBancoPosta(tasso, servizi internet, costo, carta, iban) **Pre:** tasso != null&& Serviziinternet != null && costo != null && Carta != null && iban!= null && !BancoPosta->exists(bp| bp.IBAN=iban));

	Post: BancoPosta->exists(bp| bp.IBAN=iban));
siServiziInternet	**Context:** BancoPostaModel: siServiziInternet(iban) **Pre:** iban!= null && BancoPosta->exists(bp| bp.IBAN=iban)); **Post:** BancoPosta->exists(bp| bp.IBAN=iban && bp.ServiziInternet==’y’));
siCarta	**Context:** BancoPostaModel: siCarta(iban) **Pre:** iban != null && BancoPosta->exists(bp| bp.IBAN=iban)); **Post:** BancoPosta->exists(bp| bp.IBAN=iban && bp.Carta==’y’));
aggiornaCosto	**Context:** BancoPostaModel: aggiornaCosto(costo, iban) **Pre:** iban != null && costo != null && BancoPosta->exists(bp| bp.IBAN=iban)); **Post:** BancoPosta->exists(bp| bp.IBAN=iban && bp.costo==costo));

5.4. Dipendenti Model

DipendentiModel	
cercaByCF	**Context:** DipendentiModel: cercaByCF(CF) **Pre:** CF !=null && Dipendenti->exists(dp| dp.CF==CF); **Post:** dipendente: dipendente.CF==CF;
cercaByMatricola	**Context:** DipendentiModel: cercaByMatricola(matricola) **Pre:** Dipendenti->exists(dp| dp.matricola==matricola); **Post:** dipendente: dipendente.matricola==matricola;
inserisciDipendente	**Context:** DipendentiModel: inserisciDipendente(dipendente) **Pre: !**Dipendenti->include(dipendente); **Post:** Dipendenti->include(dipendente);

5.5. Notifiche Model

NotificheModel	
cercaByCF	**Context:** NotificheModel: cercaByCF(CF) **Pre:** CF!=null; **Post:** Notifiche->select(n| n.CF==CF);
inserisciNotifica	**Context:** NotificheModel: inserisciNotifiche(cliente, matricola, tipo, iban, codPosta) **Pre:** cliente!=null && tipo!=null, iban && Dipendenti->exists(dp| dp.matricola==matricola); **Post:** Notifiche-> exists(n|n.iban==iban && n.cliente==cliente && n.tipo==tipo && n.matricola==matricola && n.codPosta==codPosta);
cancellaNotifica	**Context:** NotificheModel: cancellaNotifica(codice) **Pre:** Notifiche-> exists(n|n.codice==codice); **Post:** !Notifiche-> exists(n|n.codice==codice);

5.6. Operazioni Model

OperazioniModel	
CercaOperazioni	**Context:** OperazioniModel: CercaOperazioni(iban) **Pre:** iban != null && conto->exists(co| co.IBAN=iban) **Post:** operazione->select(op| interessatoDa->exists(in| in.iban=iban && in.codice=op.codice));
Operazione	**Context:** OperazioniModel: Operazione(ibanFrom, ibanTo, importo) **Pre:** ibanFrom != null && ibanTo != null && conto->exists(co| co.IBAN=ibanFrom) && conto->exists(co| co.IBAN=ibanTo) && importo > 0; **Post:** operazioi→exists(op| op. importo==importo && interessatoDa→exists(int| int.codice==op.codice && int.iban== ibanFrom) && interessatoDa→exists(int| int.codice==op.codice && int.iban== ibanTo))

5.7. Pacchi Model

PacchiModel	
cercaPacco	**Context:** PacchiModel: cercaPacco(codice) **Pre:** codice != null && posta → exist (pa| pa.codice==codice); **Post:** pacco → select (pa| pa.codice== codice);
InserisciInPa	**Context:** PacchiModel: InserisciInPa(pacco) **Pre:** pacco != null && dataSpadizione != null && tipo != null && destinatario != null && mittente != null && dataConsegna != null && !(Posta → exist (po| po.codice == pacco.codice) && !(pacchi→ exist(pa|pa.codice == pacco.codice); **Post:** Posta->exists(po| po.codice==codice) && Pacchi → exist(pa|pa.codice==codice);

5.8. Posta Model

PostaModel	
aggiornaConsegna	**Context:** PostaModel: aggiornaConsegna(data, codice, dipendente) **Pre:** data != null && codice != null && dipendente != null && dipendente .nome != null && dipendente .cognome != null && dipendente .matricola != null && dipendente .tipo != null && Dipendenti->exists(dp| dp.matricola==matricola) && posta → exist(po | po.codice == codice); **Post:** posta → exist(po | po.codice == codice && po.dipendente=dipendente.matricola) && notifica->exists(n| n.matricola=matricola && n.codPosta=codice && n.cf=(posta-> select(po | po.codice == codice)).cf)
cercaByCodice	**Context:** PostaModel: cercaByCodice(codice) **Pre:** codice != null; **Post:** Posta → select (po| po.codice == codice);
cercaPerCfMit	**Context:** PostaModel: cercaPerCfMit(CF) **Pre:** CF != null && posta → exist(po | po.Cf == CF); **Post:** posta->select(po| po.Cf == CF);
inserisciInPo	**Context:** PostaModel: InserisciInPo(PostaDaInserire) **Pre:** PostaDaInserire != null && DataSpedizione != null && tipo != null && mittente != null && destinatario != null && dataConsegna != null && !(Posta → exist(po| po.codice !=

	postaDaInserire.codice)); **Post:** Posta->exists(po| po.codice=codice));

5.9. PostePayModel

PostePayModel	
cercaPerCf	**Context:** PostePayModel: cercaPerCf(CF) **Pre:** Cf != null PostePay->exists(pp| conti->exists(co| co.IBAN == pp.IBAN && co.CF=CF)); **Post:** PostePay->select(pp| conti->exists(co| co.IBAN == pp.IBAN && co.CF=CF));
cercaByIban	**Context:** PostePayModel: cercaByIBan(iban) **Pre:** iban != null PostePay->exists(pp| pp.IBAN=IBaN); **Post:** PostePay → select (pp| pp.iban == iban);
creaPostePay	**Context:** PostePayModel: creaPostePay(iban, codice) **Pre:** iban != null && codice != null && !(Postepay → exist(pp| pp.codice == codice) && !(Postepay → exist(pp| pp.iban == iban); **Post:** PostePay->exists(pp| pp.IBAN=iban && pp.codice=codice));

5.10. Utente Model

UtenteModel	
doSave	**Context:** UtenteModel: doSave(utente) **Pre:** utente != null && cliente != null && username != null && password != null && tipo != null && !(utentiRegistrati → exist (ut| ut.username == username)); **Post:** utentiRegistrati->exists(ut| ut.username==username));
esisteUser	**Context:** UtenteModel: esisteUser(user, password) **Pre:** user != null && password != null && (utentiRegistrati → exist (ut| ut.username == username); **Post:** utentiRegistrato→ select (ut| ut.username == username) && utentiRegistrato→ select (ut| ut.password == password);
doRetrieveByUser	**Context:** UtenteModel: doRetrieveByUser(username) **Pre:** username != null && (utentiRegistrati → exist (ut|

	ut.username == username); **Post:** utentiRegistrato→ select (ut| ut.username == username);
esisteCf	**Context:** UtenteModel: esisteCf(Cf) **Pre:** Cf!= null; **Post:** UtentiRegistrato → exist(ut| ut.cliente == Cf);
setPasswordDip	**Context:** UtenteModel: setPasswordDip(user, password); **Pre:** user!= null && password!=null && utentiRegistrati → exist (ut| ut.username == user); **Post:** utentiRegistrati → exist (ut| ut.username == user && ut.password= password);

5.11. Servlet conti

AttivaServizi	**Pre:** carta!=null && ServiziInternet!=null && iban!=null && BancoPosta->exists(bp| bp.IBAN=iban)); **Post:** BancoPosta->exists(bp| bp.IBAN=iban && bp.ServiziInternet==ServiziInternet && bp.carta==carta));
NoBancoPostaServlet	**Pre**: cf!=null && matricola!=null && clienti->exists(cl| cl.CF==clienteCF) && Dipendenti->exists(dp| dp.matricola==matricola); **Post:** Notifiche-> exists(n| n.clien==CF && n.tipo=”noBacnoPosta” && n.matricola==matricola);
NoCartaServlet	**Pre**: cf!=null && matricola!=null && iban!=null && clienti->exists(cl| cl.CF==clienteCF) && Dipendenti->exists(dp| dp.matricola==matricola) && BancoPosta->exists(bp| bp.IBAN=iban)); **Post:** Notifiche-> exists(n| n.clien==CF && n.tipo=”noCarta” && n.matricola==matricola && n.iban=iban);
NoPostePayServlet	**Pre**: cf!=null && matricola!=null && clienti->exists(cl| cl.CF==clienteCF) && Dipendenti->exists(dp| dp.matricola==matricola); **Post:** Notifiche-> exists(n| n.clien==CF && n.tipo=”noPostePay”

	&& n.matricola==matricola);
NoServiziInternetServlet	**Pre**: cf!=null && matricola!=null && iban!=null && clienti->exists(cl| cl.CF==clienteCF) && Dipendenti->exists(dp| dp.matricola==matricola) && BancoPosta->exists(bp| bp.IBAN=iban)); **Post:** Notifiche-> exists(n| n.clien==CF && n.tipo="noServiziInternet" && n.matricola==matricola && n.iban=iban);
NuovaPostePayServlet	Pre: codice!=null && cf!= null && clienti->exists(cl| cl.CF==clienteCF); **Post:** PostePay->exists(pp| pp.codice=codice && conti->exixsts(co|co.CF==CF)));
NuovoBancoPostaServlet	**Pre:** carta!= null && serviziInternet!= null && && cf!= null && clienti->exists(cl| cl.CF==clienteCF); **Post:** BancoPosta->exists(bp| bp.carta=carta && bp.SErviziInternet= serviziInternet && conti->exixsts(co|co.CF==CF));

5.12. Servlet gestioneUtenti

ComlpetaRegistrazione	**Pre:** username!= null && password!=null && utenteRegistrato->exists(ut| ut.user==username) **Post:** utenteRegistrato->exists(ut| ut.user==username && password=pasword)
LoginServlet	**Pre:** Username!=null && Password!=null && !utenteRegistrato->exists(ut| ut.user==username);
LogoutServlet	
RegistratiServel	**Pre:** nome!=null && cognome!=null && cf!=null && indirizzo!=null && luogoNascita!=null && dataNascita!=null && username!=null && password!= null && !utenteRegistrato->exists(ut| ut.user==username) && !clienti->exists(cl| cl.CF==clienteCF); **Post:** utenteRegistrato->exists(ut| ut.user==username) && clienti->exists(cl| cl.CF==clienteCF);

RegistrazioneDipendenteServlet	**Pre:** nome!=null && cognome!=null && cf!=null && indirizzo!=null && luogoNascita!=null && dataNascita!=null && matricola!=null && email!= null & ! dipendente→ exists(di| di.matricola=matricola) !utenteRegistrato->exists(ut| ut.user==username) && !clienti->exists(cl| cl.CF==clienteCF); **Post:** utenteRegistrato->exists(ut| ut.user==username) && dipendenrte->exists(di| di.matricola==matricola);

5.13. Servlet notifica

EliminaNotifica	**Pre:** codice!=null && Notifiche-> exists(n|n.codice==codice); **Post:** !Notifiche-> exists(n|n.codice==codice);

5.14. Servlet operazioni

BonificoServlet	**Pre**: ibanFRom!=null && ibanTo!=null && importo!=null && conto->exists(co| co.IBAN=ibanFrom) && conto->exists(co| co.IBAN=ibanTo) **Post**: operazioi→exists(op| op. importo==importo && interessatoDa→exists(int| int.codice==op.codice && int.iban== ibanFrom) && interessatoDa→exists(int| int.codice==op.codice && int.iban== ibanTo))
GirocontoServlet	**Pre**: ibanFRom!=null && ibanTo!=null && importo!=null && conto->exists(co| co.IBAN=ibanFrom) && conto->exists(co| co.IBAN=ibanTo); **Post**: operazioi→exists(op| op. importo==importo && interessatoDa→exists(int| int.codice==op.codice && int.iban== ibanFrom) && interessatoDa→exists(int| int.codice==op.codice && int.iban== ibanTo))

5.15. Servlet posta

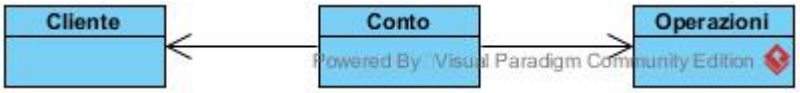
ConsegnaServlet	**Pre**: matricola!=null && codice!= nunll && Dipendenti->exists(dp| dp.matricola==matricola) && posta → exist(po | po.codice == codice); **Post**: **:** posta → exist(po | po.codice == codice && po.dipendente=dipendente.matricola) && notifica->exists(n| n.matricola=matricola && n.codPosta=codice && n.cf=(posta-> select(po | po.codice == codice)).cf)

EtichettaServlet	**Pre:** cliente!=null && codice!= null && posta → exist(po | po.codice == codice);
SpedisciServlet	**Pre:** cf!=null && clienti->exists(cl| cl.CF==clienteCF) && destinatario!= null && indirizzo!=null && tipo!=null; **Post:** Posta->exists(po| po.codice=codice && po.destinatario=destinatario && po.indirizzo=indirizzo));
VisualizzaPostaServlet	**Pre:** codice : posta→exists (po!po.codice=codice);

5.16. Servlet ajax

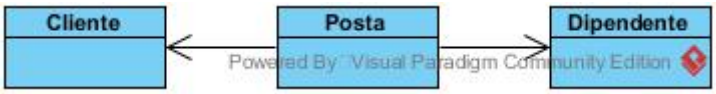
Iban servlet	**Pre**: codicefiscale != null && iban != null;
SaldoServlet	**Pre:** iban != null;

5.17. Mapping Models



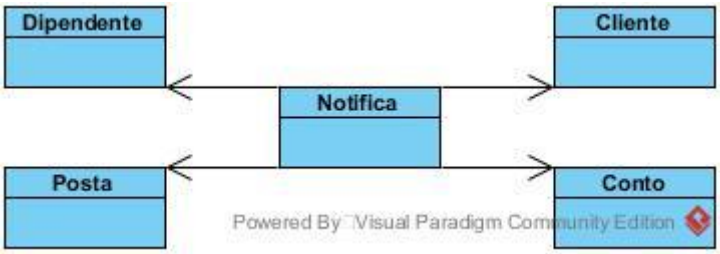
Per gestire questo mapping abbiamo inserito all'interno dell'oggetto "Conto" due attributi quali:

- String Cliente.CF;
- ArrayList<Operazioni> Op;



Per gestire questo mapping abbiamo inserito all'interno dell'oggetto "Posta" due attributi quali:

- String Cliente.CF;
- String Dipendente.CF;



Per gestire questo mapping abbiamo inserito all'interno dell'oggetto "Notifica" quattro attributi quali:

- String Dipendente.CF;
- String Posta.Codice;
- String Cliente.CF;
- String Conto.Iban;

6. 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 utente che ha effettuato la registrazione sul sistema

Web-Based: Il termine identifica un sistema basato sul web, quindi accessibile simultaneamente da più postazioni