Object Design Document

Just Traditions

Riferimento	2022_ODD_C08
Versione	1.0
Data	12/02/2023
Destinatario	Top management
Presentato da	G. Sgambato, L. Sorrentino, M. Spagnuolo, D. Avella, G. Miele
Approvato da	F. Cirillo, G. Brescia

Revision History

Data	Versione	Descrizione	Autori
22/12/2022	0.1	Prima stesura	GM
27/12/2022	0.2	Introduzione	GM, LS
05/01/2023	0.3	Aggiunta Componenti Off-The-Shelf	GM, DA
15/01/2023	0.4	Aggiunta Packages	LS,GS,MS
25/01/2023	0.5	Aggiunta Class Interfaces	GS
07/02/2023	0.6	Aggiunta Class Diagram	GM
10/02/2023	0.7	Revisione Class Diagram	TUTTI
12/02/2023	1.0	Revisione finale	TUTTI

Team Members

Nome	Ruolo	Acronimo	Informazioni di contatto
Gerardo Brescia	Project Manager	GB	g.brescia3@studenti.unisa.it
Franco Cirillo	Project Manager	FC	f.cirillo30@studenti.unisa.it
Giovanni Miele	Team Member	GM	g.miele18@studenti.unisa.it
Michele Spagnuolo	Team Member	MS	m.spagnuolo26@studenti.unisa.it
Lorenzo Sorrentino	Team Member	LS	l.sorrentino66@studenti.unisa.it
Domenico Avella	Team Member	DA	d.avella11@studenti.unisa.it
Giuseppe Sgambato	Team Member	GS	g.sgambato1@studenti.unisa.it

Sommario

1 Introduzione **5**

1.1 Object design trade-offs 5

1.1.1 Componenti off-the-shelf 5

1.2 Linee guida per la documentazione dell'interfaccia 7

1.3 Definizioni, acronimi e abbreviazioni 7

1.4 Riferimenti 7

2 Packages **7**

3 Class Interfaces **8**

4 Class Diagram **8**

5 Design Patterns **8**

6 Glossario 40

1 Introduzione

Just Traditions si propone di semplificare l'interazione tra artigiani e clienti.

Al fine di aumentare il bacino di utenza di queste piccole realtà, si propone di creare uno strumento che si interponga tra le due parti, semplificando i processi di scoperta di nuove attività e di prenotazione di una visita.

In questa prima sezione del documento verranno descritti i trade-offs individuati e componenti off-the-shelf usati.

1.1 Object design trade-offs

Durante la fase di analisi e di progettazione sono stati individuati diversi compromessi per lo sviluppo del sistema. Inoltre, anche nella fase dell'Object Design sorgono diversi trade-offs di progettazione analizzati nel corso di questa sezione del documento:

Readability vs. Release Time

Il tempo di rilascio della piattaforma è uno degli obiettivi prioritari che sono stati individuati. A tal proposito non verrà commentata ogni singola riga del codice con descrizioni dettagliate, nonostante ciò, ci si propone di rendere il codice comprensibile e manutenibile.

Buy vs. Build

L'idea è quella di riutilizzare il più possibile soluzioni off-the-shelf, scegliendo in modo oculato al fine di trarne il maggior numero di vantaggi possibili. Ci si propone, quindi, di utilizzare soluzioni precostruite data la varietà di tools per lo sviluppo di applicazioni web. La scelta sarà effettuata basandosi sulle conoscenze dei membri del team evitando così che il tempo di apprendimento della nuova tecnologia risulti svantaggioso rispetto alla sua effettiva utilità.

1.1.1 Componenti off-the-shelf

Per facilitare e velocizzare lo sviluppo della piattaforma Just Traditions, saranno utilizzate diverse componenti off-the-shelf di seguito riportate.

jQuery e AJAX

Per permettere all'interfaccia grafica di rispondere alle azioni dell'utente in modo rapido ed efficiente e per migliorare la user experience durante l'interazione dell'utente con la piattaforma verranno utilizzati jQuery e AJAX.

Bootstrap

Per permettere lo sviluppo di un'interfaccia grafica che sia responsive ed esteticamente piacevole, per la maggior parte degli utenti, sarà utilizzato il framework Bootstrap.

MySql-JDBC

MySQL Connector/J (JDBC) è il driver che permette l'interoperabilità fra Java e MySQL. Viene impiegato per permettere alla WebApp di interfacciarsi col Database, e dunque compiere operazioni CRUD.

Spring Data JPA

Basato su tecnologia ORM (Object-Relational Mapping), ci permette di interrogare e gestire le tabelle di un database senza dover necessariamente scrivere query, ma utilizzando classi, oggetti e metodi. Inoltre, in caso di cambiamento di dbms, non avendo scritto query, non bisogna preoccuparsi di adattarle al nuovo DBMS, farà tutto il framework.

String Data JPA ci servirà per:

- Generare in automatico il codice per la creazione delle tabelle
- Implementare le operazioni crud
- implementare query complesse
- Gestire le relazioni tra tabelle

Spring Web MVC

Framework web di Spring basato su Servlet-API di Java. Nonostante il nome possa essere fuorviante, non sarà utilizzata l'architettura MVC, come illustrato nel documento di System Design.

Spring Boot

Spring Boot è una soluzione "convention over configuration" per il framework Spring di Java, che riduce la complessità di configurazione di nuovi progetti Spring.

1.2 Linee guida per la documentazione dell'interfaccia

Le linee guida includono una lista di regole che gli sviluppatori dovrebbero rispettare durante la progettazione delle interfacce.

Per tale scopo, utilizzeremo il plugin "CheckStyle-IDEA", uno strumento di sviluppo per aiutare i programmatori a scrivere codice aderendo a uno standard di codifica. Automatizza il processo di controllo del codice applicando un file di configurazione che supporta le convenzioni di ''Google Checks''.

Di seguito una lista di link alle convenzioni usate per definire le linee guida:

- **Java**: Link alla convenzione
- **HTML/CSS**: Link alla convenzione
- **JS:** Link alla convenzione

1.3 Definizioni, acronimi e abbreviazioni

- **DP**: Design Pattern

1.4 Riferimenti

Di seguito una lista di riferimenti ad altri documenti utili durante la lettura:

- 2022_SOW_C08
- 2022_RAD_C08
- 2022_SDD_C08
- 2022_ODD_C08
- 2022_TP_C08
- 2022_TCS_C08
- 2022_MatriceDiTracciabilità_C08
- Javadoc di JustTraditions
- 2022_ManualeUtente_C08
- 2022_ManualediInstallazione_C08

2 Packages

Struttura del Progetto

La struttura del progetto ricalca la struttura di directory standard definita da Maven e Spring.

- **src**, contiene tutti i file sorgente
  - **main**
    - **java**, contiene le classi Java relative alle componenti Application Logic e Storage
    - **resources**, contiene i file relativi alla componente Interface
      - **static,** contiene i fogli di stile CSS e gli script JS
      - **templates,** contiene i file HTML reindirizzati da Thymeleaf
  - **test**, contiene tutto il necessario per il testing
    - **java**, contiene le classi Java per l'implementazione del testing
- **pom.xml**, contiene la configurazione del progetto Maven

Package Just Traditions

La divisione in package è molto legata alla suddivisione in sottosistemi effettuata nel documento di System Design; pertanto, nella divisione in package si evidenzia l'utilizzo dell'architettura Three-tier.

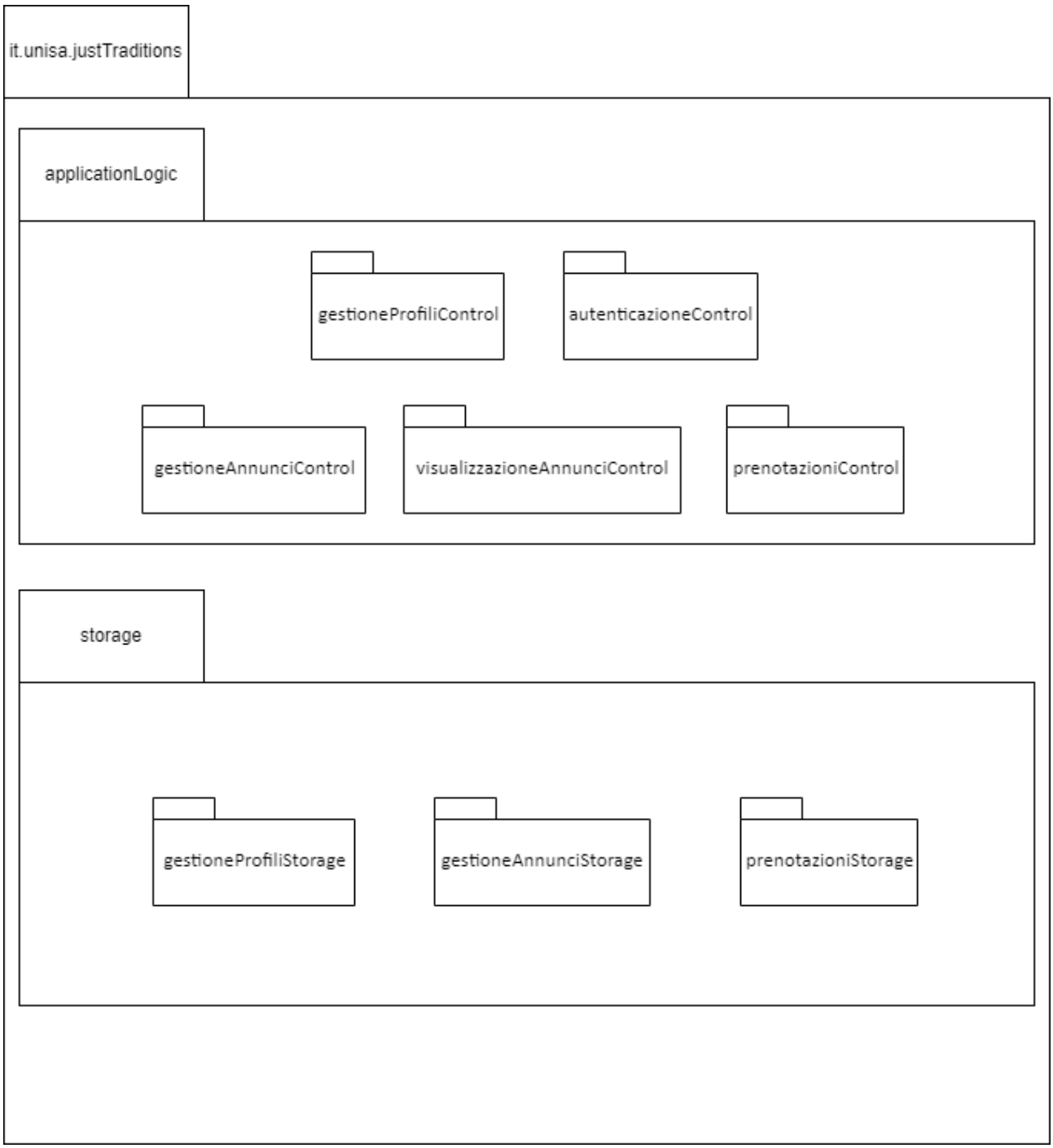
Nella presente sezione si mostra la struttura del package principale di JustTraditions. La struttura generale è stata ottenuta a partire da due principali scelte:

1. Creare dei package separati per i layer di Application Logic e Storage, contenenti a loro volta i package dei singoli sottosistemi.
2. Suddividere i sottopackage del package storage in package dao ed entity, in modo tale da suddividere la logica di accesso al database dalle entità che mappano le rispettive tabelle.

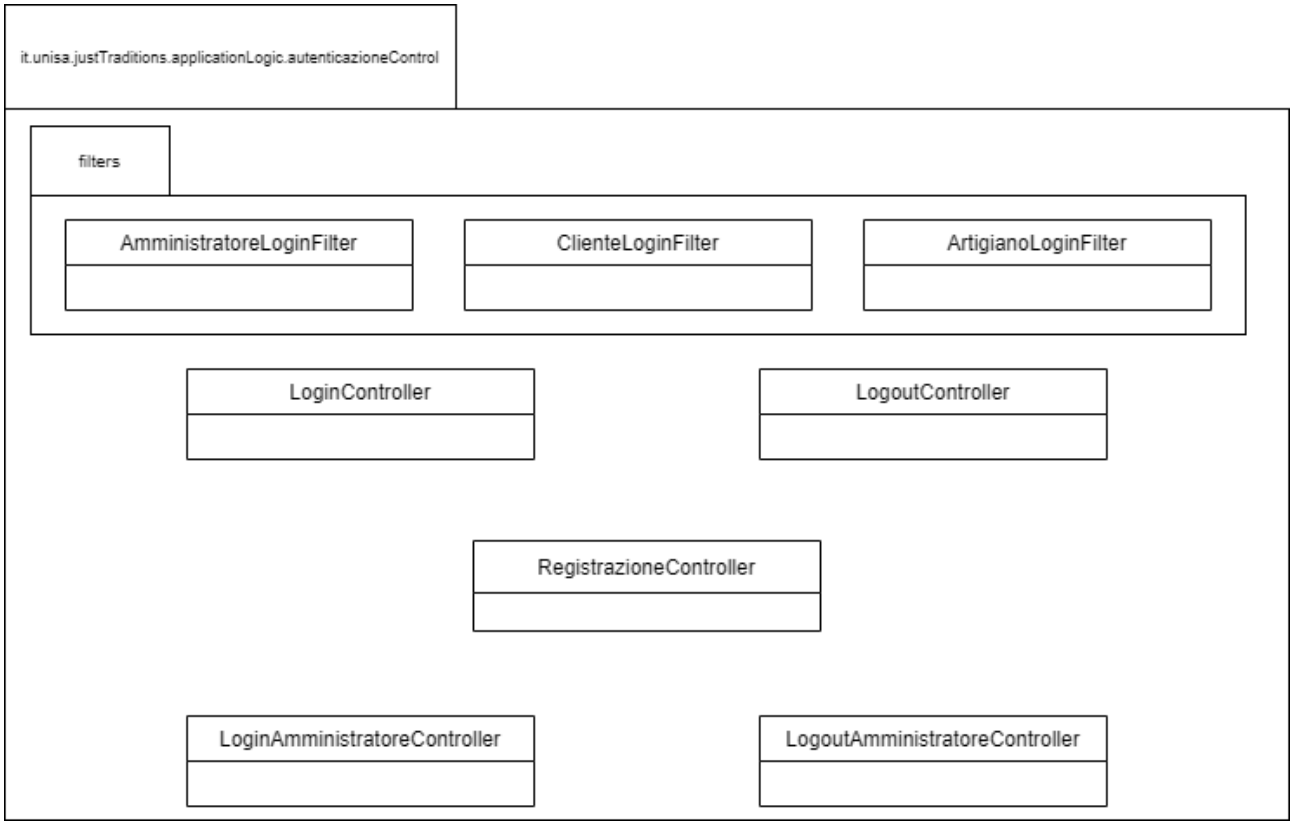
Per ciò che concerne le dipendenze tra i packages, esse corrispondono alle dipendenze indicate tra i sottosistemi nel SDD.

Alcune classi e packages di utility sono stati omessi nei seguenti schemi a scopo di leggibilità, perché non fondamentali per comprendere la struttura del progetto. Questi sono comunque visionabili nel JavaDoc.

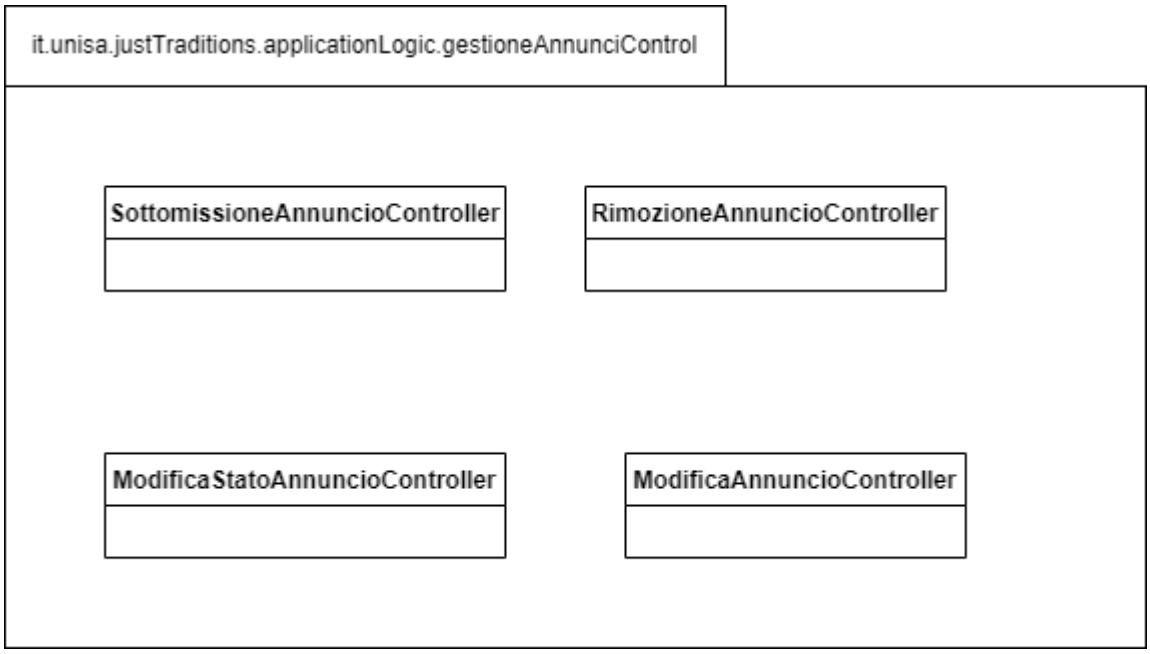
Package Application Logic



Package Autenticazione Control

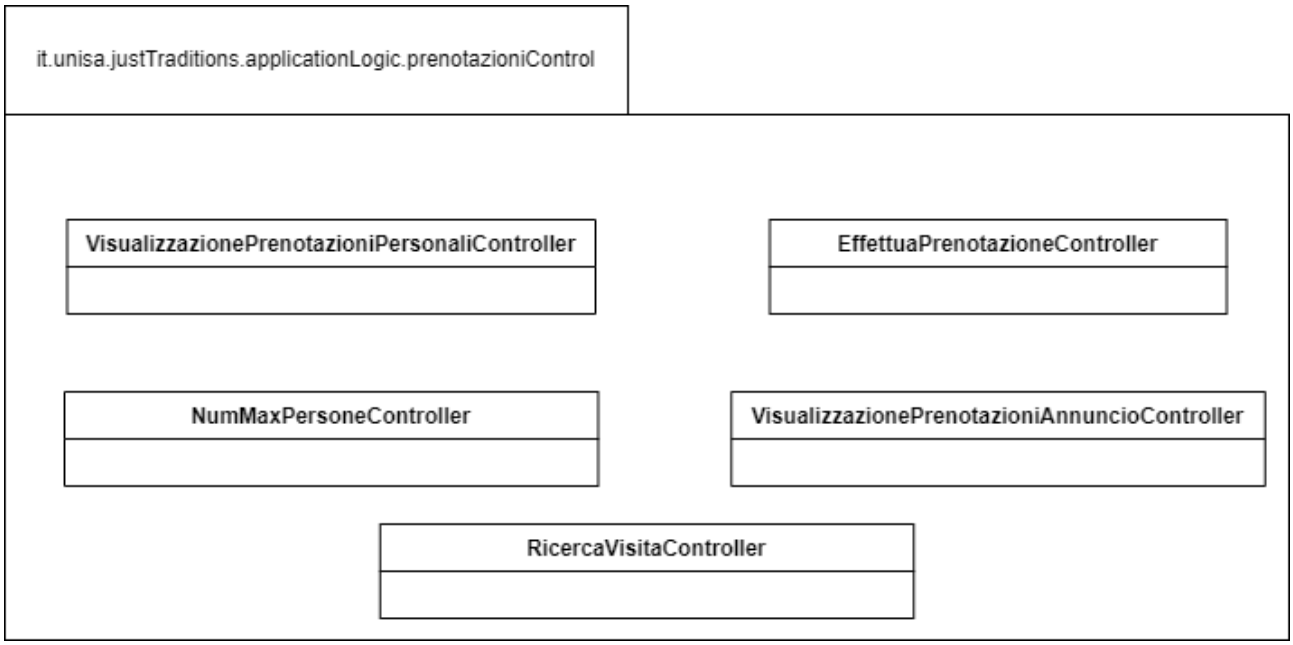


Package Gestione Annunci Control

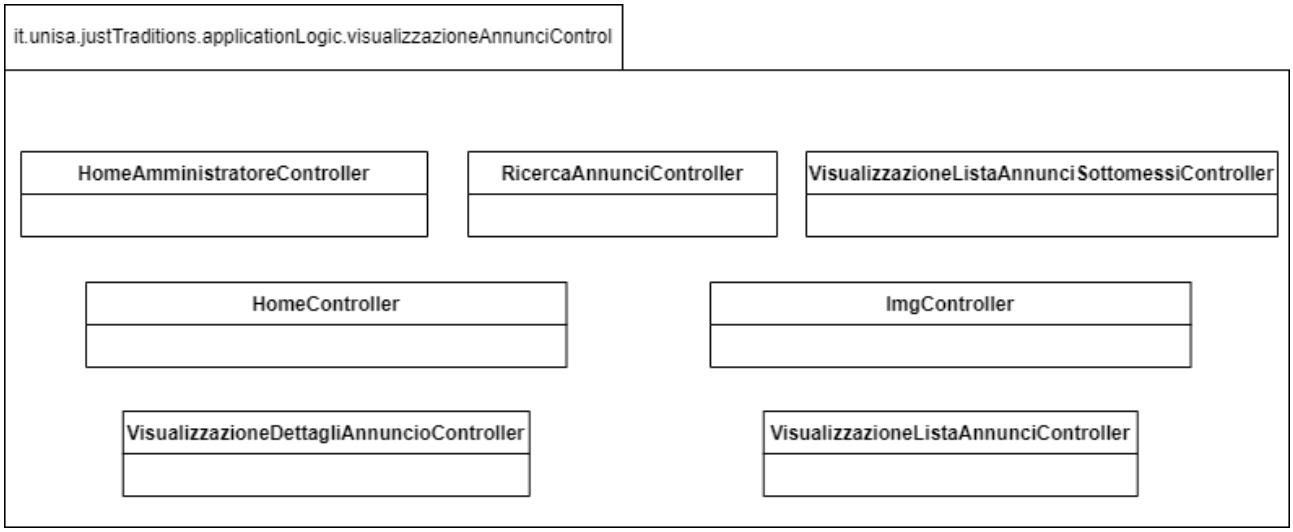


Package Gestione Profili Control

Package Prenotazioni Control



Package Visualizzazione Annunci Control



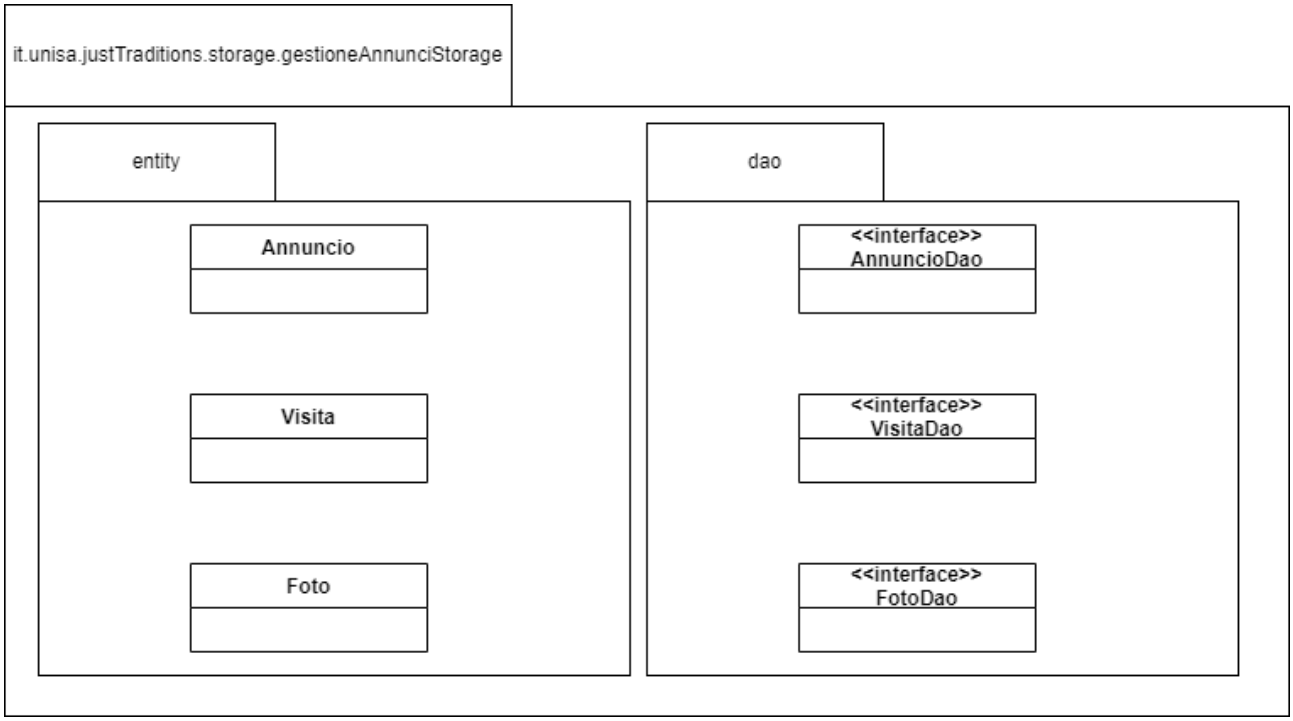
Le funzionalità di Ricerca Annunci e Visualizzazione Lista Annunci Approvati saranno gestite da un unico controller, data la loro affinità.

Le funzionalità di Visualizzazione Lista Annunci e Filtra Lista Annunci saranno gestite da un unico controller, data la loro affinità.

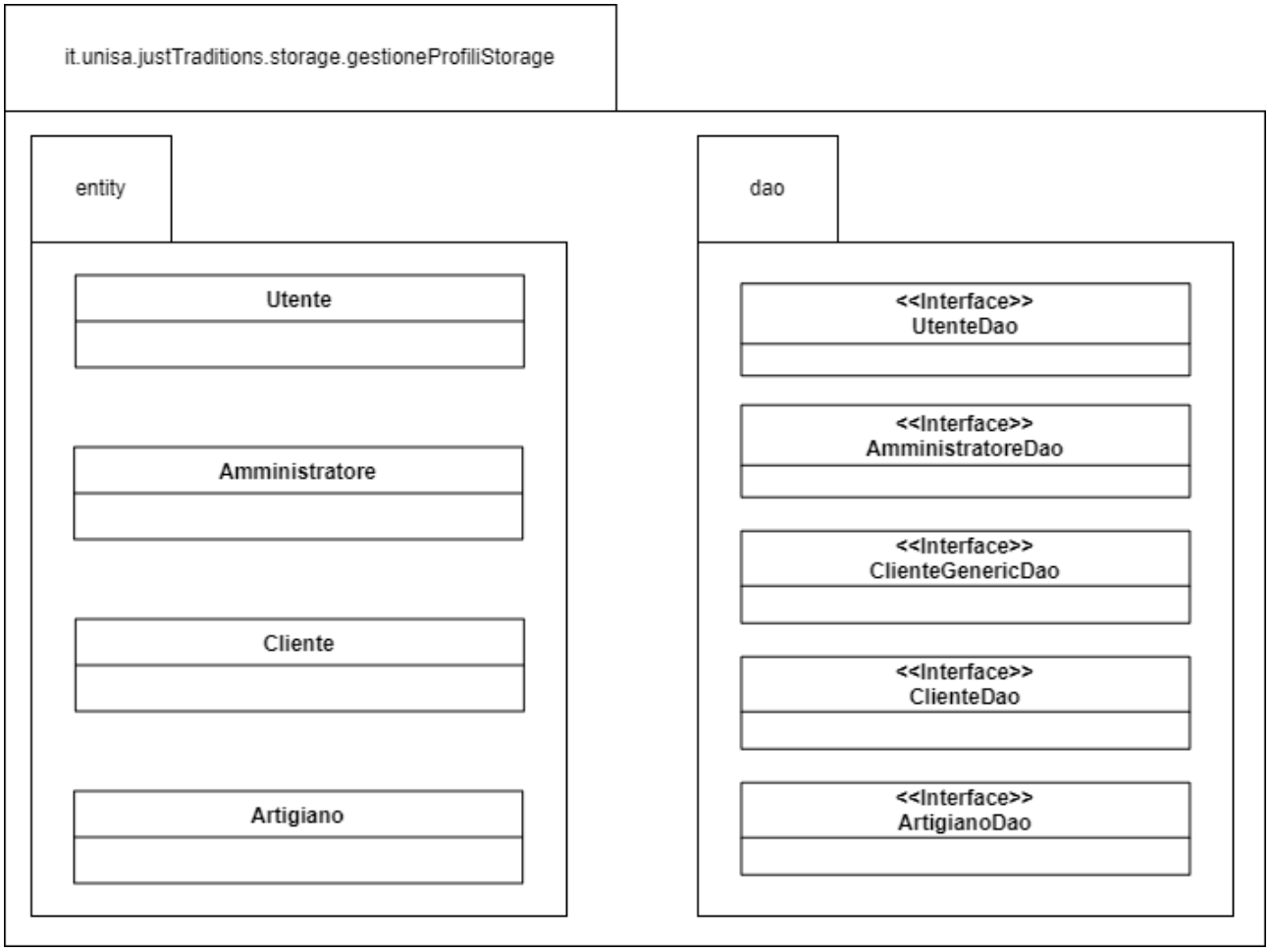
La funzionalità di Visualizzazione Scheda Annuncio non ha un controller dedicato, dato che è stata accorpata all'interno di ModificaStatoAnnuncioController, nel package gestioneAnnunciControl, dato il forte accoppiamento che sussiste fra le due funzionalità.

Package Storage

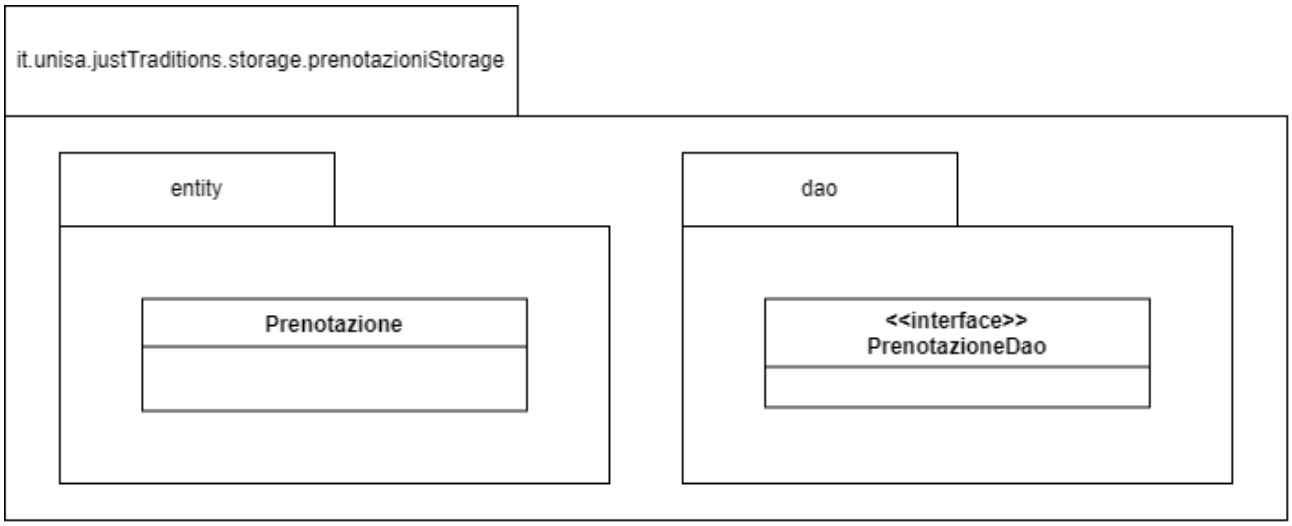
Package Gestione Annunci Storage



Package Gestione Profili Storage



Package Prenotazioni Storage



3 Class Interfaces

1 Package autenticazioneControl

Nome classe	LoginAmministratoreController
Descrizione	Implementa il controller per il login per gli Amministratori.
Metodi	+get(LoginForm loginForm, Model model):String +post(LoginForm loginForm, BindingResult bindingResult, Model model): String
Invariante di classe	/

Nome Metodo	+get(LoginForm loginForm, Model model):
Descrizione	Implementa la funzionalità di smistare il Cliente sulla view di autenticazioneView/login.
Pre-condizione	
Post-condizione	
Invariante di classe	/
Nome Metodo	+post(LoginForm loginForm, BindingResult bindingResult, Model model)
Descrizione	Implementa la funzionalità di login di un Cliente.
Pre-condizione	**context:** LoginAmministratoreController::post(loginForm, bindingResult, model) **pre:** not bindingResult.hasErrors()
Post-condizione	**context:** LoginAmministratoreController::post(loginForm, bindingResult, model) **post:** sessionAmministratore.getAmministrstore().isPresent()
Invariante di classe	/

Nome classe	LoginController
Descrizione	Implementa il controller per il login per del Cliente.
Metodi	+get(LoginForm loginForm, Model model):String +post(LoginForm loginForm, BindingResult bindingResult, Model model): String
Invariante di classe	/

Nome Metodo	+get(LoginForm loginForm, Model model):
Descrizione	Implementa la funzionalità di smistare il Cliente sulla view di autenticazioneView/login.
Pre-condizione	
Post-condizione	
Invariante di classe	/
Nome Metodo	+post(LoginForm loginForm, BindingResult bindingResult, Model model)
Descrizione	Implementa la funzionalità di login di un Cliente.
Pre-condizione	**context:** LoginController::post(loginForm, bindingResult, model) **pre:** not bindingResult.hasErrors()
Post-condizione	**context:** LoginController::post(loginForm, bindingResult, model) **post:** sessionCliente.getCliente().isPresent()
Invariante di classe	/

Nome classe	LogoutAmministratoreController
Descrizione	Implementa il controller per il logout per L'Amministratore.
Metodi	+get():String
Invariante di classe	/

Nome Metodo	+get():
Descrizione	Implementa la funzionalità di logout di un Amministratore.
Pre-condizione	
Post-condizione	**context**:LogoutAmministratoreController::get() **Post**:sessionAmministratore.getAmministratore().isEmpty()
Invariante di classe	/

Nome classe	LogoutController
Descrizione	Implementa il controller per il logout per il Cliente.
Metodi	+get():String
Invariante di classe	/

Nome Metodo	+get(LoginForm loginForm, Model model):
Descrizione	Implementa la funzionalità di logout di un Cliente.
Pre-condizione	
Post-condizione	**context**:LogoutController::get() **Post**:sessionCliente.getCliente().isEmpty()
Invariante di classe	/

Nome classe	RegistrazioneController
Descrizione	Implementa il controller per la registrazione del Cliente.
Metodi	+get(RegistrazioneForm RegistrazioneForm,):String +post(RegistrazioneForm registrazioneForm, BindingResult bindingResult,Model model): String
Invariante di classe	/

Nome Metodo	+get(RegistrazioneForm registrazioneForm)
Descrizione	Implementa la funzionalità di smistare il Cliente sulla view di autenticazioneView/registrazione.
Pre-condizione	
Post-condizione	
Invariante di classe	/
Nome Metodo	+post(RegistrazioneForm registrazioneForm, BindingResult bindingResult): String
Descrizione	Implementa la funzionalità di login di un Cliente.
Pre-condizione	**context:** RegistrazioneController::post(registrazioneForm, bindingResult) **pre:** not bindingResult.hasErrors()
Post-condizione	**context:** RegistrazioneController::post(registrazioneForm, bindingResult) **post:** clienteDao.save(cliente) == true.
Invariante di classe	/

2 Package gestioneAnnunciControl

Nome classe	ModificaAnnuncioController
Descrizione	Implementa il controller per la modifica di un annuncio.
Metodi	+get(Long id,Model model):String +post(AnnuncioForm annuncioForm, BindingResult bindingResult,idFoto, Model model): String
Invariante di classe	/

Nome Metodo	+get(Long id,Model model):String
Descrizione	Implementa la funzionalità di smistare l'Artigiano sulla view di gestioneAnnunciView/modificaAnnuncio.
Pre-condizione	**context:** ModificaAnnuncioController::get(id, model) **pre:** sessionCliente.getCliente().isPresent() **pre:** sessionCliente.getCliente().get().getClass() == Artigiano.class **pre:** annuncioDao.existsById(id)
Post-condizione	**context:** ModificaAnnuncioController::get(id, model) **post:** model.getAttribute("annuncioForm") != null **post:** model.getAttribute("idFoto") != null
Invariante di classe	/
Nome Metodo	+post(AnnuncioForm annuncioForm, BindingResult bindingResult,idFoto, Model model): String
Descrizione	Implementa la funzionalità di Modifica di un annuncio.
Pre-condizione	**context:** ModificaAnnuncioController::post(annuncioForm, bindingResult,idFoto, model) **pre:** sessionCliente.getCliente().isPresent()

	pre: sessionCliente.getCliente().get().getClass() == Artigiano.class **pre:** not bindingResult.hasErrors()
Post-condizione	**context:** ModificaAnnuncioController::post(annuncioForm, bindingResult,idFoto, model) **post:** annuncioDao.save(annuncio) == true
Invariante di classe	/

Nome classe	**ModificaStatoAnnuncioController**
Descrizione	Implementa il controller per la modifica di uno stato del Annuncio.
Metodi	+ get(Long idAnnuncio, ModificaStatoAnnuncioForm modificaStatoAnnuncioForm, Model model):String + post(ModificaStatoAnnuncioForm modificaStatoAnnuncioForm, BindingResult bindingResult, Model model): String
Invariante di classe	/

Nome Metodo	+ get(Long idAnnuncio, ModificaStatoAnnuncioForm modificaStatoAnnuncioForm, Model model):String
Descrizione	Implementa la funzionalità di smistare l'Amministratore sulla view di gestioneAnnunciView/modificaStatoAnnuncio.
Pre-condizione	**context:**ModificaStatoAnnuncioController:: get(idAnnuncio, modificaStatoAnnuncioForm, model) **pre:**sessionAmministratore.getAmministratore( ).isPresent(). **pre**: annuncioDao.existsById(idAnnuncio) **pre:** not bindingResult.hasErrors()

Post-condizione	**context:**ModificaStatoAnnuncioController:: get(idAnnuncio, modificaStatoAnnuncioForm, model) **pre:**pre: model.getAttribute("annuncio") != null
Invariante di classe	/
Nome Metodo	+ post(ModificaStatoAnnuncioForm modificaStatoAnnuncioForm, BindingResult bindingResult, Model model): String
Descrizione	Implementa la funzionalità di Modifica dello stato di un annuncio.
Pre-condizione	**context:**ModificaStatoAnnuncioController:: post(modificaStatoAnnuncioForm, bindingResult, model) **pre:**sessionAmministratore.getAmministratore( ).isPresent() **pre:** not bindingResult.hasErrors()
Post-condizione	**context:**ModificaStatoAnnuncioController:: post(modificaStatoAnnuncioForm, bindingResult, model) **post:** annuncioDao.save(annuncio) == true **post:** model.getAttribute("isAmministratore") != null
Invariante di classe	/

Nome classe	**RimozioneAnnuncioController**
Descrizione	Implementa il controller per la Rimozione di un annuncio.
Metodi	+ get(Long id):String
Invariante di classe	/

Nome Metodo	+ get(@RequestParam Long id)):String
Descrizione	Implementa la funzionalità di Rimozione di un annuncio.
Pre-condizione	**context:**RimozioneAnnuncioController::get(id) **pre:** sessionCliente.getCliente().isPresent() **pre:** sessionCliente.getCliente().get().getClass() == Artigiano.class **pre:**sessionCliente.getCliente().getAnnunci().content(annuncioDao.findByid(id))
Post-condizione	**context:**RimozioneAnnuncioController::get(id) **post:** sessionCliente.getCliente().getAnnunci().size() == @pre.sessionCliente.getCliente().getAnnunci().size() - 1
Invariante di classe	/

Nome classe	**SottomissioneAnnuncioController**
Descrizione	Implementa il controller per la sottomissione di un Annuncio.
Metodi	+ get(AnnuncioForm annuncioForm);String +post(AnnuncioForm annuncioForm, BindingResult bindingResult, Model model): String
Invariante di classe	/

Nome Metodo	+ get(AnnuncioForm annuncioForm):String
Descrizione	Implementa la funzionalità di smistare l'Artigiano sulla view di gestioneAnnunciView/sottomissioneAnnuncio.
Pre-condizione	**context:**SottomissioneAnnuncioController::get(annuncioForm) **pre:** sessionCliente.getCliente().isPresent() **pre:** sessionCliente.getCliente().get().getClass() == Artigiano.class **pre:** not bindingResult.hasErrors()

Post-condizione	
Invariante di classe	/
Nome Metodo	+post(AnnuncioForm annuncioForm, BindingResult bindingResult, Model model): String
Descrizione	Implementa la funzionalità di Sottomissione di un annuncio.
Pre-condizione	**context:**SottomissioneAnnuncioController::post (annuncioForm, bindingResult, model) **pre:** sessionCliente.getCliente().isPresent() **pre:** sessionCliente.getCliente().get().getClass() == Artigiano.class
Post-condizione	**context:**SottomissioneAnnuncioController::post (annuncioForm, bindingResult, model) **pre:**annuncioDao.save(annuncio)
Invariante di classe	/

3 Package gestioneProfiliControl

Nome classe	**AggiuntaAmministratoreController**
Descrizione	Implementa il controller per l'aggiunta di un amministratore.
Metodi	+ get(AggiuntaAmministratoreForm aggiuntaAmministratoreForm):String + post(AggiuntaAmministratoreForm aggiuntaAmministratoreForm, BindingResult bindingResult): String
Invariante di classe	/

Nome Metodo	+ get(AggiuntaAmministratoreForm aggiuntaAmministratoreForm):String
Descrizione	Implementa la funzionalità di smistare l'Amministratore sulla view di gestioneAnnunciView/modificaAnnuncio.
Pre-condizione	**context:**AggiuntaAmministratoreController::get(aggiuntaAmministratoreForm) **pre:** sessionAmministratore.getAmministratore().isPresent()
Post-condizione	**context:**AggiuntaAmministratoreController::get(aggiuntaAmministratoreForm) post:amministratoreDao.save(amministratore)
Invariante di classe	/
Nome Metodo	+ post(AggiuntaAmministratoreForm aggiuntaAmministratoreForm, BindingResult bindingResult): String
Descrizione	Implementa la funzionalità di aggiunta di un Amministratore.
Precondizione	**context:**AggiuntaAmministratoreController::post(aggiuntaAmministratoreForm, bindingResult) **pre:** sessionAmministratore.getAmministratore().isPresent() **pre:** not bindingResult.hasErrors()
Post-condizione	**context:**AggiuntaAmministratoreController::post(aggiuntaAmministratoreForm, bindingResult) amministratoreDao.save(amministratore) == true;
Invariante di classe	/

Nome classe	CancellazioneAccountController
Descrizione	Implementa il controller per la cancellazione di un account.
Metodi	+ get():String
Invariante di classe	/

Nome Metodo	+ get():String
Descrizione	Implementa la funzionalità di eliminare l'account di un Cliente.
Pre-condizione	**context:**CancellazioneAccountController:: get() **pre:** sessionCliente.getCliente().isPresent()
Post-condizione	**context:**CancellazioneAccountController:: get() **post:**clienteDao.delete(cliente) == true
Invariante di classe	/

Nome classe	ModificaProfiloController
Descrizione	Implementa il controller per la modifica di un profilo.
Metodi	+ get(RegistrazioneForm registrazioneForm):String + post(RegistrazioneForm registrazioneForm, BindingResult bindingResult): String
Invariante di classe	/

Nome Metodo	+ get(RegistrazioneForm registrazioneForm):String
Descrizione	Implementa la funzionalità di smistare il Cliente su la view di gestioneProfiliView/modificaProfilo.
Pre-condizione	**context:**ModificaProfiloController:: get(registrazioneForm) **pre:** sessionCliente.getCliente().isPresent()

Post-condizione	
Invariante di classe	/
Nome Metodo	+ post(RegistrazioneForm registrazioneForm, BindingResult bindingResult): String
Descrizione	Implementa la funzionalità di Modifica profilo di un Cliente.
Pre-condizione	**context:**ModificaProfiloController:: post(registrazioneForm, bindingResult) **pre:**sessionCliente.getCliente().isPresent() **pre:** not bindingResult.hasErrors()
Post-condizione	**context:**ModificaProfiloController:: post(registrazioneForm, bindingResult) **post:**sessionCliente.getCliente().isPresent() **post**:bindingResult.hasFieldErrors("nome") || bindingResult.hasFieldErrors("cognome") || bindingResult.hasFieldErrors("email") || bindingResult.hasFieldErrors("codiceFiscale") || bindingResult.hasFieldErrors("artigiano") || bindingResult.hasFieldErrors("iban") || bindingResult.hasGlobalErrors() **post:**clienteDao.save(cliente) == true
Invariante di classe	/

Nome classe	**RimozioneAmministratoreController**
Descrizione	Implementa il controller per la rimozione di un amministratore.
Metodi	+ get(Long id):String
Invariante di classe	/

Nome Metodo	+ get(Long id):String
Descrizione	Implementa la funzionalità di rimozione di un Amministratore.
Pre-condizione	**context**:RimozioneAmministratoreController::get(id) **pre:** sessionAmministratore.getAmministratore().isPresent() **pre:** amministratoreDao.existsById(id)
Post-condizione	**context**:RimozioneAmministratoreController::get(id) **post:**amministratoreDao.deleteById(id) == true
Invariante di classe	/

Nome classe	VisualizzazioneAmministratoriController
Descrizione	Implementa il controller per la visualizzazione di una lista di Amministratori.
Metodi	+ get():ModelAndView
Invariante di classe	/

Nome Metodo	+ get():ModelAndView
Descrizione	Implementa la funzionalità di smistare l Amministratore sulla view di gestioneProfiliView/visualizzazioneAmministratori.
Pre-condizione	**context:**VisualizzazioneAmministratoriController::get() **pre:** sessionAmministratore.getAmministratore().isPresent()
Post-condizione	**context:**VisualizzazioneAmministratoriController::get() **post:** result.getObject("Amministratori").equals(Amministratori->asList())

Invariante di classe	/

Nome classe	**VisualizzazioneProfiloArtigianoController**
Descrizione	Implementa il controller per la visualizzazione di un profilo di un artigiano.
Metodi	+ get(Long id, Integer pagina, Model model):String
Invariante di classe	/

Nome Metodo	+ get(Long id, Integer pagina, Model model):String
Descrizione	Implementa la funzionalità di smistare l Amministratore sulla view di gestioneProfiliView/visualizzazioneAmministratori.
Pre-condizione	**context:**VisualizzazioneProfiloArtigianoController::get(id, pagina, model) **pre:**artigianoDao.existsById(id)
Post-condizione	**context:**VisualizzazioneProfiloArtigianoController::get(id, pagina, model) **post:** model.addAttribute("annunci", annunci)!=null **post:** model.addAttribute("pagina", pagina)!=null **post:** model.addAttribute("pagineTotali", totalPages)!=null **post:** model.addAttribute("artigiano", artigiano)!=null
Invariante di classe	/

Nome classe	VisualizzazioneProfiloPersonaleController
Descrizione	Implementa il controller per la visualizzazione del profilo personale di un Cliente.
Metodi	+ get():String
Invariante di classe	/

Nome Metodo	+ get():String
Descrizione	Implementa la funzionalità di smistare il Cliente sulla view di gestioneProfiliView/visualizzazioneProfiloPersonale.
Pre-condizione	**context:**VisualizzazioneProfiloPersonaleController::get() **pre:** sessionCliente.getCliente().isPresent()
Post-condizione	
Invariante di classe	/

4 Package prenotazioniControl

Nome classe	EffettuaPrenotazioneController
Descrizione	Implementa il controller per effettuare una prenotazione ad un annuncio
Metodi	+ get(PrenotazioneForm prenotazioneForm, BindingResult bindingResult, Model model):String + post(PrenotazioneForm prenotazioneForm, BindingResult bindingResult):String
Invariante di classe	/

Nome Metodo	+ get(PrenotazioneForm prenotazioneForm, BindingResult bindingResult, Model model):String
Descrizione	Implementa la funzionalità di smistare il Cliente sulla view di prenotazioniView/effettuaPrenotazione.
Pre-condizione	**context:** EffettuaPrenotazioneController::get(prenotazioneForm, bindingResult, model) **pre:** not bindingResult.hasErrors() **pre:** sessionCliente.getCliente().isPresent()
Post-condizione	**context:** EffettuaPrenotazioneController::get(prenotazioneForm, bindingResult, model) **post:** model.getAttribute("orarioInizio") != null **post:** model.getAttribute("orarioFine") != null **post:** model.getAttribute("prezzo") != null **post:** model.getAttribute("idAnnuncio") != null
Invariante di classe	/

Nome Metodo	+ post(PrenotazioneForm prenotazioneForm, BindingResult bindingResult):String
Descrizione	Implementa la funzionalità di prenotazione ad un annuncio
Pre-condizione	**context:** EffettuaPrenotazioneController::post(prenotazioneForm, bindingResult) **pre:** not bindingResult.hasErrors() **pre:** sessionCliente.getCliente().isPresent()
Post-condizione	**context:** EffettuaPrenotazioneController::post(prenotazioneForm, bindingResult) **post:** sessionCliente.getCliente().getPrenotazioni().size() == @pre.sessionCliente.getCliente().getPrenotazioni().size() + 1
Invariante di classe	/

Nome classe	NumMaxPersoneController
Descrizione	Controller per il trasferimento del dato numero massimo di persone che possono partecipare ad una specifica visita in una specifica data
Metodi	+ post(Long idVisita, LocalDate datavisita):ResponseEntity<?>
Invariante di classe	/

Nome Metodo	+ post(Long idVisita, LocalDate datavisita):ResponseEntity<?>
Descrizione	Implementa la logica per il calcolo delle persone che già partecipano ad una visita in una specifica data
Pre-condizione	**context:** NumMaxPersoneController::post(idVisita, datavisita) **pre:** visitaDao.existsById(idVisita)
Post-condizione	**context:** NumMaxPersoneController::post(idVisita, datavisita) **post:** result.getBody().equals(numMaxPersone)
Invariante di classe	/

Nome classe	RicercaVisitaController
Descrizione	implementa il controller per la ricerca della visita
Metodi	+ post(Long idAnnuncio, LocalDate datavisita):ResponseEntity<?>
Invariante di classe	/

Nome Metodo	+ post(Long idAnnuncio, LocalDate datavisita):ResponseEntity<?>
Descrizione	Implementa la logica per la ricerca delle visite di un annuncio in una determinata data
Pre-condizione	**context:** RicercaVisitaController::post(idAnnuncio, datavisita) **pre:** annuncioDao.existsById(idAnnuncio) **pre:** dataVisita.isAfter(LocalDate.now())
Post-condizione	**context:**

	RicercaVisitaController::post(idAnnuncio, datavisita) **post:** result.getBody().equals(visite->asList())
Invariante di classe	/

Nome classe	**VisualizzazionePrenotazioniAnnuncioController**
Descrizione	Implementa il controller per la visualizzazione delle prenotazioni di un annuncio
Metodi	+ get(Long idAnnuncio):ModelAndView + post(Long idAnnuncio, LocalDate dataVisita, Integer pagina, Model model):String
Invariante di classe	/

Nome Metodo	+ get(Long idAnnuncio):ModelAndView
Descrizione	Implementa la funzionalità di smistare l'artigiano sulla view di prenotazioniView/visualizzazionePrenotazioniAnnuncio.
Pre-condizione	**context:** VisualizzazionePrenotazioniAnnuncioController::get(idAnnuncio) **pre:** annuncioDao.existsById(idAnnuncio) **pre:** sessionCliente.getCliente().isPresent() **pre:** sessionCliente.getCliente().get().getClass() == Artigiano.class
Post-condizione	**context:** VisualizzazionePrenotazioniAnnuncioController::get(idAnnuncio) **post:** result.getObject("idAnnuncio") == idAnnuncio
Invariante di classe	/

Nome Metodo	+ post(Long idAnnuncio, LocalDate dataVisita, Integer pagina, Model model):String
Descrizione	Implementa la funzionalità di visualizzazione delle prenotazioni dell'annuncio in una precisa data
Pre-condizione	**context:** VisualizzazionePrenotazioniAnnuncioController::post(idAnnuncio, dataVisita, pagina, model) **pre:** annuncioDao.existsById(idAnnuncio) **pre:** sessionCliente.getCliente().isPresent() **pre:** sessionCliente.getCliente().get().getClass() == Artigiano.class
Post-condizione	**context:** VisualizzazionePrenotazioniAnnuncioController::post(idAnnuncio, dataVisita, pagina, model) **post:** model.getAttribute("prenotazioni") != null **post:** model.getAttribute("pagina") != null **post:** model.getAttribute("pagineTotali") != null **post:** model.getAttribute("idAnnuncio") != null **post:** model.getAttribute("dataVisita") != null
Invariante di classe	/

Nome classe	VisualizzazionePrenotazioniPersonaliController
Descrizione	Implementa il controller per la visualizzazione delle prenotazioni personali
Metodi	+ get(Integer pagina):ModelAndView
Invariante di classe	/

Nome Metodo	+ get(Integer pagina):ModelAndView
Descrizione	Implementa la funzionalità di smistare il cliente sulla view di prenotazioniView/visualizzazionePrenotazioniPersonali
Pre-condizione	**context:** VisualizzazionePrenotazioniPersonaliController::get(pagina) **pre:** sessionCliente.getCliente().isPresent()
Post-condizione	**context:** VisualizzazionePrenotazioniPersonaliController::get(pagina) **post:** result.getObject("prenotazioni") != null **post:** result.getObject("pagina") != null **post:** result.getObject("pagineTotali") != null
Invariante di classe	/

5 Package visualizzazioneAnnunciControl

Nome classe	HomeAmministratoreController
Descrizione	Implementa il controller la visualizzazione della home degli amministratori.
Metodi	+ get(Model model):String
Invariante di classe	/

Nome Metodo	+ get(Model model):String
Descrizione	Implementa la funzionalità di smistare Amministratore alle view di visualizzazioneAnnunciView/homeAmministratore.
Pre-condizione	**context:** HomeAmministratoreController::get(model) **pre:**sessionAmministratore.getAmministratore.isPresent()
Post-condizione	**context:** HomeAmministratoreController::get(model) **post:**model.getAttribute("annuncirifiutati") != null;

	post:model.getAttribute("annunciapprovati") != null; **post:** model.getAttribute("annunciproposti") != null; **post:** model.getAttribute("annunciinrevisione") != null; **post:**model.getAttribute(annunci).equals(annuncio -> asList);
Invariante di classe	/

Nome classe	**HomeController**
Descrizione	Implementa il controller della visualizzazione della home del sito.
Metodi	+ get():ModelAndView
Invariante di classe	/

Nome Metodo	+ get(Model model):String
Descrizione	Implementa la funzionalità di smistare l'utente nella view di visualizzazioneAnnunciView/home.
Pre-condizione	
Post-condizione	**context:** HomeAmministratoreController::get(model) **post:**result.getObject(annunci).equals(annuncio -> asList);
Invariante di classe	/

Nome classe	**ImgController**
Descrizione	Implementa il controller per la visualizzazione delle foto.
Metodi	+ get(Long id, HttpServletResponse response)
Invariante di classe	/

Nome Metodo	+ get(Long id, HttpServletResponse response):void
Descrizione	Implementa la visualizzazione delle immagini
Pre-condizione	**context:**ImgController:: get(id, response) **pre:**fotoDao.existsById(id)
Post-condizione	**context:** HomeAmministratoreController::get(model) **post:**result.getbody()!=null
Invariante di classe	/

Nome classe	RicercaAnnunciController
Descrizione	Implementa il controller per la ricerca di un annuncio.
Metodi	+ get(String nomeAttivita, String provincia, Integer pagina, Model model):String
Invariante di classe	/

Nome Metodo	+ get(String nomeAttivita, String provincia, Integer pagina, Model model):String
Descrizione	Implementa la funzionalità della ricerca di un annuncio con o senza filtro province.
Pre-condizione	**context:**get(nomeAttivita, provincia, pagina, model)
Post-condizione	**context:**get(nomeAttivita, provincia, pagina, model) **post:**model.getAttribute("annunci") .equals(annuncio -> asList); **post:**model.getAttribute("pagina" !=null); **post:** model.getAttribute("nomeAttivita") !=null; **post:** model.getAttribute("provincia" !=null); **post:**model.getAttribute("pagineTotali") !=null;
Invariante di classe	/

Nome classe	VisualizzazioneDettagliAnnuncioController
Descrizione	Implementa il controller per la visualizzazione dei dettagli di un annuncio.
Metodi	+get(Long id):ModelAndView
Invariante di classe	/

Nome Metodo	+get(Long id):ModelAndView
Descrizione	Implementa la funzionalità di smistate l'utente nella view di visualizzazioneAnnunciView/visualizzazioneDettagliAnnuncio.
Pre-condizione	**context:**VisualizzazioneDettagliAnnuncioController::get(id) **pre:**annucioDao.existsById(id)
Post-condizione	**context:**VisualizzazioneDettagliAnnuncioController::get(id) **post:**result.getObject(totalFoto) != null **post:**result,getObject(annuncio) != null
Invariante di classe	/

Nome classe	VisualizzazioneListaAnnunciController
Descrizione	Implementa la funzionalità ricerca di liste di annunci è di smistare l'amministratore nella view di visualizzazioneAnnunciView/visualizzazioneListaAnnunci.
Metodi	+get(Annuncio.Stato stato, Integer pagina, Model model):String
Invariante di classe	/

Nome Metodo	+get(Annuncio.Stato stato, Integer pagina, Model model):String
Descrizione	Implementa la funzionalità ricerca di liste di annunci è di smistare l'amministratore nella view di visualizzazioneAnnunciView/visualizzazioneListaAnnunci.
Pre-condizione	**context:**VisualizzazioneListaAnnunciController::(stato, pagina, model) **pre:** sessionAmministratore.getAmministratore().isPresent()
Post-condizione	**context:**VisualizzazioneListaAnnunciController::get(stato, pagina, model) **post:**model.getAttribute("pagineTotali") !=null; **post:**model.getAttribute("annunci") .equals(annuncio -> asList); **post:**model.getAttribute("pagina" !=null);
Invariante di classe	/

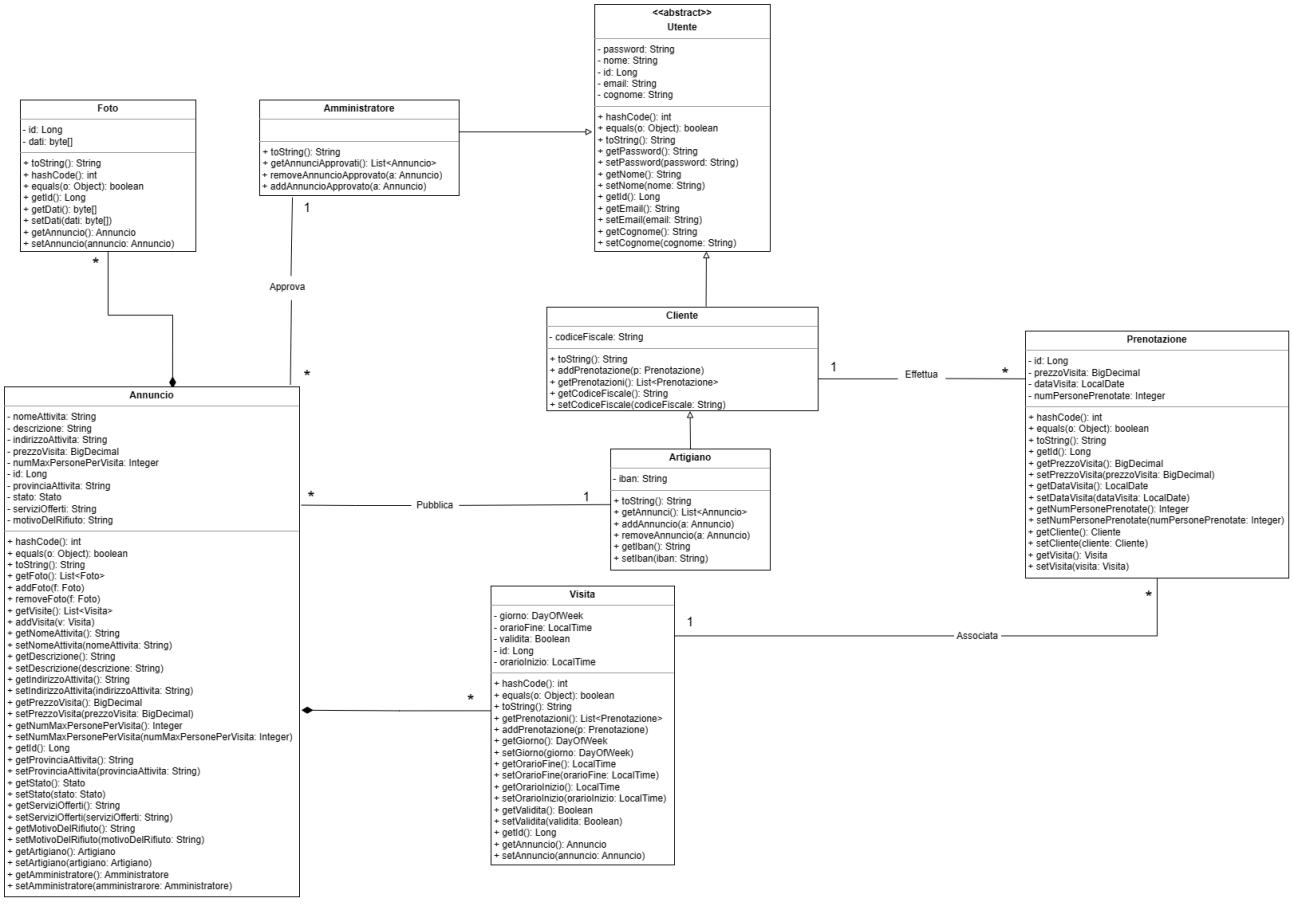
Nome classe	VisualizzazioneListaAnnunciSottomessiController
Descrizione	Implementa il controller per la visualizzazione di una lista di Annunci sottomessi.
Metodi	+get():ModelAndView
Invariante di classe	/

Nome Metodo	+get(Annuncio.Stato stato, Integer pagina, Model model):String
Descrizione	Implementa la funzionalità di smistare l Artigiano nella view di visualizzazioneAnnunciView/visualizzazioneListaAnnunciSottomessi.
Pre-condizione	**context:**VisualizzazioneListaAnnunciSottomessiController::get()

Post-condizione	**context:**VisualizzazioneListaAnnunciController::(stato, pagina, model) **post:**result.getObject("annunci").equals(annuncio -> asList);
Invariante di classe	/

4 Class Diagram

Di seguito è riportato il class diagram delle entity, ottenuto dal raffinamento del class diagram riportato nel RAD.



[link all'immagine]

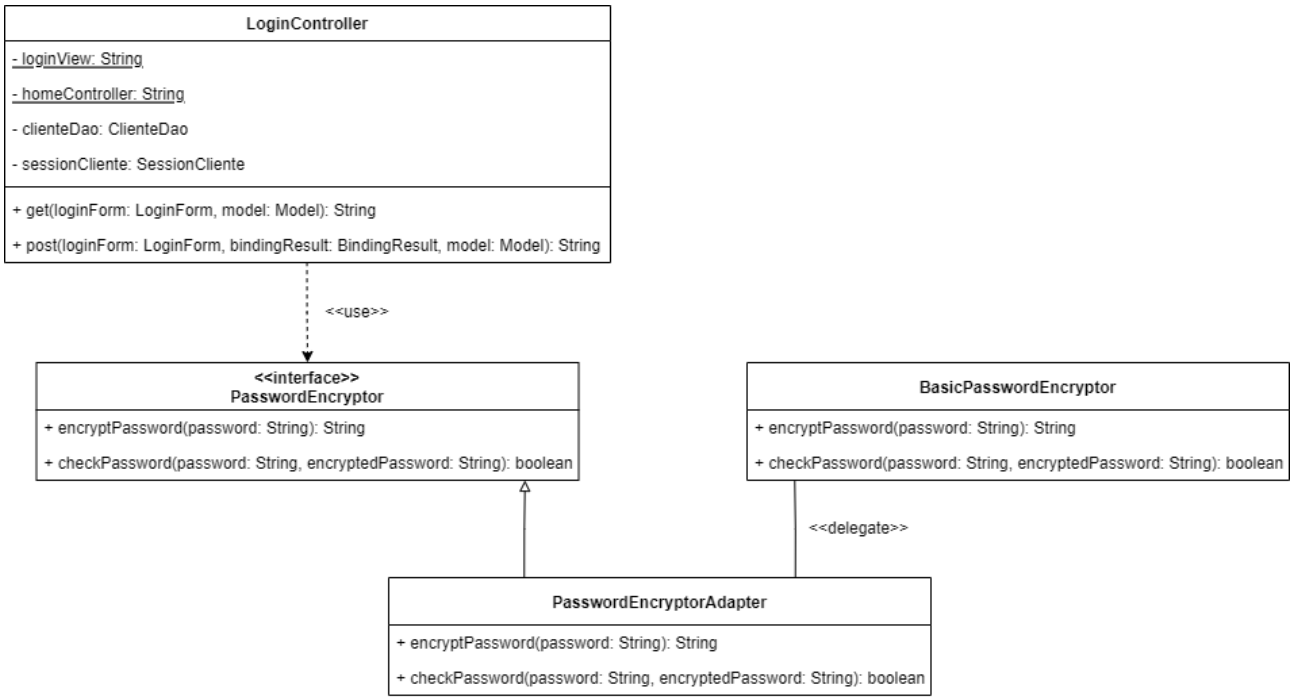
5 Design Patterns

Nella presente sezione si andranno a descrivere e dettagliare i design patterns utilizzati nello sviluppo del sistema Just Traditions:

Adapter

L'Adapter è un design pattern strutturale, ovvero quei design pattern che si occupano delle modalità di composizione di classi e oggetti per formare strutture complesse. Adapter, in particolare, è utile per l'utilizzo di componenti off-the-shelf, infatti le componenti esistenti sono incapsulate e in questo modo si ha una separazione tra il sistema e le componenti, minimizzando l'impatto delle componenti esistenti sul progetto. Si implementa attraverso l'utilizzo di un'interfaccia e di una classe "adapter", che la implementa, delegando il comportamento alla componente esistente.

Just Traditions pone fra i design goals principali la sicurezza delle password degli utenti, che devono essere salvate in forma criptata. Per soddisfare tale design goal si è scelto l'utilizzo della libreria Jasypt, che rappresenta un punto di riferimento per la crittografia in Java. È stato deciso quindi l'utilizzo del design pattern Adapter per permettere l'utilizzo della classe BasicPasswordEncryptor, offerta dalla libreria sopra citata, incapsulandone il comportamento, così da realizzare il massimo disaccoppiamento fra il progetto e la libreria esterna, in modo tale che se in futuro si rendesse necessario l'utilizzo di una diversa implementazione per il PasswordEncryptor, eventualmente con un algoritmo più efficiente, la sostituzione risultasse quanto più immediata possibile.



6 Glossario

- **jQuery** è una libreria di JavaScript per applicazioni web il cui obiettivo è quello di semplificare la selezione, la manipolazione, la gestione degli eventi e l'animazione di elementi DOM in pagine HTML, nonché semplificare le funzionalità di AJAX.
- **AJAX** (Asynchronous JavaScript and XML) è una tecnica di sviluppo software per la realizzazione di applicazioni web interattive, basandosi su uno scambio di informazioni in background fra client e server.
- **Bootstrap** è uno dei framework CSS più utilizzati. Bootstrap può considerarsi lo standard de facto dei framework CSS per lo sviluppo di interfacce Web.