
Laurea Triennale in informatica-Università di Salerno
Corso di *Ingegneria del Software*- Prof. C. Gravino

# Object Design Document

# Turing Careers

| | |
|---|---|
| Riferimento | |
| Versione | 1.0 |
| Data | 15/02/24 |
| Destinatario | Prof. C. Gravino |
| Presentato da | Antonio Pagnotta,<br>Antonino Lorenzo,<br>Claudio Gaudino,<br>Jacopo Passariello |
| Approvato da | |

*Pagina intenzionalmente lasciata in bianco.*

# Revision History

| Data | Versione | Descrizione | Autori |
|---|---|---|---|
| 13/01/24 | 0.1 | Prima Stesura | Antonino Lorenzo, Claudio Gaudino |
| 01/02/2024 | 0.2 | Specifica delle Interfacce | Antonio Pagnotta, Antonino Lorenzo, Claudio Gaudino |
| 04/02/24 | 0.3 | Scelta di Design Pattern | Antonino Lorenzo, Jacopo Passariello |
| 10/02/24 | 0.4 | Definizione Packages | Antonio Pagnotta, Jacopo Passariello |
| 13/02/24 | 0.5 | Correzione Packages | Jacopo Passariello |
| 15/02/24 | 1.0 | Revisione Finale del Documento | Antonio Pagnotta, Antonino Lorenzo, Jacopo Passariello |

# Sommario

# 1. Introduzione

## 1.1 Object Design Trade-Offs

### 1.1.1. Sviluppare vs Comprare:

Al fine di facilitare lo sviluppo del sistema si è deciso di adottare Componenti COTS con lo scopo di velocizzare la scrittura di codice e di implementare funzionalità complesse. I componenti COTS utilizzati sono descritti nella sottosezione "5.2 Componenti COTS" nella sezione "5. Elementi di Riuso".

### 1.1.2. Prestazioni vs Affidabilità

Dovendo il sistema gestire dati sensibili e di struttura complessa, si preferisce garantire un maggior controllo di input e consistenza, effettuando diversi controlli sia a livello client-side che server-side, a scapito della latenza.

### 1.1.3 Spazio di Memoria vs Tempo di Risposta:

Al fine di migliorare i tempi di risposta, è stato deciso di distribuire copie ridondanti dei dati all'interno del sistema. In particolare vengono conservate informazioni non sensibili all'interno del client Utente al fine di velocizzare i tempi di risposta delle pagine.

**1.1.4 Manutenibilità** vs Complessità:
Al fine di rendere il sistema supportabile nel lungo periodo si è scelto di mantenere basso l'accoppiamento tra classi.

**1.1.5 Costi** vs Affidabilità: Non essendo un sistema critico, al fine di contenere i costi di gestione la disponibilità del sistema sarà ristretta nell'arco delle 24 ore al fine di effettuare manutenzione.

**1.1.6 Centralizzazione** vs Distribuzione: La distribuzione del sistema su più nodi offre maggiore ridondanza e migliori performance, ma comporta una complessità più elevata. Per mantenere la gestione semplice, si opta per il deployment su una singola macchina, in modo da mantenere comunque dei moduli separati che possono essere eseguiti su nodi diversi.

**1.1.7 Stabilità delle tecnologie esistenti** vs Tecnologie Emergenti: Per assicurare stabilità, si preferisce utilizzare tecnologie consolidate anziché emergenti, pur sacrificando alcune performance per semplificare lo sviluppo e ridurre lo sforzo del team.

## 1.2 Linee Guida per la Documentazione delle Interfacce

Verranno utilizzate diverse convenzioni condivise dai componenti del team per rendere uniformi e consistenti definizione e documentazione delle interfacce, quali:

- Ad ogni classe sarà assegnato un nome univoco e rappresentativo delle funzionalità di cui essa è responsabile, utilizzando upper camel case.
- Ad ogni metodo sarà assegnato un nome che sia un verbo, rappresentativo della funzionalità specifica implementata dal metodo, utilizzando lower camel case per le classi da implementare in Java e snake case per quella da implementare in Python.
- In caso di errore, questo sarà comunicato dai metodi tramite il lancio di un'eccezione.

Inoltre, per automatizzare il controllo della qualità della scrittura del codice sarà effettuato tramite GitHub Actions e l'utilizzo di:

- Checkstyle, per il codice Java del Sistema Core
- Pylint, per il codice Python del Sistema di Raccomandazione

### 1.3 Definizioni, Acronimi e Abbreviazioni

Elenco delle definizioni, acronimi ed abbreviazioni utilizzate all'interno del documento:

- **UpperCamelCase**: pratica che prevede la scrittura di parole composte o frasi unendole tra loro, lasciando le loro iniziali maiuscole;
- **lowerCamelCase**: pratica che prevede la scrittura di parole composte o frasi unendole tra loro, in modo tale che le parole in mezzo alla frase abbiano l'iniziale maiuscola;
- **Presentation Layer**: nel pattern Three-Tier rappresenta la parte che si occupa della logica di visualizzazione dell'applicazione
- **Business Layer**: nel pattern Three-Tier rappresenta la parte che si occupa della logica di business dell'applicazione
- **Data Layer**: nel pattern Three-Tier rappresenta la parte che si occupa dell'interazione con il database.
- **JavaDoc:** strumento di documentazione automatica utilizzato per descrivere le interfacce delle classi implementate garantendo accessibilità e leggibilità.

### 1.4 Riferimenti

Bernd Bruegge, Allen H. Dutoit - Object-Oriented Software Engineering.

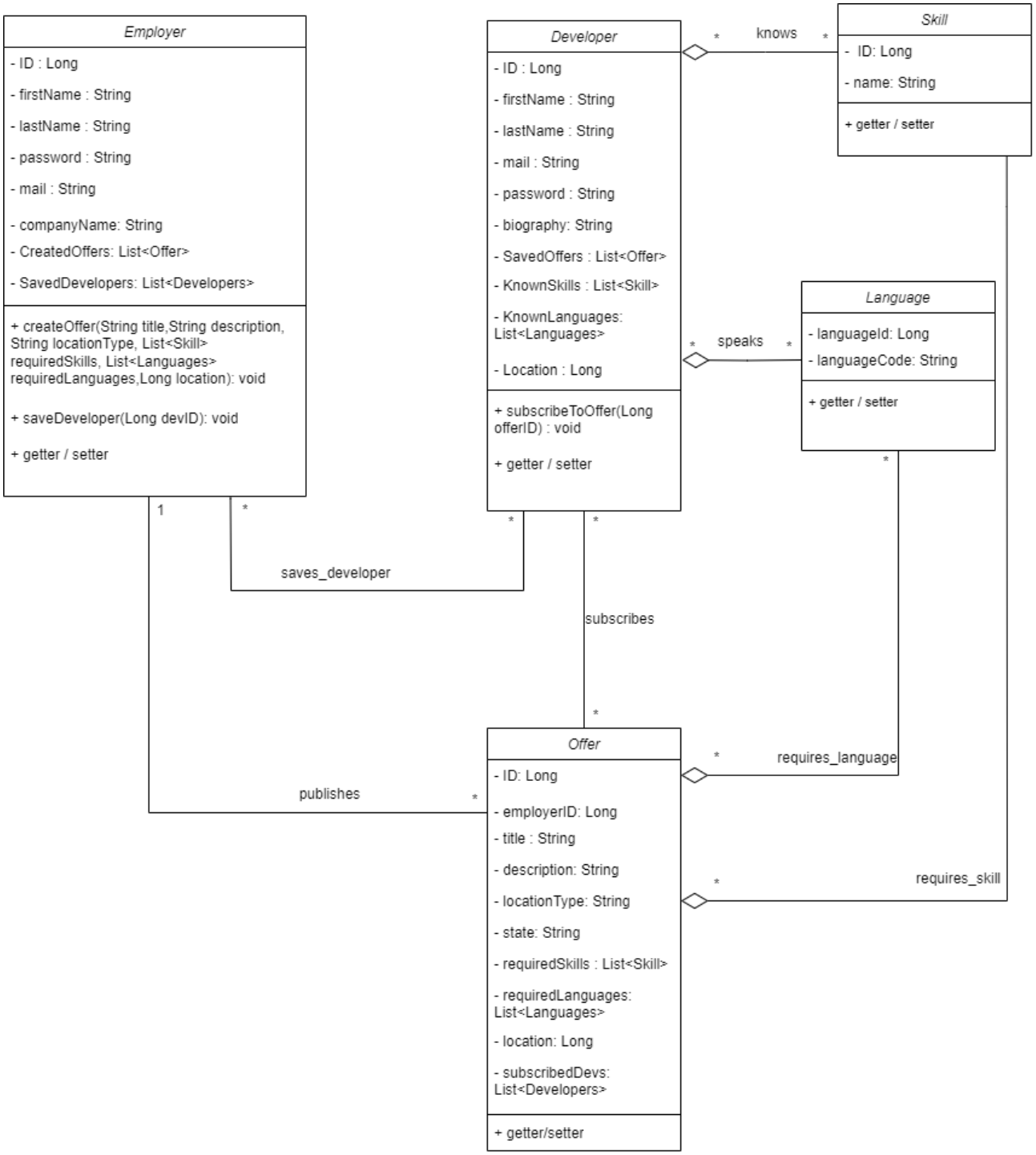
Dispense del Prof. C. Gravino.

Turing Careers RAD.

Turing Careers SOW.

Turing Careers SDD.

# 2. Class Diagram Ristrutturato




E' stato aggiornato il class diagram al fine di includere i tipi e gli specificatori di accesso ai metodi.

# 3. Packages

## 2.1 Turing Careers Core

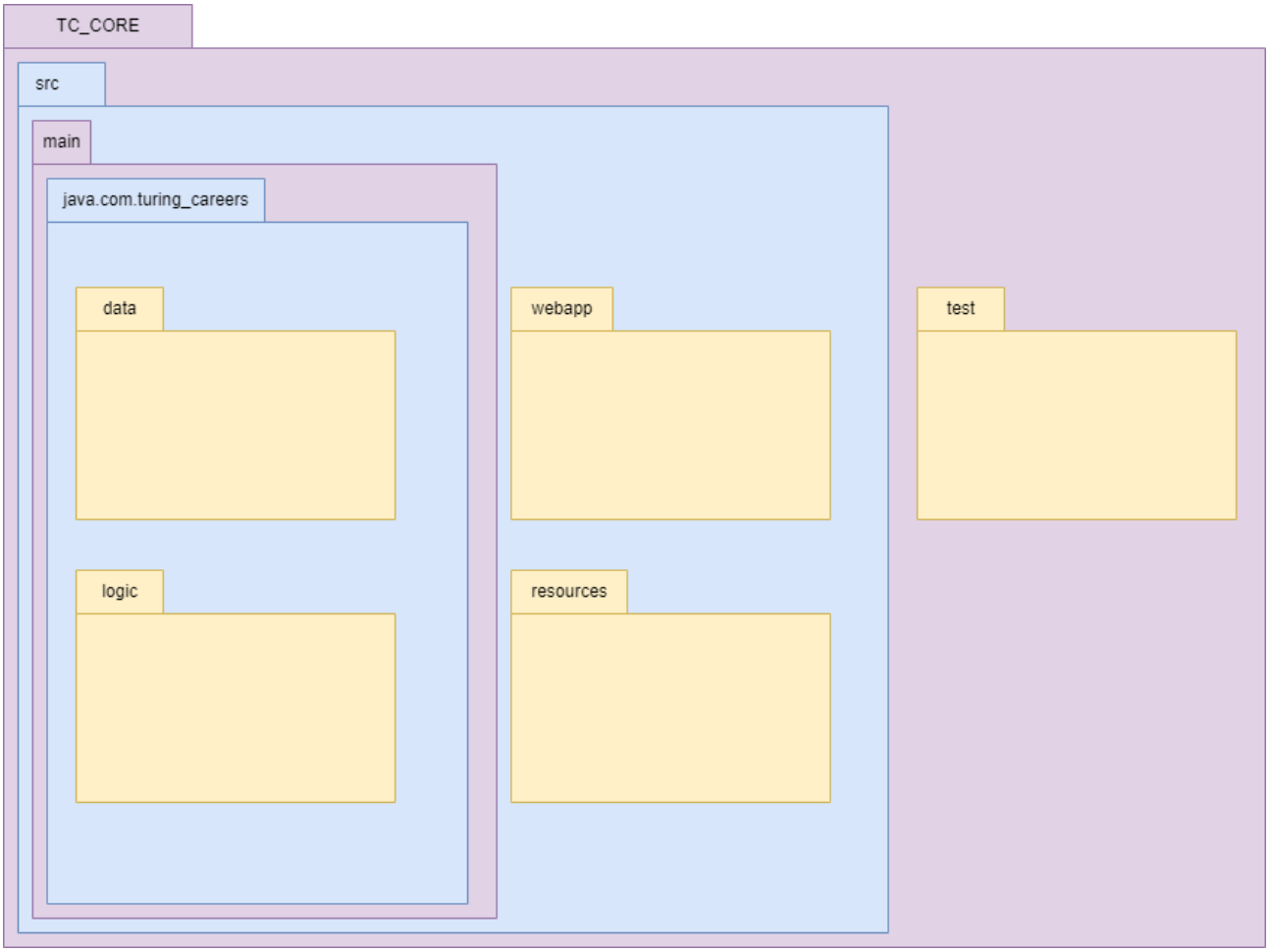
**- .github/*workflows*** : contiene i file per specificare l'esecuzione delle GitHub Actions

**- src** : contiene il sistema core

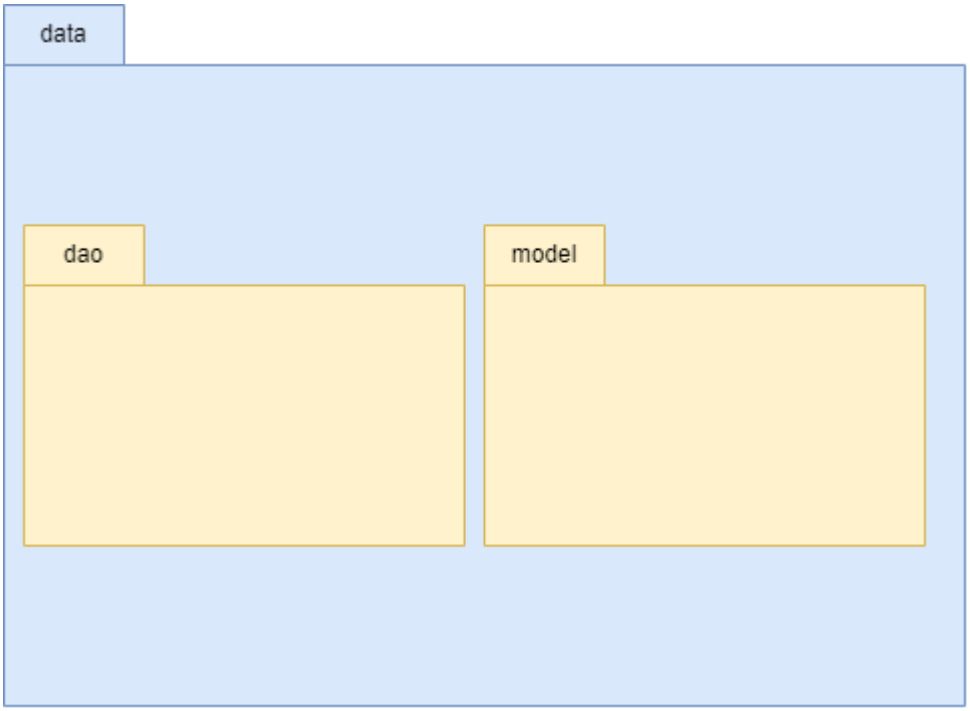
**- *main*** : contiene le principali componenti del sistema in esecuzione sul Server

- ***java.com_turing_careers.logic :***
  - ***java.com.turing careers.control*** : contiene le classi che gestiscono le richieste http dei client (servlet).
  - ***java.com.turing_careers.service*** : contiene le classi per la logica di business.
    - *auth:* contiene le classi che gestiscono l'autenticazione degli utenti.
    - *offer :* contiene le classi che gestiscono le offerte.
    - *search:* contiene le classi che gesticono la ricerca.
    - *suggestions:* contiene le classi che si occupano di interfacciarsi con il sistema di raccomandazione.
    - *user:* contiene le classi che gestiscono gli utenti.
- ***java.com.turing_careers.data :***
  - *model :* contiene le classi entity.
  - *dao :* contiene i dao per poter interagire con il database.

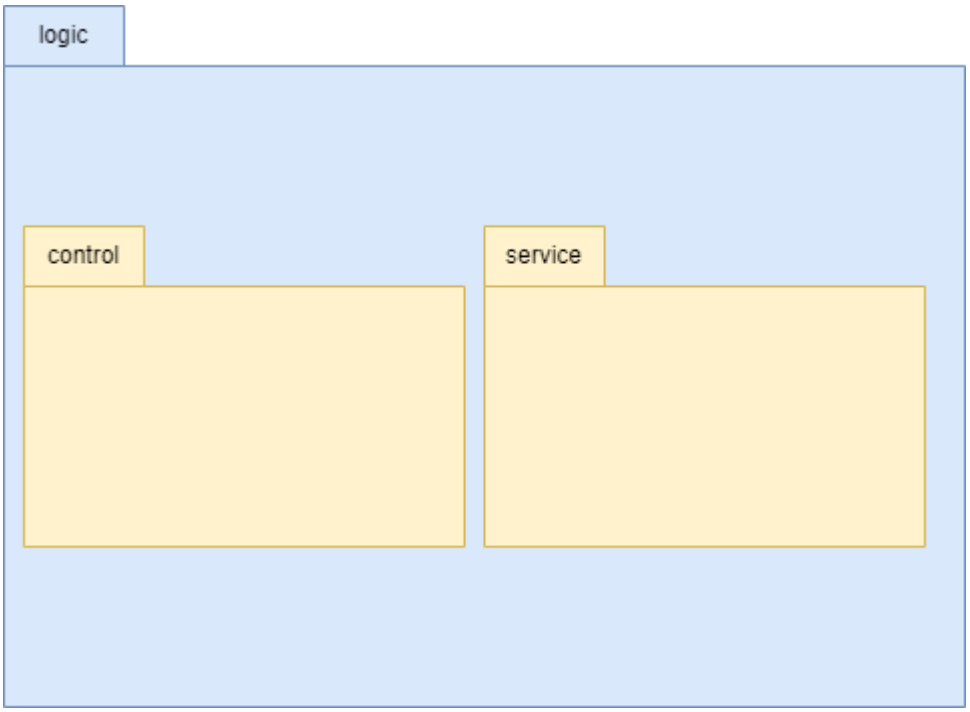
- **webapp :** contiene le pagine html e le relative risorse.
  - *css:* contiene i file di layout per la visualizzazione delle pagine HTML del sistema.
  - *font:* contiene i tipo di carattere necessari ai CSS.
  - *js:* contiene i file javascript che gestiscono l'interazione con l'utente.
  - *static:* contiene le risorse necessarie(es. immagini).
  - *WEB-INF:* contiene risorse che sono private all'applicazione web.
- **resources/META-INF** : contiene i file di configurazione per la persistenza.
- **- test** : contiene le classi per eseguire il testing del sistema.


TC_CORE
src
main
java.com.turing_careers
data
webapp
test
logic
resources

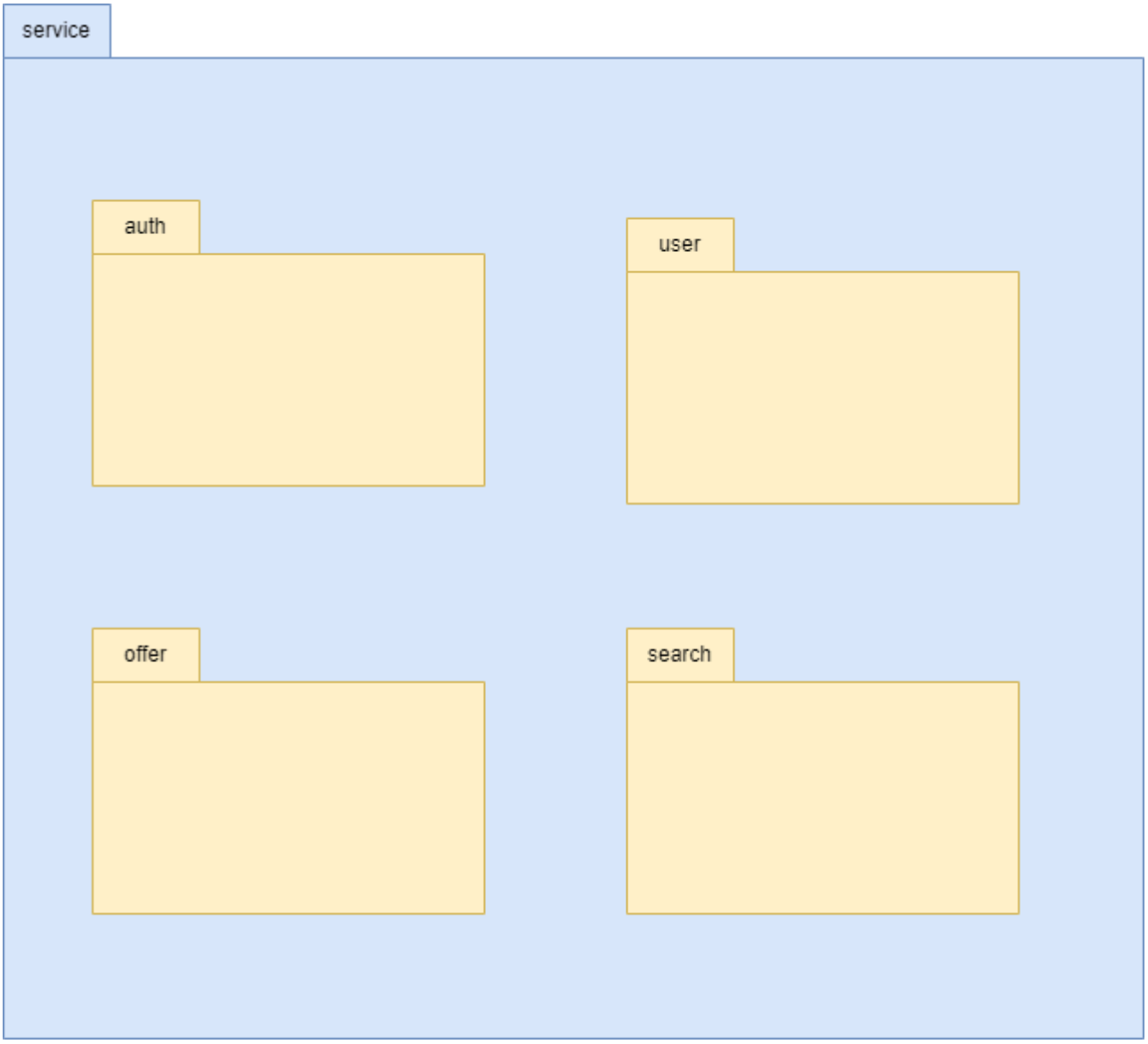
### 3.1.1 Data Package




### 3.1.2 Logic Package



### 3.1.3 Service Package

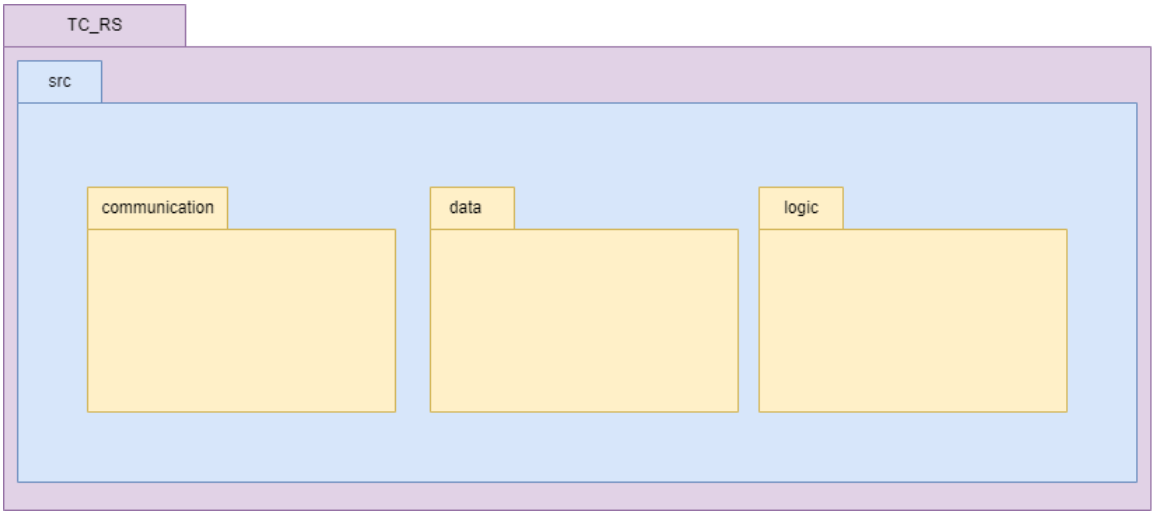


## 3.2 Turing Careers Recommender API

**- .github/*workflows*** : contiene i file per l'esecuzione delle GitHub Actions

**- src** :

- ***communication***: contiene i moduli relativi all'esposizione dell'API al client
- ***logic*** : contiene i moduli e le classi che raccolgono i dati dal database e effettuano ricerca e raccomandazione, è il core del progetto di FIA.
- ***data :***
  - *orm :* contiene i moduli le classi che rappresentano Offerte e Profili di Sviluppatori.
  - *dao :* contiene le classi per interagire con le entità le Offerte e i Profili degli Sviluppatori.



# 4. Class interfaces

## 4.1 JavaDoc

Si è deciso di utilizzare il tool JavaDoc per la specifica delle interfacce nei package data e control, segue il link alle GitHub pages:
https://jacopopassariello.github.io/TuringCareers/

## 4.2 Definizione Interfacce:

Si è scelto di documentare le interfacce delle classi all'interno del Package **service** per via della loro importanza nel sistema. I contratti sono stati definiti utilizzando **Linguaggio Naturale**.

**1. Package Authenticator**

**Package java.com.turing_careers.logic.service.auth**

**Authenticator:**

| **Nome Classe** | **Authenticator** |
|---|---|
| **Descrizione** | Classe astratta, implementa la logica per cambiare algoritmo di hashing per le password e dichiara i metodi relativi all'autenticazione. |
| **Metodi** | + loginUser(email: String, password: String): User<br>+ signupUser(user: User): User<br>+ setEncryptionStrategy(strategy: Encryption Strategy) |
| **Invariante** | / |

| **Nome Metodo** | **+ setEncryptionStrategy(strategy: EncryptionStrategy)** |
|---|---|
| **Descrizione** | Riceve l'implementazione di un algoritmo di hashing per le password che verrà usato dalle classi che implementano Authenticator. |
| **Pre-Condizione** | L'oggetto strategy non deve essere nullo. |
| **Post-Condizione** | strategy viene utilizzata come strategia di crittazione da parte dell'istanza di Authenticator. |

**Argon2Encryption:**

| Nome Classe | Argon2Encryption |
|---|---|
| Descrizione | È un wrapper per l'implementazione dell'algoritmo di hashing Argon2 di Spring Security Crypto. |
| Metodi | + encrypt(str: String): String<br>+ verify(raw: String, original: String) |
| Invariante | / |

| Nome Metodo | + encrypt(str: String): String |
|---|---|
| Descrizione | Effettua l'hashing della stringa fornita in input. |
| Pre-Condizione | L'oggetto str non è null e non è vuoto. |
| Post-Condizione | La stringa risultante è criptata tramite algoritmo di hashing Argon2. |

| Nome Metodo | + verify(raw: String, original: String) |
|---|---|
| Descrizione | Verifica che la stringa raw corrisponda alla password original. |
| Pre-Condizione | La stringhe raw e original non devono essere nulle e non devono essere vuote. |
| Post-Condizione | Il metodo non restituisce eccezioni se la verifica della password ha avuto successo, altrimenti lancia l'eccezione InvalidCredentialsException nel caso di password non verificata. |

**DeveloperAuthenticator:**

| **Nome Classe** | **DeveloperAuthenticator** |
|---|---|
| **Descrizione** | Implementa Authenticator, fornisce i servizi di autenticazione per Sviluppatori. |
| **Metodi** | + loginUser(email: String, password: String)<br>+ signupUser(user: User) |
| **Invariante** | / |

| **Nome Metodo** | **+ loginUser(email: String, password: String)** |
|---|---|
| **Descrizione** | Il metodo effettua il login dell'utente con email e password passati al metodo. |
| **Pre-Condizione** | Le stringhe email e password non devono essere nulle o vuote. |
| **Post-Condizione** | Se l'email è presente nel database e l'account ha come password la password inserita, allora restituisce il developer, altrimenti lancia InvalidCredentialsException. |

| **Nome Metodo** | **+ signupUser(user: User)** |
|---|---|
| **Descrizione** | Il metodo effettua la creazione del profilo dell'utente passato come parametro. |
| **Pre-Condizione** | L'oggetto user non è null. |
| **Post-Condizione** | L'utente se validato è inserito nel database, altrimenti viene lanciata l'eccezione ValidationException. |

**EmployerAuthenticator:**

| Nome Classe | EmployerAuthenticator |
|---|---|
| Descrizione | Implementa Authenticator, fornisce i servizi di autenticazione per Employer. |
| Metodi | + loginUser(email: String, password: String)<br>+ signupUser(user: User) |
| Invariante | / |

| Nome Metodo | + loginUser(email: String, password: String) |
|---|---|
| Descrizione | Il metodo effettua il login dell'utente con email e password passati al metodo. |
| Pre-Condizione | Le stringhe email e password non devono essere nulle o vuote. |
| Post-Condizione | Se l'email è presente nel database e l'account ha come password la password inserita, allora restituisce l'employer, altrimenti lancia InvalidCredentialsException. |

| Nome Metodo | + signupUser(user: User) |
|---|---|
| Descrizione | Il metodo effettua la creazione del profilo dell'utente passato come parametro. |
| Pre-Condizione | L'oggetto user non è null. |
| Post-Condizione | L'utente se validato è inserito nel database, altrimenti viene lanciata l'eccezione ValidationException. |

## Package java.com.turing_careers.logic.service.search

### ClientFactory:

| Nome Classe | ClientFactory |
|---|---|
| Descrizione | Viene usata come interfaccia per creare un RecommenderEngine o un UpdateEngine del tipo specificato da ClientType. |
| Metodi | + setType(type: ClientType): ClientFactory<br>+ getRecommenderEngine(): RecommenderEngine<br>+ setRecommenderEngine(): UpdateEngine |
| Invariante | / |

| Nome Metodo | + setType(type: ClientType): ClientFactory |
|---|---|
| Descrizione | Specifica se il Client deve effettuare richieste all'API sul percorso per Sviluppatori o Offerte. |
| Pre-Condizione | / |
| Post-Condizione | Il ClientType specificato è salvato in ClientFactory. |

| Nome Metodo | + getRecommenderEngine(): RecommenderEngine |
|---|---|
| Descrizione | Crea e restituisce un nuovo RecommenderEngine del tipo ClientType. |
| Pre-Condizione | Deve essere stato invocato setType e specificato il ClientType. |
| Post-Condizione | L'istanza restituita di RecommenderEngine è del tipo ClientType salvato da ClientFactory. |

| Nome Metodo | + getUpdateEngine(): UpdateEngine |
|---|---|
| Descrizione | Crea e restituisce un nuovo UpdateEngine del tipo ClientType. |
| Pre-Condizione | Deve essere stato invocato setType e specificato il ClientType. |
| Post-Condizione | L'istanza restituita di UpdateEngine è del tipo ClientType salvato da ClientFactory. |

**RecommenderEngine**

| Nome Classe | RecommenderEngine |
|---|---|
| Descrizione | Espone i metodi per effettuare la ricerca, internamente usa APIClient per effettuare le richieste alla TC Recommender APII. |
| Metodi | + search(offer: Offer): List<Developer><br>+ search(query: String, user: Developer): List<Offer> |
| Invariante | / |

| Nome Metodo | + search(offer: Offer): List<Developer> |
|---|---|
| Descrizione | Chiama APIClient per effettuare una ricerca di Sviluppatori tramite una offerta; effettuata la richiesta ad APIClient elabora il risultato di APIClient per ottenere una lista di Sviluppatori. |
| Pre-Condizione | Il ClientType deve essere ClientType.DEVELOPER.<br>L'Offerta fornita in input deve essere valida. |
| Post-Condizione | La lista di developer restituita non è nulla. |

| Nome Metodo | + search(query: String, user: Developer): List<Offer> |
|---|---|
| Descrizione | Chiama APIClient per effettuare una ricerca di Offerte in base ad una stringa di ricerca e l'utente che effettua la ricerca; effettuata la richiesta ad APIClient elabora il risultato di APIClient per ottenere una lista di Offerte. |
| Pre-Condizione | Il ClientType deve essere ClientType.OFFER.<br>Il parametro query deve essere una stringa valida. |
| Post-Condizione | La lista di offer restituita non è nulla. |

**UpdateEngine:**

| Nome Classe | UpdateEngine |
|---|---|
| Descrizione | Espone i metodi per aggiornare TC Recommender API quando viene modificato lo stato nel database di un Item, internamente usa APIClient per effettuare le richieste. |
| Metodi | + add(item: Item)<br>+ modify(item: Item)<br>+ delete(item: Item) |
| Invariante | / |

| Nome Metodo | + add(item: Item) |
|---|---|
| Descrizione | Aggiorna il sistema TC Recommender API con un nuovo Item. |
| Pre-Condizione | L'Item deve essere del tipo specificato da ClientType.<br>L'Item non deve essere presente in TC Recommender API. |
| Post-Condizione | L'item è stato aggiunto al sistema e reso persistente nel database. |

| Nome Metodo | + modify(item: Item) |
|---|---|
| Descrizione | Aggiorna l'Item contenuto nel sistema TC Recommender API con una nuova versione dell'Item. |
| Pre-Condizione | L'Item deve essere del tipo specificato da ClientType.<br>L'Item deve essere presente in TC Recommender API. |
| Post-Condizione | La versione aggiornata dell'item è stata resa persistente nel database. |

| Nome Metodo | + delete(item: Item) |
|---|---|
| Descrizione | Rimuove dal sistema TC Recommender API l'Item specificato. |
| Pre-Condizione | L'Item deve essere del tipo specificato da ClientType.<br>L'Item deve essere presente in TC Recommender API. |
| Post-Condizione | L'item è stato rimosso dal sistema. |

**APIClient:**

| Nome Classe | APIClient |
|---|---|
| Descrizione | Implementa la logica per effettuare una richiesta a TC Recommender API, è composto dal Client JAX-RS e da RequestBody. |
| Metodi | + sendRequest(String endpoint): Optional<String> |
| Invariante | API_ENDPOINT: una stringa invariante che contiene l'indirizzo base di T TC Recommender API. |

| Nome Metodo | + sendRequest(String endpoint): Optional<String> |
|---|---|
| Descrizione | Effettua una POST request a TC Recommender API tramite l'endpoint specificato dall'Engine chiamante e manda il contenuto di RequestBody. |
| Pre-Condizione | L'endpoint specificato deve essere esposto da TC Recommender. |
| Post-Condizione | / |

**RequestBody:**

| Nome Classe | RequestBody |
|---|---|
| Descrizione | Rappresenta il contenuto JSON da inviare a TC Recommender API, viene inizializzato con il contenuto da inviare e lo converte in formato JSON. |
| Metodi | + sendRequest(String endpoint): Optional<String> |
| Invariante | / |

| Nome Metodo | + toJSON(): String |
|---|---|
| Descrizione | Ritorna il contenuto del body JSON da inviare a TC Recommender API. |
| Pre-Condizione | RequestBody è stato inizializzato con gli oggetti che saranno contenuti nella stringa JSON. |
| Post-Condizione | / |

## Package java.com.turing_careers.logic.service.offer

### OfferManager:

| Nome Classe | OfferManager |
|---|---|
| Descrizione | Espone i metodi per effettuare creare, modificare e cambiare stato ad un'offerta. |
| Metodi | + createOffer(offer: Offer): void<br>+ editOffer(offer: Offer): void<br>+ createOffer(offer: Offer): void<br>- checkValidity(offer: Offer): void |
| Invariante | / |

| Nome Metodo | + createOffer(offer: Offer): void |
|---|---|
| Descrizione | Effettua la persistenza dell'offerta. |
| Pre-Condizione | L'oggetto offer e i suoi campi non sono null, tranne per Location che non deve essere null solo se il campo "type" dell'offerta è uguale a "ON_SITE". |
| Post-Condizione | offer è resa persistente nel database. |

| Nome Metodo | + deleteOffer(offer: Offer): void |
|---|---|
| Descrizione | Elimina un'offerta dal database. |
| Pre-Condizione | L'offerta offer non deve essere null. |
| Post-Condizione | offer non è presente nel database. |

| **Nome Metodo** | **+ editOffer(offer: Offer): void** |
|---|---|
| **Descrizione** | Effettua la persistenza dell'offerta modificata. |
| **Pre-Condizione** | L'oggetto offer e i suoi campi non sono null, tranne per Location che non deve essere null solo se il campo "type" dell'offerta è uguale a "ON_SITE". |
| **Post-Condizione** | offer è stata modificata nel database. |

| **Nome Metodo** | **+ subscribeToOffer(offer: Offer, dev: Developer): void** |
|---|---|
| **Descrizione** | Iscrive un developer ad una offerta aggiungendo il profilo developer alla lista di dev iscritti. |
| **Pre-Condizione** | Gli oggetti offer e dev non sono null e dev non è già contenuto nella lista di sviluppatori iscritti salvata in offer. |
| **Post-Condizione** | dev è stato aggiunto alla lista di sviluppatori iscritti ad offer. |

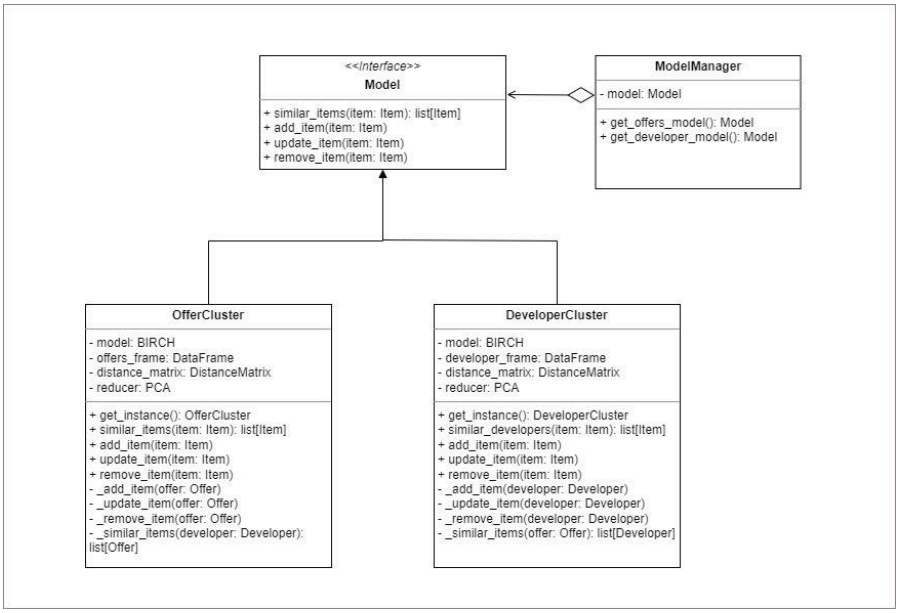
# 5. Elementi di riuso

## 5.1 Design Pattern

### Bridge

Nel sistema di Raccomandazione una parte soggetta a notevoli cambiamenti è il modello di Machine Learning, che fornisce le funzionalità di raccomandazione. Questo è vero in quanto potrebbe essere **necessario modificare l'algoritmo di ML** utilizzato oppure cambiare a quest'ultimo parametri. Infine, per via dei requisiti del sistema, sono state già fornite **due implementazioni concrete del modello** da chiamare in base a che tipo di richiesta di raccomandazione è stata inviata al modulo.
Questo problema è risolto tramite l'uso del Design Pattern **Bridge**:

- Il client è stato **isolato** dalla specifica implementazione della funzionalità.
- **Gerarchie ortogonali** di modelli per Offerte e Sviluppatori possono essere **implementate senza stravolgere il sistema**.




Il beneficio dell'uso del Bridge è un **minore accoppiamento** e una **maggiore estensibilità** al costo di una **struttura più complessa**.

## 5.2 Off-the-Shelf Components

Segue una lista di Componenti COTS utilizzati all'interno del sistema a fini di fornire servizi e facilitare lo sviluppo:

- **Spring Security Crypto:** componente dello Spring Framework che contiene algoritmi di cifratura come Argon2.
- **Jackson:** libreria Java per il parsing di JSON.
- **Lombok**: libreria di annotazioni per ridurre la scrittura di codice boilerplate di Java.
- **TalkJS:** componente esterna che realizza il servizio di comunicazione tra utenti.
- **SQLAlchemy:** realizza una interfaccia per comunicare con il database MySQL all'interno del modulo di raccomandazione.
- **scikit-learn:** fornisce librerie per l'implementazione di modelli di Machine Learning.
- **Jakarta EE 2**: fornisce il framework operativo del sistema core.