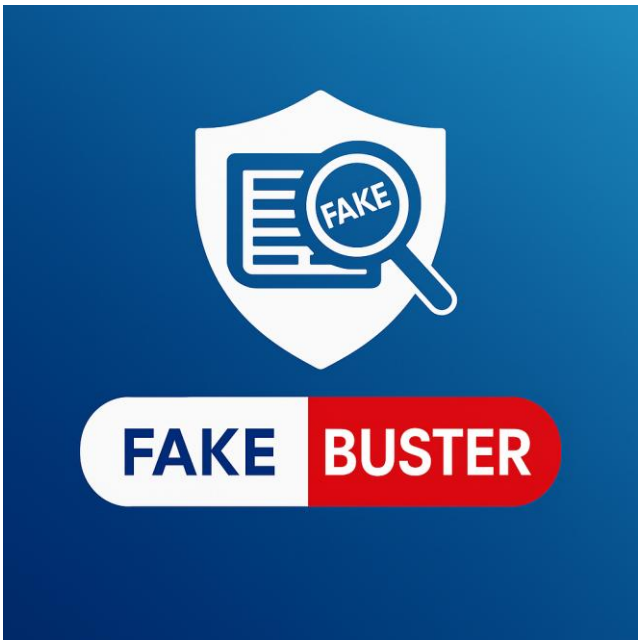
**Università degli Studi di Salerno**
**Corso di Ingegneria del Software**

# FakeBuster
# Object Design Document
# Versione 1.0




Data: 14/12/2025

### Coordinatore del progetto:

| Nome | Matricola |
|---|---|
| Bruno Santo | 0512116161 |
| Emiliano Di Giuseppe | 0512119155 |

**Partecipanti:**

| Nome | Matricola |
|---|---|
| Bruno Santo | 0512116161 |
| Emiliano Di Giuseppe | 0512119155 |
| | |
| | |
| | |
| | |

| | |
|---|---|
| **Scritto da:** | Bruno Santo & Emiliano Di Giuseppe |

# Revision History

| Data | Versione | Descrizione | Autore |
|---|---|---|---|
| 14/12/2025 | 1.0 | Creazione del Object Design Document | Bruno Santo & Emiliano Di Giuseppe |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

# Indice

# 1. Introduction

FakeBuster Social nasce come piattaforma sperimentale che unisce intelligenza artificiale e partecipazione umana per creare un ecosistema informativo più affidabile. L'obiettivo principale è ridurre la diffusione di notizie false alla radice, bloccandone la pubblicazione già in fase di inserimento mediante una validazione preventiva . Questo documento (Object Design Document) descrive i dettagli di progettazione degli oggetti per il sistema proposto, traducendo l'architettura logica definita nel System Design Document in specifiche di implementazione per l'ambiente Python/Flask.

## 1.1. Object design trade offs

- **Sicurezza vs Tempi di risposta:** Il sistema verrà implementato privilegiando l'integrità referenziale e la coerenza dei dati (ACID) garantita dal database relazionale MySQL , a discapito di una latenza potenzialmente maggiore rispetto a soluzioni NoSQL, al fine di assicurare la tracciabilità delle decisioni di moderazione.

- **Robustezza vs Tempi di risposta:** Al fine di aumentare la robustezza del codice e prevenire l'iniezione di contenuti malevoli, verranno effettuati controlli rigorosi sugli input lato server prima dell'interazione con il modulo AI, accettando un lieve incremento nei tempi di elaborazione della richiesta.

- **Costi di sviluppo vs Prestazioni:** Al fine di ridurre i tempi di sviluppo e sfruttare le librerie di Intelligenza Artificiale esistenti, si è scelto l'utilizzo di un linguaggio interpretato (Python) e di framework web (Flask), accettando prestazioni di esecuzione inferiori rispetto a linguaggi compilati, ma garantendo una maggiore manutenibilità e rapidità di implementazione.

- **Accuratezza vs Tempi di risposta:** Al fine di garantire un'esperienza utente fluida (decisione entro 5 secondi) , il sistema utilizzerà modelli di AI ottimizzati per l'inferenza

rapida, accettando una soglia di accuratezza del 75% piuttosto che utilizzare modelli più complessi che richiederebbero tempi di calcolo insostenibili per un'interazione in tempo reale.

## 1.2. Off-the-shelf components

Il sistema comprende l'utilizzo di componenti off-the-shelf, che forniranno un supporto alla realizzazione del progetto. Tali componenti saranno:

- **Flask:** È un micro-framework web scritto in Python. Fornisce gli strumenti essenziali per la gestione delle richieste HTTP (routing), il rendering dei template e la gestione delle sessioni, fungendo da spina dorsale per il Backend.
- **MySQL Connector/Python:** È il driver standard che permette alle applicazioni Python di connettersi a un server di database MySQL, implementando le API di database Python (PEP 249).
- **SQLAlchemy:** (Opzionale, se si usa ORM) È un toolkit SQL e un sistema di mappatura relazionale a oggetti (ORM) per Python, che facilita la gestione dei dati persistenti astraendo le query SQL complesse.
- **Scikit-learn / Transformers (Hugging Face):** Sono librerie per l'apprendimento automatico in Python. Verranno utilizzate nel modulo AIService per il caricamento del modello, il preprocessing del testo e il calcolo dello score di attendibilità.
- **Jinja2:** È il motore di templating predefinito di Flask, utilizzato per generare dinamicamente le pagine HTML che costituiscono l'interfaccia utente (View).

## 1.3. Interface documentation guidelines

Le linee guida per la documentazione delle interfacce contengono le convenzioni che gli sviluppatori saranno tenuti a seguire durante la progettazione e lo sviluppo delle interfacce del sistema. In particolare, per le seguenti tecnologie verranno utilizzate le indicazioni riportate nei link sottostanti:

- **Python (PEP 8):** https://peps.python.org/pep-0008/
- **HTML e CSS:** https://google.github.io/styleguide/htmlcssguide.html
- **Docstring Python (PEP 257):** https://peps.python.org/pep-0257/ (per la documentazione del codice)

## 1.4. Definitions, acronyms, and abbreviations

**Definizioni:** vedi Glossario.

**Acronimi:**

- **ODD** = Object Design Document;
- **SDD** = System Design Document;
- **RAD** = Requirements Analysis Document;
- **HTML** = HyperText Markup Language;
- **CSS** = Cascading Style Sheet;
- **API** = Application Programming Interface;
- **REST** = Representational State Transfer;
- **SQL** = Structured Query Language;
- **ORM** = Object-Relational Mapping;
- **JSON** = JavaScript Object Notation;
- **AI** = Artificial Intelligence;
- **NLP** = Natural Language Processing.

## 1.5. References


- B. Bruegge, A.H. Dutoit, *Object Oriented Software Engineering - Using UML, Patterns and Java*, Prentice Hall.
- Documento di *Problem Statement* relativo a questo progetto.
- Documento di *Requirement Analysis Document* relativo a questo progetto.
- Documento di *System Design Document* relativo a questo progetto.
- Linee guida per lo stile:

- https://peps.python.org/pep-0008/
- https://google.github.io/styleguide/htmlcssguide.html

## 2. Packages

Il sistema proposto è sviluppato in linguaggio **Python** utilizzando il framework **Flask**.
L'organizzazione dei pacchetti (in Python definiti come *Package* contenenti moduli) segue rigorosamente il pattern architetturale **MVC (Model-View-Controller)** definito nel System Design Document.
A differenza dell'ambiente Java (basato su una struttura di classi compilate), in Python la struttura è organizzata in **moduli** e **Blueprints**. Il sistema è integrato come segue:

```
FakeBuster/
├── app/
│   ├── __init__.py             # Inizializzazione Flask e configurazione DB
│   │
│   ├── models/                 # PACKAGE MODEL (Tier 3 - Dati & Persistenza)
│   │   ├── __init__.py
│   │   ├── account.py          # Mappa la tabella 'Account' (User + FactChecker)
│   │   ├── post.py             # Mappa la tabella 'Post' (Dati e Score IA)
│   │   ├── appello.py          # Mappa la tabella 'Appello'
│   │   └── segnalazione.py     # Mappa la tabella 'Segnalazione'
│   │
│   ├── controllers/            # PACKAGE CONTROLLER (Tier 2 - Logica di Business)
│   │   ├── __init__.py
│   │   ├── gestione_utenza.py        # Sottosistema Gestione Utenza
│   │   ├── gestione_pubblicazioni.py # Sottosistema Gestione Pubblicazioni
│   │   ├── gestione_appelli.py       # Sottosistema Gestione Appelli
│   │   └── gestione_segnalazioni.py  # Sottosistema Gestione Segnalazioni
│   │
│   ├── services/               # SERVIZI DI SUPPORTO (Logica Esterna)
│   │   ├── __init__.py
│   │   └── ai_service.py       # Implementa il sottosistema Servizio IA
│   │
│   ├── templates/              # PACKAGE WEBCONTENT (Tier 1 - View HTML)
│   │   ├── guest/              # Interfacce pubbliche
│   │   ├── user/               # Interfacce utente loggato
│   │   └── fact_checker/       # Interfacce moderazione
│   │
│   └── static/                 # ASSET STATICI (Tier 1 - View CSS/JS)
│       ├── css/                # Fogli di stile
│       ├── js/                 # Script client-side
│       └── img/                # Immagini e loghi
│
└── run.py                      # Entry point del server
```

## 2.1. Package Model

Il compito del Package models è di implementare il livello di persistenza dei dati, interfacciandosi con il Database MySQL definito nello Schema Logico dell'SDD.
In coerenza con la scelta dell'ORM (Object-Relational Mapping), questo package non utilizza classi DAO separate (tipiche di altri linguaggi), ma definisce classi "Entity" che gestiscono sia la struttura dei dati che le operazioni di query estendendo la classe base dell'ORM (db.Model). Le classi contenute sono:

- **account.py:** Gestisce la persistenza degli utenti unificati (User e Fact-checker).
- **post.py:** Gestisce la persistenza delle notizie e dei relativi metadati IA.

- **appello.py:** Gestisce la persistenza dei ricorsi.
- **segnalazione.py:** Gestisce la persistenza dei report.

### 2.2. Package Control

Il Package controllers implementa la logica di controllo del pattern MVC. Contiene i moduli Python (implementati come *Blueprints* Flask) che corrispondono esattamente alla decomposizione in sottosistemi definita nel System Design Document.
Ogni modulo gestisce le richieste HTTP relative al proprio dominio funzionale:

- **gestione_utenza.py:** Gestisce le rotte per l'autenticazione (login/logout) e la gestione del profilo utente.
- **gestione_pubblicazioni.py:** Coordina il flusso di creazione del post, invocando il modulo ai_service per l'analisi e aggiornando lo stato nel Model.
- **gestione_appelli.py:** Gestisce la logica di business per la creazione (da parte dell'User) e la risoluzione (da parte del Fact-checker) degli appelli.
- **gestione_segnalazioni.py:** Gestisce la logica per l'invio e la moderazione delle segnalazioni.

### 2.3. Package WebContent

Il Package templates ha lo scopo di fungere da interfaccia grafica per l'Utente finale (View). Questo package contiene i file **HTML** necessari per l'interfaccia utente. Questi file utilizzano il motore di templating **Jinja2** per visualizzare dinamicamente i dati forniti dai Controller.
La struttura delle cartelle rispecchia i ruoli definiti nel RAD per garantire la segregazione delle interfacce:

- **guest:** Contiene le viste accessibili a tutti (es. index.html, login.html).
- **user:** Contiene le viste per l'utente loggato (es. feed.html, storico_post.html).
- **fact_checker:** Contiene le viste di amministrazione (es. dashboard.html).

Gli asset statici (fogli di stile CSS, script JavaScript) sono organizzati nella cartella parallela static per ottimizzare il caricamento lato client.

## 3. Class Interfaces

In questa sezione vengono descritte le interfacce delle classi principali del sistema, specificando per ciascuna: descrizione, pre-condizioni (stato necessario prima dell'esecuzione), post-condizioni (stato del sistema dopo l'esecuzione) e invarianti (condizioni che devono rimanere sempre vere)

## 3.1. Models

| **Nome classe** | **Account.py** |
|---|---|
| **Descrizione** | Rappresenta l'utente del sistema. Unifica i ruoli di User e Fact-checker gestendo le credenziali di accesso. |
| **Invarianti** | email deve essere univoca nel sistema.<br><br>username deve essere univoco.<br><br>ruolo deve assumere solo i valori {'User', 'FactChecker'}. |

| **Nome classe** | **Post.py** |
|---|---|
| **Descrizione** | Rappresenta la notizia sottomessa dall'utente. Memorizza il contenuto testuale, lo stato di pubblicazione corrente e i risultati dell'analisi IA (Score e Log). |
| **Invarianti** | autore_id non può essere nullo (ogni post deve avere un autore).<br><br>ai_score (se presente) deve essere un valore decimale compreso tra 0.0 e 1.0. |

| **Nome classe** | **Appello.py** |
|---|---|
| **Descrizione** | Rappresenta la richiesta di revisione manuale per un post bloccato. Collega l'autore |

| | |
|---|---|
| **Nome classe** | **Appello.py** |
| | del post al fact-checker che prende in carico la revisione. |
| **Invarianti** | Un post può avere un solo appello attivo in un dato momento (stato = 'Aperto'). |

| | |
|---|---|
| **Nome classe** | **Segnalazione.py** |
| **Descrizione** | Rappresenta la segnalazione inviata da un utente verso un contenuto pubblico ritenuto falso o inappropriato. |
| **Invarianti** | Un singolo utente non può segnalare lo stesso post più di una volta (univocità della coppia post_id, segnalatore_id). |

### 3.2. Services

| | |
|---|---|
| **Nome classe** | **ai_service.py** |
| **Descrizione** | Modulo di servizio che gestisce l'interazione con il motore di Intelligenza Artificiale per l'analisi semantica del testo. |
| **Pre-condizione** | context: analizza_testo(testo: str)<br><br>**pre:** testo non deve essere nullo o vuoto.<br><br>**pre:** Il modello AI (o l'API esterna) deve essere accessibile. |
| **Post-condizione** | context: analizza_testo(testo: str)<br><br>**post:** Restituisce un oggetto risultato contenente: score (float), timestamp dell'analisi e log_features (dettagli tecnici). |

## 3.3. Controllers

**Modulo: gestione_utenza.py**

| | |
|---|---|
| **Nome Funzione** | **login (Route: /login)** |
| **Descrizione** | Gestisce il processo di autenticazione dell'utente e l'inizializzazione della sessione. |
| **Pre-condizione** | **pre:** request.method == 'POST'<br><br>**pre:** I campi Email e Password sono presenti nella richiesta. |
| **Post-condizione** | **post:** current_user.is_authenticated è True.<br><br>**post:** L'utente viene reindirizzato alla vista corretta in base al ruolo (feed per User, dashboard per FactChecker). |

| | |
|---|---|
| **Nome Funzione** | **registrazione (Route: /registrazione)** |
| **Descrizione** | Gestisce la creazione di un nuovo account utente nel sistema. |
| **Pre-condizione** | **pre:** request.method == 'POST'<br><br>**pre:** L'Email e l'Username forniti non esistono già nel database. |
| **Post-condizione** | **post:** Un nuovo record viene inserito nella tabella Account.<br><br>**post:** La password viene salvata in formato hash.<br><br>**post:** L'utente viene loggato automaticamente. |

**Modulo: gestione_pubblicazioni.py**

| | |
|---|---|
| **Nome Funzione** | **crea_post (Route: /post/new)** |
| **Descrizione** | Gestisce il flusso di sottomissione di una notizia, invocando l'analisi IA e determinando lo stato iniziale (Pubblicato o Bloccato). |
| **Pre-condizione** | **pre:** L'utente corrente è autenticato e ha ruolo 'User'.<br><br>**pre:** Il testo del post non è vuoto. |
| **Post-condizione** | **post:** Il metodo AIService.analizza_testo() è stato eseguito.<br><br>**post:** Un nuovo record Post è creato nel DB.<br><br>**post:** Se score >= soglia (0.7), Post.stato è impostato a 'Pubblicato'.<br><br>**post:** Se score < soglia, Post.stato è impostato a 'Bloccato'. |

**Modulo: gestione_appelli.py**

| **Nome Funzione** | **crea_appello (Route: /post/<id>/appello)** |
|---|---|
| **Descrizione** | Permette all'autore di un post bloccato di richiedere una revisione manuale. |
| **Pre-condizione** | **pre:** Il post specificato esiste ed è in stato 'Bloccato'.<br><br>**pre:** Non esistono altri appelli aperti per questo post. |
| **Post-condizione** | **post:** Un nuovo record Appello è creato con stato 'Aperto'.<br><br>**post:** Lo stato del Post viene aggiornato a 'In Revisione'. |

| **Nome Funzione** | **risolvi_appello (Route: /fact-checker/appello/<id>)** |
|---|---|
| **Descrizione** | Permette al Fact-checker di approvare o respingere un appello, finalizzando lo stato del post. |
| **Pre-condizione** | **pre:** L'utente corrente ha ruolo 'FactChecker'.<br><br>**pre:** L'appello specificato è in stato 'Aperto'. |
| **Post-condizione** | **post:** Appello.stato viene aggiornato a 'Chiuso'.<br><br>**post:** Se la decisione è 'Pubblica', Post.stato diventa 'Pubblicato'.<br><br>**post:** Se la decisione è 'Blocca', Post.stato torna a 'Bloccato' (o 'Rimosso'). |

**Modulo: gestione_segnalazioni.py**

| **Nome Funzione** | **crea_segnalazione (Route: /post/<id>/segnala)** |
|---|---|
| **Descrizione** | Permette a un utente di segnalare un post pubblico ritenuto sospetto. |
| **Pre-condizione** | **pre:** Il post specificato è in stato 'Pubblicato'.<br><br>**pre:** L'utente corrente non ha già segnalato questo post. |
| **Post-condizione** | **post:** Un nuovo record Segnalazione viene inserito nel database. |

| **Nome Funzione** | **risolvi_segnalazione (Route: /fact-checker/segnalazione/<id>)** |
|---|---|
| **Descrizione** | Permette al Fact-checker di gestire una segnalazione decidendo se rimuovere il post o ignorare la segnalazione. |
| **Pre-condizione** | **pre:** L'utente corrente ha ruolo 'FactChecker'. |
| **Post-condizione** | **post:** Segnalazione.stato viene aggiornato a 'Chiuso'.<br><br>**post:** Se l'esito è 'Rimuovi', Post.stato viene impostato su 'Rimosso'. |

## 4. Glossary

| **TERMINE** | **SIGNIFICATO** |
|---|---|
| **Object Design Document (ODD)** | Documento che si occupa di aggiungere dettagli all'analisi dei requisiti e prendere decisioni di implementazione a livello di oggetti. |
| **Design Goal** | Obiettivi di design progettati per il sistema proposto (es. Performance, Affidabilità). |

| TERMINE | SIGNIFICATO |
|---|---|
| **Trade-off** | Scelte e compromessi tra design goals dissonanti (es. Accuratezza vs Velocità). |
| **Sottosistema** | Un sottoinsieme dei servizi del dominio applicativo, formato da servizi legati da una relazione funzionale. |
| **Front-end** | La parte visibile all'utente di un programma e con cui egli può interagire (HTML/CSS). |
| **Back-end** | La parte che si occupa di gestire il funzionamento del sistema, la logica applicativa e la comunicazione con il Database e il servizio AI. |
| **Database** | Architettura esterna che si occupa della gestione e della memorizzazione dei dati persistenti (MySQL). |
| **Persistenza** | Caratteristica dei dati di un programma di sopravvivere all'esecuzione del programma stesso che li ha creati, salvando i dati in uno storage non volatile. |
| **Framework** | Un'architettura logica di supporto che fornisce strumenti per facilitare e velocizzare il lavoro di programmazione (es. **Flask** per il backend). |
| **Client** | Componente (Browser Web) che accede a servizi e risorse di un altro componente detto Server. |
| **Server** | Componente che gestisce traffico di informazioni e fornisce servizi e risorse attraverso la rete. |
| **Model** | Componente MVC che contiene i metodi di accesso ai dati e rappresenta le entità del dominio (es. classi SQLAlchemy). |
| **View** | Componente MVC che si occupa di visualizzare i dati all'Utente e gestisce l'interazione fra quest'ultimo e l'infrastruttura (Template HTML). |
| **Controller** | Componente MVC che riceve i comandi dell'Utente e reagisce eseguendo delle operazioni che possono interessare il Model e che portano generalmente ad un cambiamento di stato della View. |
| **Route / View Function** | Funzioni Python all'interno del framework Flask che gestiscono le richieste HTTP a specifici URL (equivalente funzionale delle Servlet in Java). |

| TERMINE | SIGNIFICATO |
| --- | --- |
| **Jinja2 Template** | Motore di templating utilizzato da Flask per generare contenuti HTML dinamici da mostrare all'utente (equivalente funzionale delle JSP). |
| **Package / Modulo** | Meccanismo per organizzare classi e funzioni Python in gruppi logici, principalmente allo scopo di definire spazi dei nomi distinti per diversi contesti. |
| **Design Pattern** | Una descrizione o modello logico da applicare per la risoluzione di un problema che può presentarsi in diverse situazioni durante le fasi di progettazione e sviluppo. |
| **ORM (Object-Relational Mapping)** | Tecnica di programmazione che converte i dati tra sistemi di tipi incompatibili (oggetti Python e tabelle SQL), gestita in questo progetto da librerie come SQLAlchemy. |
| **AI Service** | Il modulo software esterno incaricato di analizzare semanticamente il testo delle notizie e produrre una valutazione (Score). |