



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Informatica

Ingegneria del Software

Anno Accademico: 2023/2024

JACKPOT CODING

Gruppo: Jackpot Coding
Email: jackpotcoding@gmail.com

SPECIFICA TECNICA

DESTINATARI: Prof. T. Vardanega, Prof. R. Cardin

USO: ESTERNO
VERSIONE: 0.0.15

Registro delle modifiche

Versione	Data	Autore	Verificatore	Modifica
0.1.1	03/05/2024	E. Gallo	-	Sistemato <i>Design pattern</i> utilizzati - <i>Strategy</i>
0.1.0	03/05/2024	-	E. Gallo	Verifica documento
0.0.15	28/04/2024	G. Moretto	E. Gallo	Aggiunta di SonarQube come strumento di testing
0.0.14	25/04/2024	R. Simionato	G. Moretto	Aggiunta dei riferimenti al Glossario
0.0.13	23/04/2024	G. Moretto	R. Simionato	Aggiunta QueryForm, QueryGenerationView, QueryGenerator, sostituito Tensorflow con Torch
0.0.12	21/04/2024	G. Moretto	R. Simionato	Aggiornamento diagramma Views con nuova operazione per PromptCreator
0.0.11	18/04/2024	G. Moretto	R. Simionato	Aggiunta dei digrammi delle classi
0.0.10	10/04/2024	G. Moretto	R. Simionato	Aggiunta dello strumento di testing coveralls.io
0.0.9	09/04/2024	-	G. Moretto	Documento rinominato in "Specifica Tecnica"
0.0.8	09/04/2024	E. Gallo	G. Moretto	Aggiunti riferimenti normativi ed informativi
0.0.7	06/04/2024	M. Favaretto	G. Moretto	Aggiunta sezione <i>Design pattern</i> utilizzati - <i>M.V.T.</i>
0.0.6	05/04/2024	E. Gallo	M. Favaretto	Aggiunta sezione Design pattern utilizzati - Strategy
0.0.5	04/04/2024	M. Camillo	E. Gallo	Introduzione Documento
0.0.4	04/04/2024	M. Gobbo	E. Gallo	Introduzione all'architettura
0.0.3	03/04/2024	G. Moretto	M. Gobbo	Aggiunto elenco delle tabelle
0.0.2	02/04/2024	G. Moretto	M. Gobbo	Aggiunta tabelle tecnologie codifica e testing
0.0.1	27/03/2024	G. Moretto	M. Gobbo	Aggiunta struttura documento

Indice

1	Introduzione	6
1.1	Scopo del Documento	6
1.2	Scopo del Prodotto	6
1.3	Glossario	6
1.4	Riferimenti	6
1.4.1	Riferimenti normativi	6
1.4.2	Riferimenti informativi	6
2	Tecnologie	6
2.1	Codifica	6
2.2	Testing	7
3	Architettura	8
3.1	Introduzione	8
3.2	Diagrammi delle classi	9
3.2.1	Modelli	9
3.2.2	View	11
3.2.3	Form	13
3.3	Design pattern utilizzati	14
3.3.1	Model-View-Template	14
3.3.2	Strategy	14

Elenco delle immagini

1	Diagramma UML delle classi Model	9
2	Diagramma UML delle classi View	11
3	Diagramma UML delle classi Form	13

Elenco delle tabelle

1	Tecnologie di codifica.	7
2	Tecnologie di testing.	7

1 Introduzione

1.1 Scopo del Documento

Il documento ha lo scopo di presentare e motivare le scelte architetturelle e di design applicate al prodotto, oltre che a fornire una lista completa delle tecnologie utilizzate. La struttura interna del prodotto è esposta all'interno del documento sotto forma di diagramma delle classi, in maniera da rendere più chiaro il software^G sviluppato. Vengono inoltre approfonditi e motivati a loro volta i design pattern^G utilizzati.

1.2 Scopo del Prodotto

Il capitolato^G proposto dall'azienda *Zucchetti S.p.A.* ha come obiettivo la realizzazione di un applicativo web al fine di studiare la fattibilità di un prodotto in grado di elaborare una frase in linguaggio naturale^G. Questa frase, anche se fornita da un utente inesperto, deve generare come output una *query SQL*^G in grado di interrogare un database^G (di cui è conosciuta la struttura) in modo efficiente e affidabile.

1.3 Glossario

Al fine di evitare ambiguità o incomprensioni relative alla terminologia usata all'interno del documento, è fornito un *Glossario* in cui vengono riportate definizioni precise per ogni termine potenzialmente ambiguo. La presenza di tali termini all'interno del documento è indicata con la presenza, vicino alla voce, di una *G* in apice (^G).

1.4 Riferimenti

1.4.1 Riferimenti normativi

- Capitolato^G C9 - *ChatSQL*
<https://www.math.unipd.it/~tullio/IS-1/2023/Progetto/C9.pdf>
- Norme di progetto^G V1.0.2
- Glossario V1.0.0
https://jackpot-coding.github.io/chatSQL/docs/esterni/glossario_v1.0.0.pdf

1.4.2 Riferimenti informativi

Slide del corso Ingegneria del Software:

- Progettazione software
<https://www.math.unipd.it/~tullio/IS-1/2023/Dispense/T6.pdf>
- Diagramma delle classi
<https://www.math.unipd.it/~rcardin/swea/2023/Diagrammi%20delle%20Classi.pdf>
- Pattern MVC e derivati
<https://www.math.unipd.it/~rcardin/sweb/2022/L02.pdf>
- SOLID programming
https://www.math.unipd.it/~rcardin/swea/2021/SOLID%20Principles%20of%20Object-Oriented%20Design_4x4.pdf
- Pattern comportamentali
https://www.math.unipd.it/~rcardin/swea/2021/Design%20Pattern%20Comportamentali_4x4.pdf

2 Tecnologie

2.1 Codifica

Tabella 1: Tecnologie di codifica.

Tecnologia	Versione	Descrizione
Linguaggi		
HTML	5	Linguaggio di <i>markup</i> utilizzato per la definizione della struttura di pagine <i>web</i>
CSS	3	Linguaggio utilizzato per applicare stile a elementi presenti in una pagina <i>HTML</i>
Python	3.11.8	Linguaggio di programmazione ad alto livello, orientato agli oggetti. Viene utilizzato per la creazione del <i>server</i> .
Framework^G e Librerie^G		
Django	5.0.3	<i>Framework</i> per la creazione di applicazioni <i>web</i> scritto in linguaggio <i>Python</i> .
Torch	2.2.2	Libreria <i>Python</i> per l'apprendimento automatico.
Transformers	4.29.3	Libreria <i>Python</i> per l'utilizzo di modelli del portale <i>Hugging Face</i> utilizzando <i>Torch</i>
Strumenti		
Pip	24.0	Strumento per la gestione dei pacchetti utilizzati da applicazioni <i>Python</i> .
Git	2.44.0	Strumento per il controllo di versione utilizzato per la gestione della <i>repository</i> ^G remota presente su <i>GitHub</i> ^G .

2.2 Testing

Tabella 2: Tecnologie di testing.

Tecnologia	Versione	Descrizione
Framework e Librerie		
Unittest	3.11.8	<i>Framework</i> incluso nel linguaggio <i>Python</i> utilizzato per il <i>testing</i> di unità, utilizzato dal <i>framework Django</i> .
Django Test Client	5.0.3	<i>Client</i> per il <i>testing</i> di un applicazione <i>web</i> simulando un <i>browser</i> , integrato nel <i>framework Django</i> .
coverage.py	7.4.4	<i>Tool</i> per misurare il <i>code coverage</i> in applicazioni <i>Python</i> integrabile nel <i>framework Django</i> .
Prospector	1.10.3	<i>Tool</i> per l'analisi statica di codice scritto nel linguaggio <i>Python</i> .
SonarQube	10.5	<i>Tool</i> per l'analisi automatica della qualità del codice in vari linguaggi e <i>framework</i> ^G .
Strumenti		
GitHub Actions	-	Servizio di <i>Github</i> per la <i>Continuous Integration</i> , automatizza i processi di <i>build</i> , <i>test</i> e <i>deploy</i> del prodotto <i>software</i> .
Coveralls.io	-	Servizio per la visualizzazione dei rapporti di <i>code coverage</i> e applicazione di un <i>badge</i> alla <i>repository</i> .

3 Architettura

3.1 Introduzione

L'architettura di "Chat SQL" si basa sul *design pattern* architetturale *MVT*(*Model View Template*).

In questo *pattern*, il "Model" rappresenta i dati e la logica di *business* dell'applicazione, la "View" esegue la *business logic*, interagisce con il *Model* e ritorna risposte Http, infine il "Template" definisce la struttura dell'interfaccia utente e come i dati vengono visualizzati al suo interno. Più nello specifico il "Template", nel caso della nostra applicazione, definisce la struttura di documenti *HTML*.

Questo *design pattern* è simile a *MVC*(*Model View Controller*)

E' stato scelta questa architettura in quanto offre i seguenti vantaggi:

- **Separazione delle responsabilità:** *MVT* separa chiaramente le responsabilità tra *Model*, *View* e *Template*. Questo permette una migliore organizzazione del codice, facilitando la manutenzione e la scalabilità dell'applicazione.
- **Riutilizzo dei template:** I *template* in *MVT* consentono di separare la presentazione dalla logica di *business*. Ciò facilita il riutilizzo dei componenti di interfaccia utente in diverse parti dell'applicazione, riducendo la duplicazione del codice e migliorando l'efficienza dello sviluppo.
- **Aumento delle Performance:** Utilizzare i *template* pre-renderizzati può migliorare le prestazioni dell'applicazione rispetto a un'architettura *MVC* tradizionale, poiché il *rendering* dei *template* può essere più efficiente rispetto ad esempio alla generazione dinamica di *HTML* lato *server*.

3.2 Diagrammi delle classi

3.2.1 Modelli

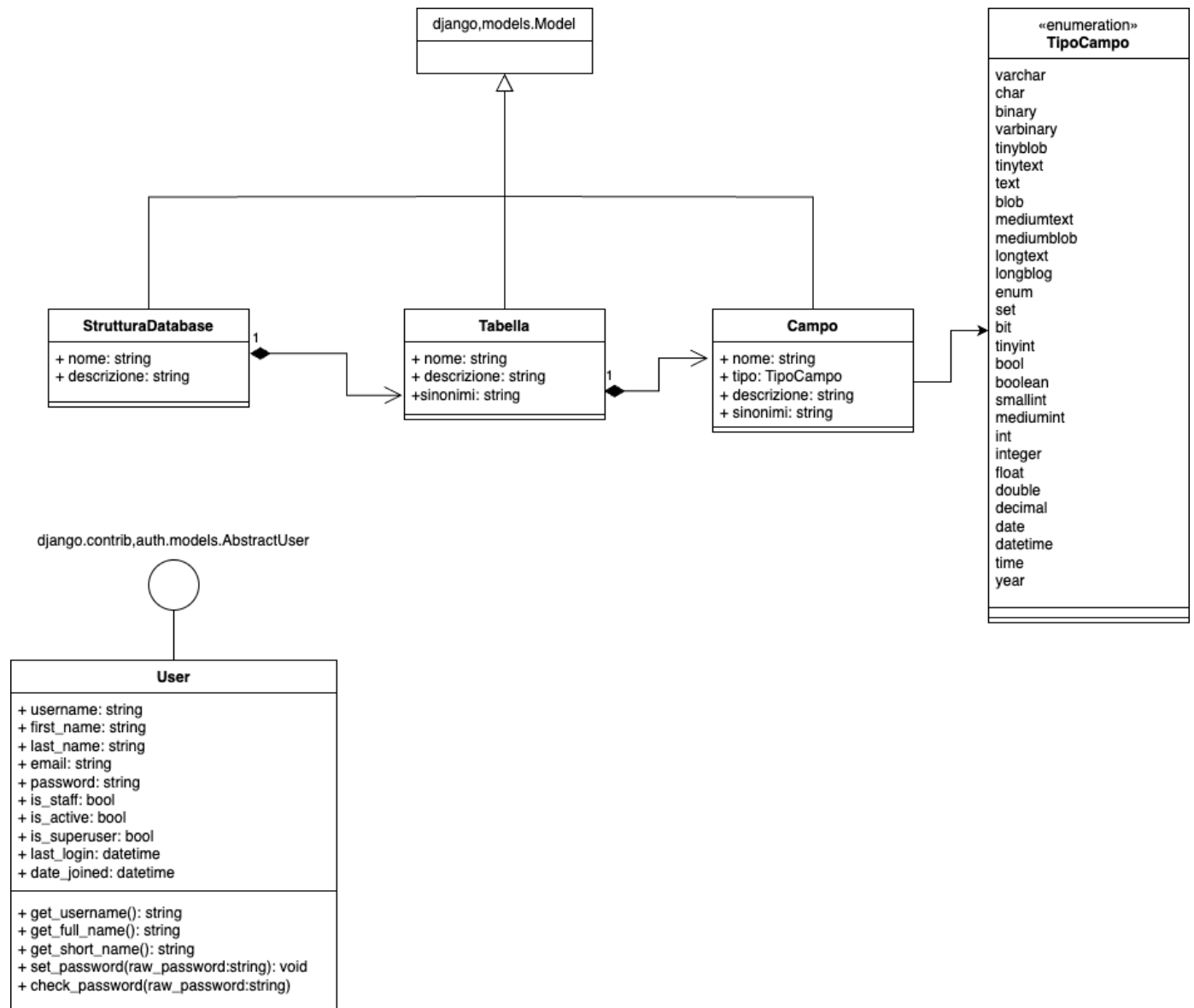


Figura 1: Diagramma UML delle classi Model

Il compito del modello è quello di gestire i dati che vengono utilizzati dall'applicazione. Formano la struttura dei dati dell'intera applicazione e sono rappresentati in un database.

I modelli definiti derivano dalla classe Model fornita dal framework Django e sono:

- **StrutturaDatabase**: rappresenta le varie strutture database definite dall'amministratore;
- **Tabella**: rappresenta le tabelle che compongono una Struttura Database;
- **Campo**: rappresenta i campi che compongono una Tabella. Questo utilizza un'enumerazione chiamata `TipoCampo` che indica le tipologie di campo selezionabili;

Viene inoltre utilizzato il modello User fornito dal framework Django del quale si elencano gli attributi e operazioni più rilevanti.

3.2.2 View

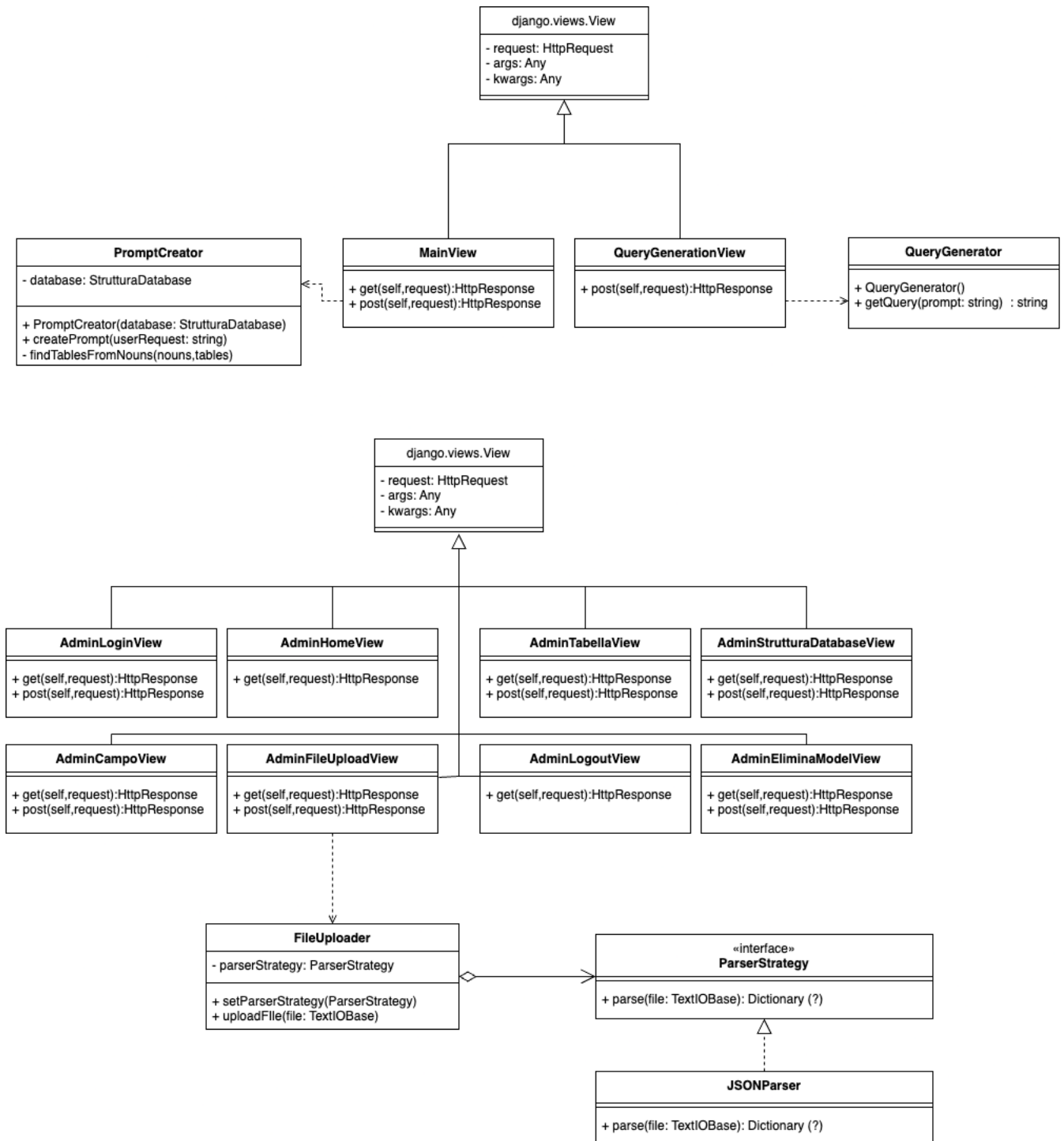


Figura 2: Diagramma UML delle classi View

Il compito delle view è quella di ricevere richieste dal browser^G e di restituire risposte sottoforma di pagine HTML o altri elementi che possono essere visualizzati da un browser.

Le view definite derivano dalla classe View fornita dal framework Django e sono divise per area principale e area amministrativa.

Le view dell'area principale sono:

- **MainView**: si occupa di ricevere la richiesta di generazione del prompt^G dall'utente e di restituirlo. Per fare questo utilizza una classe chiamata PromptCreator che si occupa di generare il prompt per la richiesta ricevuta dalla view.
- **QueryGenerationView**: si occupa di ricevere il prompt generato, inviarlo al modello LLM^G esterno tramite la classe Querygenerator e ritornare la risposta;

Le view dell'area amministrativa sono:

- **AdminLoginView**: responsabile per l'autenticazione dell'amministratore;
- **AdminLogoutView**: responsabile per il logout dell'amministratore;
- **AdminHomeView**: restituisce la pagina home dell'area amministrativa dove è presente la lista delle Strutture Database e il link per la creazione delle stesse;
- **AdminStrutturaDatabaseView**: restituisce la pagina di creazione e modifica della StrutturaDatabase e la lista delle tabelle che la compongono;
- **AdminTabellaView**: restituisce la pagina di creazione e modifica delle Tabelle e la lista dei campi che la compongono;
- **AdminCampoView**: restituisce la pagina di creazione e modifica dei Campi di una tabella;
- **AdminFileUploadView**: restituisce la pagina per il caricamento di un file di struttura database e utilizza la classe FileUploader per la conversione ed il salvataggio come StrutturaDatabase;
- **AdminEliminaModelView**: responsabile per l'eliminazione di un oggetto dal database dato il suo id ed il suo modello. Restituisce una pagina per la conferma dell'eliminazione e l'eliminazione stessa;

3.2.3 Form

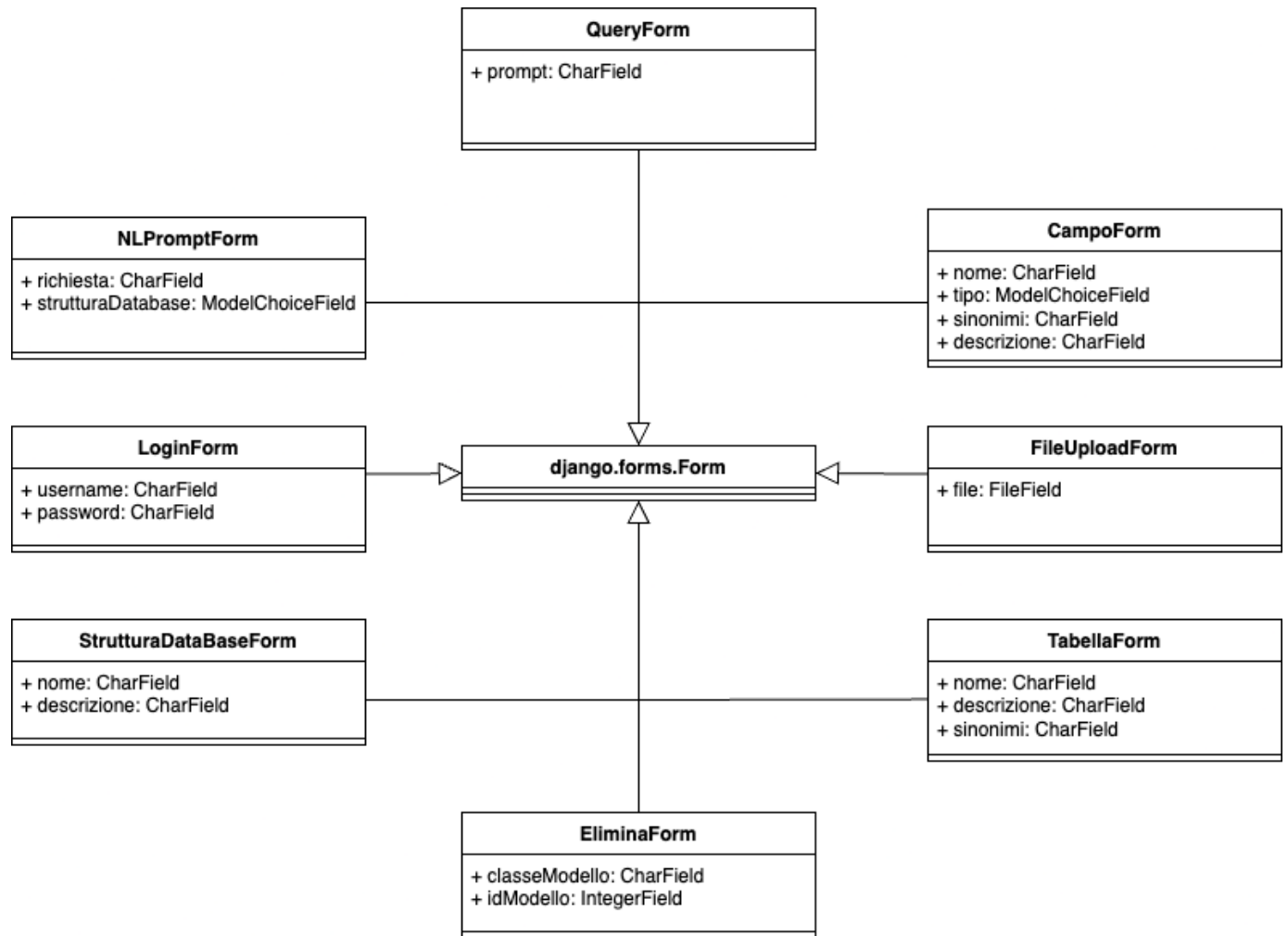


Figura 3: Diagramma UML delle classi Form

Il compito delle classi form è quella di definire form HTML sottoforma di classe. Questo per utilizzare le funzioni di validazione e gestione fornite dal framework Django tramite la classe Form dalla quale derivano i form da noi definiti:

- **NLPromptForm**: raccoglie i dati per la generazione del prompt per la generazione della query SQL;
- **QueryForm**: raccoglie i dati per la generazione della query SQL;
- **LoginForm**: raccoglie i dati per il login dell'amministratore;
- **StrutturaDatabaseForm**: raccoglie i dati per la creazione e modifica di una Struttura Database;
- **TabellaForm**: raccoglie i dati per la creazione e modifica di una Tabella;
- **CampoForm**: raccoglie i dati per la creazione e modifica di un Campo;
- **FileUploadForm**: raccoglie i dati per il caricamento del file di struttura database;
- **EliminaForm**: raccoglie i dati per l'eliminazione di un oggetto dal database;

3.3 Design pattern utilizzati

3.3.1 Model-View-Template

Per lo sviluppo del prodotto, il gruppo ha scelto l'utilizzo del *framework Django*. Il *framework* propone un'architettura integrata, basata su una generalizzazione della *view*, attraverso il *design pattern MVT*.

L'architettura proposta da *Django* si compone di:

- File per la gestione dei Modelli ;
- File per la gestione delle *View*;
- File per le impostazioni del progetto;
- File per la configurazione degli URL;
- File di *template* per la definizione dell'interfaccia utente;
- File per la gestione dei *form*;
- File per la gestione dei test;
- Cartella contenente le migrazioni verso il sistema di Database scelto;
- Cartella contenente file statici come fogli di stile e immagini;
- Cartella per ogni applicazione che compone il prodotto;

3.3.2 Strategy

Uno dei *design pattern* comportamentali scelto dal gruppo è lo *Strategy*^G. In particolare, viene integrato per l'interrogazione e l'integrazione di diversi *LLM*^G. Il prodotto infatti deve lavorare a stretto contatto con questi modelli linguistici di grandi dimensioni, ma non sempre usarne uno solo è vantaggioso.

Spesso i modelli più capaci sono a pagamento e di notevoli dimensioni, vi è quindi la necessità di interrogare modelli di dimensioni ridotte ma specializzati in una singola funzione.

Inoltre, l'interrogazione di diversi *LLM* permette un confronto dei diversi risultati, fornendo un'interessante introspezione sulle capacità, sulle possibilità e sulle differenze dei modelli.

Nel nostro caso sono due i tipi di interrogazioni agli *LLM* che il gruppo ha individuato.

1. L'interrogazione principale, su cui si basa l'intero progetto: da un *prompt*^G restituire una *query SQL*^G. Questo per valutare le performance di diversi modelli nella generazione di una *query SQL* corretta. Risulta quindi vantaggioso poter cambiare l'*LLM* interrogato a seconda della complessità della richiesta e della volontà dell'utente.
2. Le interrogazioni ausiliarie, che non formano un requisito obbligatorio ma servono per facilitare l'uso del prodotto all'utente. Un esempio è la generazione e l'inserimento dei sinonimi di tabelle e campi, che di base è lasciata all'utente, ma è facilmente integrabile con un *API*^G o un *LLM* opportuno per migliorare l'esperienza del prodotto.

Per questioni di tempo soprattutto, il gruppo ha deciso di implementare questo *design pattern* solo per la funzionalità di *upload* di un file. In base all'estensione del file, il programma sceglie quale strategia utilizzare per raccogliere ed immagazzinare nel migliore dei modi le informazioni caricate.