Università degli Studi di Padova

Informatica

Ingegneria del Software

Anno Accademico: 2023/2024

Gruppo: Jackpot Coding
Email: jackpotcoding@gmail.com

SPECIFICA TECNICA

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

USO: ESTERNO
VERSIONE: 1.0.0

Registro delle modifiche

Versione	Data	Autore	Verificatore	Modifica
1.0.0	08/05/2024	-	E. Gallo	Verifica documento
0.1.3	08/05/2024	G. Moretto	E. Gallo	Aggiunta indicazione termini presenti nel Glossario
0.1.2	06/05/2024	G. Moretto	E. Gallo	Aggiunta CSVParser e miglioramento descrizioni delle classi
0.1.1	03/05/2024	E. Gallo	G. Moretto	Sistemato *Design pattern* utilizzati - *Strategy*
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 *Design pattern* utilizzati - *M.V.T.*
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

Elenco delle immagini

Elenco delle tabelle

1 Introduzione

1.1 Scopo del Documento

Il documento ha lo scopo di presentare e motivare le scelte architetturali 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 che possa elaborare una frase in linguaggio naturale[G]. Questa frase, anche se fornita da un utente[G] 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[G]	5	Linguaggio di *markup* utilizzato per la definizione della struttura di pagine *web*
CSS[G]	3	Linguaggio utilizzato per applicare stile a elementi presenti in una pagina HTML
Python	3.11.8	Linguaggio di programmazione ad alto livello, orientato agli oggetti. Viene utilizzato per la creazione del *server*.
Framework[G] e Librerie[G]		
Django	5.0.3	*Framework* per la creazione di applicazioni *web*, scritto in linguaggio *Python*.
Torch	2.2.2	Libreria *Python*[G] per l'apprendimento automatico.
Transformers	4.29.3	Libreria *Python*[G] per l'utilizzo di modelli del portale *Hugging Face* utilizzando *Torch*
Strumenti		
Pip	24.0	Strumento per la gestione dei pacchetti utilizzati da applicazioni *Python*[G].
Git	2.44.0	Strumento per il controllo di versione utilizzato per la gestione della *repository*[G] remota presente su *GitHub*[G].

2.2 Testing

Tabella 2: Tecnologie di testing.

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

3 Architettura

3.1 Introduzione

L'architettura di *"Chat SQL"* si basa sul *design pattern*[G] 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[G].

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

E' stata scelta questa architettura sia perché dettata dal *framework* sia 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[G] lato *server*.

3.2 Diagrammi delle classi

3.2.1 Modelli

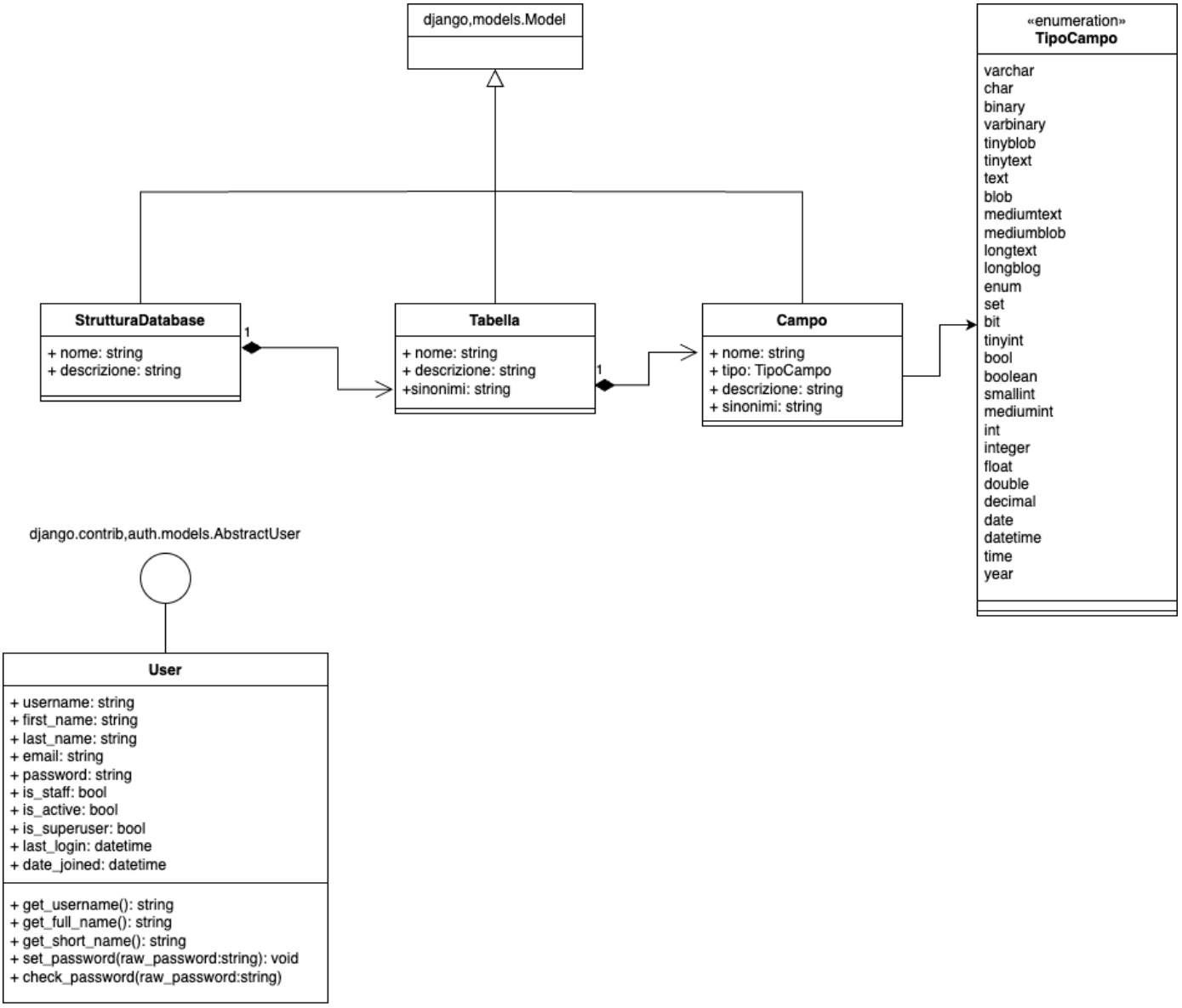

Figura 1: Diagramma UML delle classi Model

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*[G].

I modelli definiti derivano dalla classe Model fornita dal *framework*[G] Django e sono:

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

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

3.2.2 View

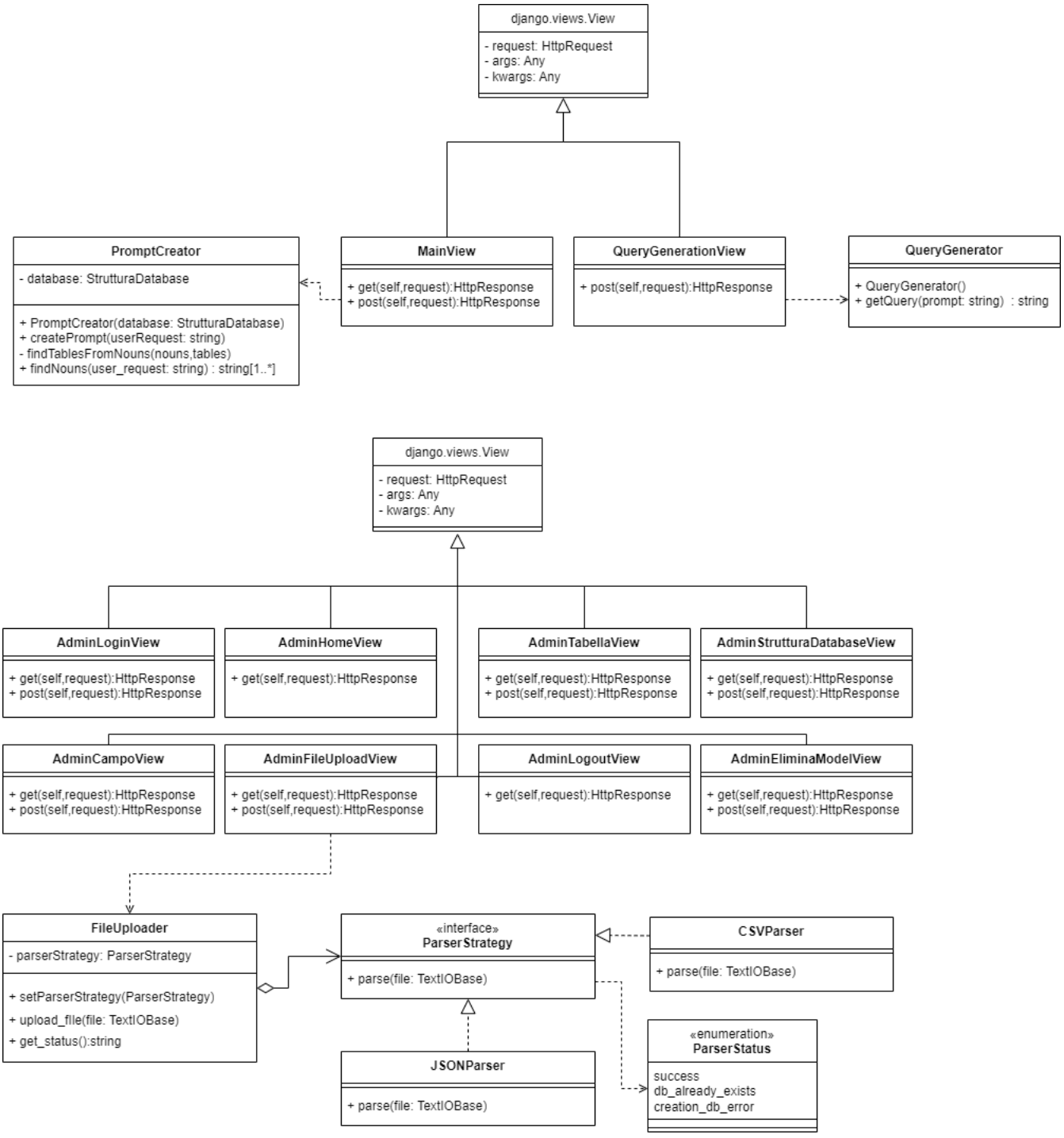

Figura 2: Diagramma UML delle classi View

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

Le *view* definite derivano dalla classe *View* fornita dal *framework*[G] 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[G] e di restituirlo. Per fare questo utilizza una classe chiamata PromptCreator che si occupa di generare il prompt per la richiesta ricevuta dalla view;
- **PromptCreator**: si occupa di creare il *prompt*[G] a partire dalla struttura *database*[G] e dalla frase in linguaggio naturale[G] fornita;
- **QueryGenerationView**: si occupa di ricevere il *prompt*[G] generato, inviarlo al modello *LLM*[G] esterno tramite la classe Querygenerator e ritornare la risposta;
- **QueryGenerator**: si occupa di generare la *query* SQL[G] richiesta inviando il *prompt*[G] generato ad un servizio *LLM*[G] esterno.

Le view dell'area amministrativa sono:

- **AdminLoginView**: responsabile per l'autenticazione dell'amministratore[G];
- **AdminLogoutView**: responsabile per il *logout* dell'amministratore[G];
- **AdminHomeView**: resituisce 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[G];
- **AdminFileUploadView**: restituisce la pagina per il caricamento di un *file*[G] di struttura *database*[G] (con eventuale messaggio di errore o successo a seconda dell'esito del caricamento) e utilizza la classe FileUploader per la conversione ed il salvataggio come StrutturaDatabase;
- **FileUploader**: responsabile per il caricamento di file che rappresentano una struttura *database*[G], utilizza l'interfaccia **ParserStrategy** che viene implementata con le classi:
  - **JSONParser**: per importare *file*[G] JSON con struttura nota;
  - **CSVParser**: per importare *file* CSV con struttura nota;

  Ognuno di questi *parser* restituisce, a seconda dell'esito, un messaggio di errore o successo contenuto nell'enumerazione **ParserStatus**;
- **AdminEliminaModelView**: responsabile per l'eliminazione di un oggetto dal *database*[G] dato il suo id ed il suo modello. Restituisce una pagina per la conferma dell'eliminazione e l'eliminazione stessa.

3.2.3 Form

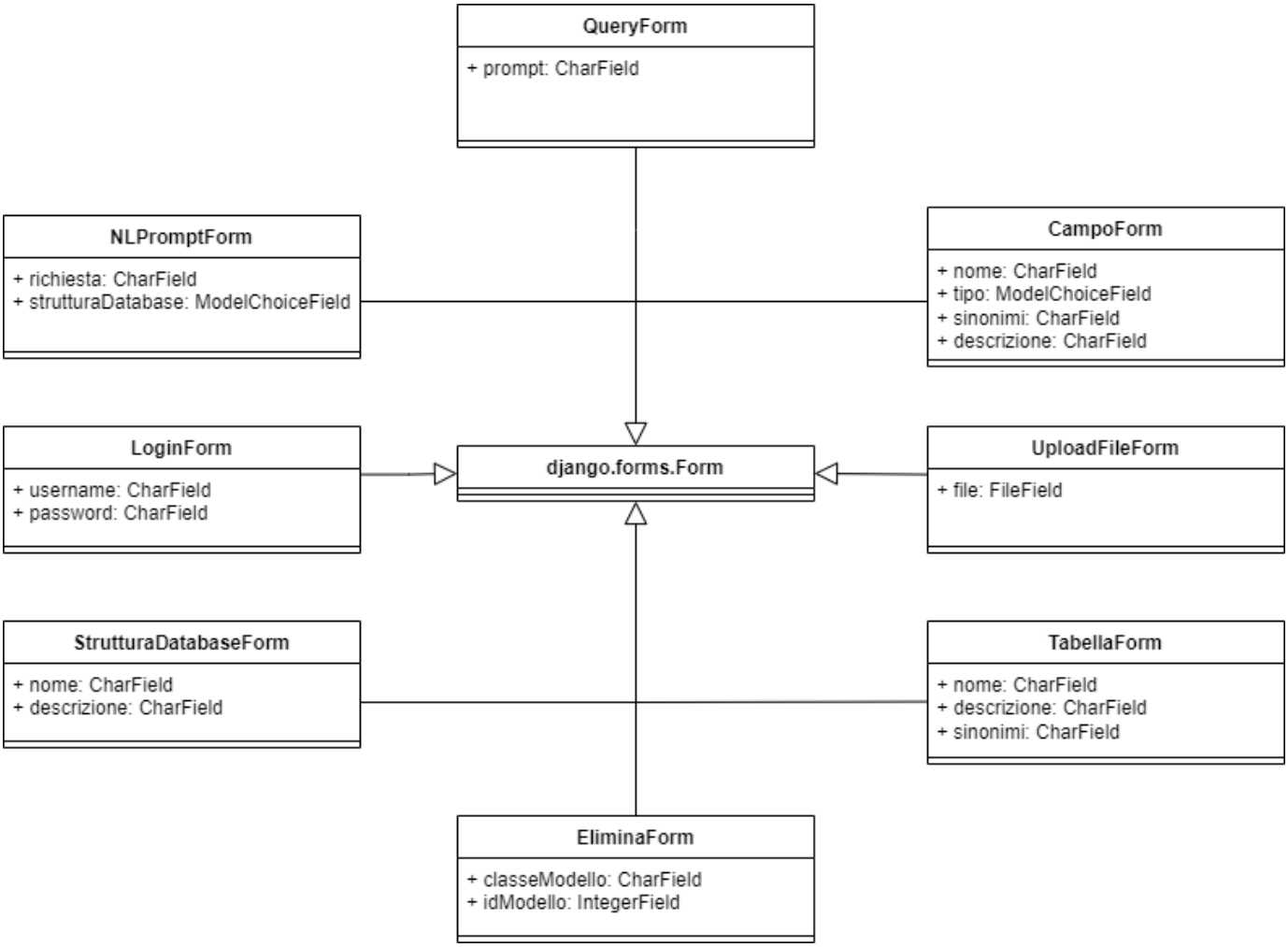

Figura 3: Diagramma UML delle classi Form

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

- **NLPromptForm**: raccoglie i dati per la generazione del *prompt*[G], da sottoporre poi ad un *LLM*[G];
- **QueryForm**: raccoglie i dati per la generazione della *query* SQL[G];
- **LoginForm**: raccoglie i dati per il *login*[G] dell'amministratore[G];
- **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[G];
- **CampoForm**: raccoglie i dati per la creazione e modifica di un Campo;
- **UploadFileForm**: raccoglie i dati per il caricamento del *file*[G] di struttura *database*[G];
- **EliminaForm**: raccoglie i dati per l'eliminazione di un oggetto dal *database*[G];

3.3 Design pattern utilizzati

3.3.1 Model-View-Template

Per lo sviluppo del prodotto, il gruppo ha scelto l'utilizzo del *framework*[G] Django. Il *framework* propone un'architettura integrata, basata su una generalizzazione della *view*, attraverso il *design pattern*[G] MVT. L'architettura proposta da *Django* si compone di:

- *File*[G] 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*[G] 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*[G] comportamentali scelto dal gruppo è lo *Strategy*[G]. In particolare, viene integrato per permettere al programma di riconoscere diversi tipi di *file*[G] quando viene eseguito *l'upload*. In base all'estensione del *file*, il programma sceglie quale strategia utilizzare per raccogliere ed immagazzinare nel migliore dei modi le informazioni caricate. Al lato pratico, una volta caricato un *file* tramite la classe *FileUploader*, questo viene passato ad un'interfaccia che verrà successivamente implementata dalle classi:

- *JSONParser*: per i *file* di tipo JSON
- *CSVParser*: per i *file* CSV

Il gruppo ha poi individuato altri ambiti in cui un *Design Pattern*[G] *Strategy* sarebbe ottimale. Per una questione di tempo principalmente, non sono stati implementati. Siccome il prodotto deve lavorare a stretto contatto con gli *LLM*[G], sarebbe utile che la loro chiamata venga effettuata solo quando necessario, senza usare i modelli più grandi per richieste piccole.