Università degli Studi di Padova

Catch em All - ***CAPTCHA: Umano o Sovraumano?***
Email: catchemallswe3@gmail.com

Specifica architetturale

Versione	0.1.0
Approvazione	-
Redazione	Zhen Wei Zheng, Matteo Stocco, Luca Brugnera
Verifica	Ana Lazic
Stato	-
Uso	Esterno
Distribuzione	Zucchetti S.p.A, Prof. Vardanega Tullio,
	Prof. Cardin Riccardo, Gruppo Catch Em All

Registro delle modifiche

Versione	Data	Descrizione	Autore	Ruolo
0.1.0	18/06/2023	Verifica generale del documento §3	Ana Lazic	Verificatore
0.0.7	18/06/2023	Modificata sezione §3	Luca Brugnera, Matteo Stocco	Progettista, Progettista
0.0.6	05/06/2023	Aggiunte descrizioni nella §3	Zhen Wei Zheng	Progettista
0.0.5	02/06/2023	Inizio stesura della §3	Zhen Wei Zheng	Progettista
0.0.4	30/05/2023	Modifiche e aggiunte nella §2	Zhen Wei Zheng	Progettista
0.0.3	25/05/2023	Inizio stesura della §2	Zhen Wei Zheng	Progettista
0.0.2	20/05/2023	Stesura della §1	Zhen Wei Zheng	Progettista
0.0.1	17/05/2023	Creazione bozza e struttura del documento	Zhen Wei Zheng	Progettista

Indice

Elenco delle figure

Elenco delle tabelle

1 Introduzione

1.1 Scopo del documento

Questo documento ha lo scopo di servire da linea guida per gli sviluppatori che andranno ad estendere o manutenere il prodotto. Di seguito lo sviluppatore troverà nel documento tutte le informazioni riguardanti i linguaggi e le tecnologie utilizzate, l'architettura del sistema e le scelte progettuali effettuate per il prodotto.

1.2 Scopo del prodotto

Dal proponente Zucchetti S.p.A. viene evidenziato, nel capitolato da loro proposto, una criticità negli attuali sistemi di sicurezza sulla rilevazione dei bot_G rispetto agli esseri umani. Oggi giorno il meccanismo più utilizzato per risolvere questo problema è il test CAPTCHA_G.
Un bot_G non è altro che una procedura automatizzata che, in questo caso, ha fini malevoli, come per esempio:

- Registrazione presso siti web;
- Creazione di spam_G;
- Violare sistemi di sicurezza.

I bot_G, grazie alle nuove tecnologie sviluppate con sistemi che utilizzano principalmente l'intelligenza artificiale, riescono a svolgere compiti che fino a poco tempo fa venivano considerati impossibili da svolgere per una macchina.
Ciò evidenzia che i CAPTCHA_G attuali risultano sempre più obsoleti, non andando a individuare correttamente tutti i bot_G, se non quasi nessuno.
Un'altra criticità individuata dal proponente è il sistema di classificazione delle immagini che sta effettuando Google grazie al proprio $\text{reCAPTCHA}_\text{G}$, che attualmente è il sistema più diffuso.
Questa criticità nasce dal beneficio che questa big tech_G ottiene dall'interazione degli utenti nel risolvere le task_G proposte, che portano alla creazione di enormi dataset_G di immagini classificate che possono essere utilizzate per l'apprendimento dei propri sistemi di machine learning o vendibili a terzi.
Il capitolato C1 richiede di sviluppare una applicazione web costituita da una pagina di login provvista di questo sistema di rilevazione in grado di distinguere un utente umano da un bot_G.
L'utente quindi, dopo aver compilato il form in cui inserirà il nome utente e la password, dovrà svolgere una task_G che sarà il cosiddetto test CAPTCHA_G.

1.3 Glossario

Per evitare ambiguità relative al linguaggio utilizzato nei documenti prodotti, viene fornito il **Glossario v 1.0.0**. In questo documento sono contenuti tutti i termini tecnici, i quali avranno una definizione specifica per comprenderne al meglio il loro significato.
Tutti i termini inclusi nel Glossario, vengono segnalati all'interno del documento Piano di qualifica con una G a pedice.

1.4 Standard di progetto

Per lo svolgimento del progetto il gruppo *CatchEmAll* ha scelto di utilizzare come norme di riferimento informativo la serie di standard **ISO/IEC 25000 SQuaRE** per definire i requisiti$_G$ e le metriche per valutazione della qualità di un prodotto e lo standard **ISO/IEC 15504 SPICE** per definire al meglio la qualità e le metriche di un processo.

1.5 Riferimenti

1.5.1 Riferimenti normativi

Riferimenti normativi utilizzati:

- Norme di Progetto v1.0.0;
- Capitolato d'appalto C1 *CAPTCHA: Umano o Sovrumano?* : `https://www.math.unipd.it/~tullio/IS-1/2022/Progetto/C1.pdf`.

1.5.2 Riferimenti informativi

Riferimenti informativi utilizzati:

- Diagrammi delle Classi - Materiale didattico del corso di Ingegneria del Software: `https://www.math.unipd.it/~rcardin/swea/2023/Diagrammi%20delle%20Classi.pdf`;
- Software Architecture Patterns - Materiale didattico del corso di Ingegneria del Software: `https://www.math.unipd.it/~rcardin/swea/2022/Software%20Architecture%20Patterns.pdf`;
- Design Pattern Architetturali - Materiale didattico del corso di Ingegneria del Software:
  - Dependency Injection: `https://www.math.unipd.it/~rcardin/swea/2022/Design%20Pattern%20Architetturali%20-%20Dependency%20Injection.pdf`;
  - Model View Controller: `https://www.math.unipd.it/~rcardin/sweb/2022/L02.pdf`.
- Design Pattern Creazionali: `https://www.math.unipd.it/~rcardin/swea/2022/Design%20Pattern%20Creazionali.pdf`;
- Design Pattern Strutturali: `https://www.math.unipd.it/~rcardin/swea/2022/Design%20Pattern%20Strutturali.pdf`;
- Design Pattern Comportamentali: `https://www.math.unipd.it/~rcardin/swea/2021/Design%20Pattern%20Comportamentali_4x4.pdf`;
- SOLID Principles of Object-Oriented Design: `https://www.math.unipd.it/~rcardin/swea/2021/SOLID%20Principles%20of%20Object-Oriented%20Design_4x4.pdf`.

2 Tecnologie coinvolte

2.1 Tecnologie per la codifica

2.1.1 Linguaggi

Tecnologia	Versione	Descrizione
PHP	8.1	Linguaggio principale per lo sviluppo del progetto, usato per gestire tutti le componenti dell'API. E' il linguaggio utilizzato dal framework Laravel
HTML	5	Linguaggio di markup utilizzato per impostare la struttura delle pagine web.
CSS	3	Utilizzato per la formattazione e la definizione dello stile delle pagine HTML.
JavaScript	-	Utilizzato per implementare il proof of work una volta richiesta la generazione del catpcha e per la gestione dinamica del front-end.
Python	3.10.9	Utilizzato per creazioni degli script di download e elaborazione delle immagini da Unsplash.

Tabella 2.1: Linguaggi utilizzati

2.1.2 Strumenti

Tecnologia	Versione	Descrizione
Composer	2.5.5	Gestore di dipendenze per il linguaggio di programmazione PHP.
Sqlite	3	Utilizzato per salvare le varie informazioni necessarie per la gestione dei captcha in un DB;
Scribe	4.19	Utilizzato per la generazione della documentazione dell'API. Visibile in \docs

Tabella 2.2: Strumenti utilizzati

2.1.3 Framework e librerie

Tecnologia	Versione	Descrizione
Laravel	10.10	Framework open-source scritto in PHP, semplifica lo sviluppo delle applicazioni web offrendo una vasta gamma di funzionalità.
Sanctum	3.2	Libreria offerta da Laravel per la gestione dell'autenticazione attraverso l'utilizzo di Middleware.

Tabella 2.3: Framework e librerie utilizzati

2.2 Strumenti per l'analisi del codice

Strumento	Versione	Descrizione
Phpunit	10.1	Strumento per l'analisi statica del codice;
Xdebug	3.2.1	Strumento utilizzato per il calcolo del code coverage.

Tabella 2.4: Strumenti per analisi utilizzati

3 Architettura

L'obiettivo principale del progetto è la creazione di un servizio captcha insieme a un'applicazione web dimostrativa che ne faccia uso, al fine di illustrarne il funzionamento. Per raggiungere questo obiettivo, il prodotto è stato realizzato come un'API REST.
Un'architettura API REST (Representational State Transfer) è un approccio architetturale per la progettazione di servizi web che si basa diversi principi e vincoli che la definiscono. Queste sono:

- **Stateless**: Ogni richiesta del client al server deve contenere tutte le informazioni necessarie per comprendere e soddisfare la richiesta. Il server non deve mantenere alcun stato delle richieste precedenti del client. Ogni richiesta viene considerata isolata e indipendente dalle altre;
- **Rappresentazione delle risorse**: Le risorse, come ad esempio i dati, sono rappresentate in un formato standard, come JSON o XML. Le rappresentazioni delle risorse possono essere trasferite tra client e server tramite richieste HTTP.
- **Interfaccia uniforme**: Un'API REST segue un insieme di operazioni standardizzate, tra cui GET, POST, PUT e DELETE, che sono applicate a risorse identificate da URL univoci. Questa interfaccia uniforme semplifica l'interazione tra client e server e rende l'API più intuitiva e prevedibile.
- **Livelli**: L'architettura API REST può essere composta da più livelli, come il bilanciamento del carico, i server intermedi e i servizi di cache, che possono essere utilizzati per migliorare la scalabilità, l'affidabilità e le prestazioni del sistema.

La componente centrale dell'applicazione risiede nel back-end, dove è stato utilizzato il framework Laravel per la codifica.
La parte back-end è responsabile principalmente della gestione delle richieste della generazione e verifica del Captcha.
D'altro canto, per la parte front-end, è stata creata una semplice pagina HTML per fare le richieste all'API. Questa pagina fornisce un'interfaccia utente elementare attraverso la quale gli utenti possono interagire con il servizio Captcha.

3.1 Diagrammi delle classi

3.1.1 Back-end

Il diagramma delle classi della parte back-end si può suddividere in varie sezioni, le quali svolgono ognuna una parte portante del progetto. Le sezioni sono:

- Generazione del Captcha;
- Captcha Resource;
- Verifica del Captcha;
- Gestione delle chiavi per l'encrypt e decrypt della soluzione del captcha;

- Download ed elaborazione delle immagini.

Per lo sviluppo delle varie funzionalità elencate il gruppo ha sfruttato diversi strumenti forniti dal framework Laravel, come gli Eloquent Model che forniscono un metodo semplice ed intuitivo per interagire con le entità presenti nel DB, e delle classi Controller per poter gestire la logica di gestione delle richieste nel modo più chiaro possibile. Infatti i Controllers possono raggruppare la logica di gestione delle richieste correlate in una singola classe.

3.1.1.1 Generazione del Captcha

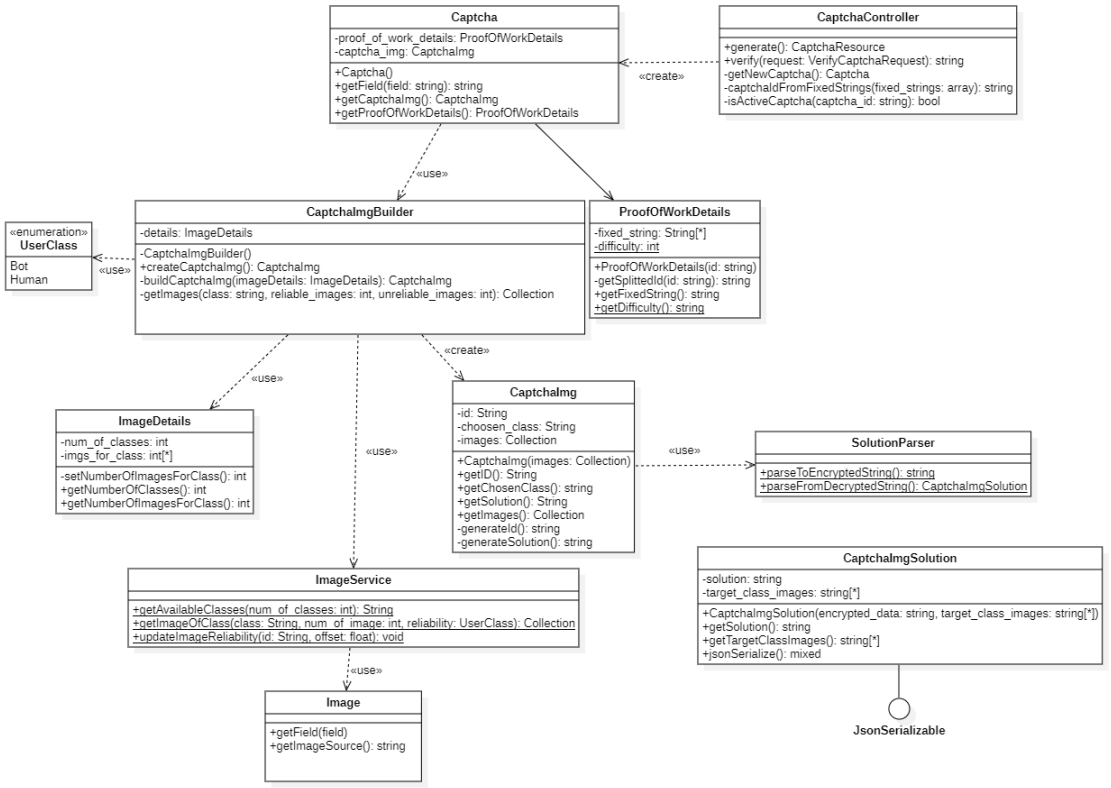

Figura 3.1: Diagramma delle classi. Generazione del Captcha

Nella figura sopra viene mostrata la logica utilizzata per la generazione di un Captcha.

- Ogni Captcha è composto da:
  - 9 immagini in bianco e nero con solo i contorni visibili;
  - L'immagine honeypot, invisibile all'utente;
  - Il proof of work.

In particolare:

- **CaptchaController**: Classe cardine del progetto, la quale si occupa di gestire tutte le richieste che verranno fatte all'API. In questa classe è anche contenuto il metodo che ritornerà l'oggetto *CaptchaResource* a seguito di una richiesta da parte dell'utente;
- **Captcha**: Classe che si occupa di mettere insieme tutti i pezzi che compongono il Captcha, ovvero immagini, honeypot e proof of work. Questa classe viene creata dal Controller quando viene richiesto un Captcha;
- **CaptchaImgBuilder**: Classe che utilizza il pattern dependency injection e si occupa dell'effettiva creazione della parte del Captcha che contiene le immagini e l'honeypot. Fa inoltre utilizzo di una enumerazione per la gestione dell'affidabilità delle immagini;
- **ProofOfWorkDetails**: Classe che contiene i dati necessari per il calcolo dei nonce nel proof of work;
- **ImageDetails**: Classe che contiene i dettagli utili alla creazione delle 9 immagini che comporranno il Captcha, quali il numero di classi e di immagini per classe che dovrà avere. Viene utilizzata dalla classe *CaptchaImgBuilder*;
- **CaptchaImg**: Classe che contiene le varie informazioni che la parte del captcha composta dalle immagini deve avere. È l'oggetto che la classe *CaptchaImgBuilder* va a creare;
- **ImageService**: Classe che si interfaccia il DB al fine di recuperare informazioni e immagini necessarie alla creazione del Captcha. Viene utilizzata dalla classe *CaptchaImgBuilder*;
- **Image**: Eloquent Model che contiene le informazioni dell'oggetto immagine. Viene utilizzato da *ImageService*;
- **CaptchaImgSolution**: Classe che contiene le informazioni della soluzione della parte del Captcha composta dalle immagini;
- **Solution Parser**: Classe che fornisce i metodi per criptare e decriptare istanze della classe *CaptchaImgSolution*.

3.1.1.2 Captcha Resource

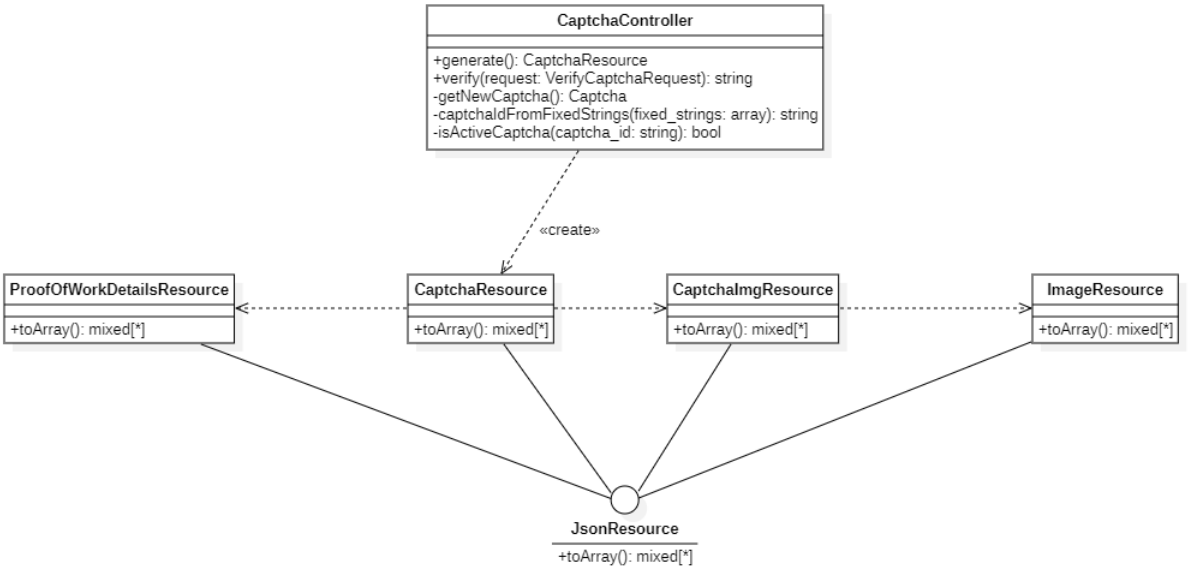

Figura 3.2: Diagramma delle classi. Captcha resource

Nella figura sopra viene mostrata la logica utilizzata per la gestione delle informazioni che si vogliono passare all'utente al seguito di una richiesta. Infatti dopo la creazione di un Captcha, viene creato dal Controller l'oggetto *CaptchaResource*, il quale sarà un JSON che contiene solamente le informazioni chiave per il caricamento del captcha lato front-end e la soluzione criptata che servirà durante la verifica del Captcha. Le classi Resource, come suggerito dal framework Laravel, vengono utilizzate per tradurre un Eloquent Model in formato JSON.

In particolare:

- **CaptchaController**: Dopo aver creato il Captcha, genera l'oggetto *CaptchaResource* il quale contiene le varie informazioni da passare in risposta alla richiesta dell'utente;
- **CaptchaResource**: Classe Eloquent Resource che utilizza le classi *CaptchaImgResource* e *ProofOfWorkDetailsResource* per generare il JSON che verrà poi inviato all'utente;
- **CaptchaImgResource**: Classe Eloquent Resource che seleziona le informazioni da inviare all'utente per la parte del Captcha immagini. Utilizza la classe *ImageResource* per le informazioni delle immagini contenute nel Captcha;
- **ProofOfWorkDetailsResource**: Classe Eloquent Resource che seleziona le informazioni da inviare all'utente per la parte del proof of work;

- **ImageResource**: Classe Eloquent Resource che seleziona le informazioni che ogni immagine dovrà avere nella risposta all'utente.

3.1.1.3 Verifica del Captcha

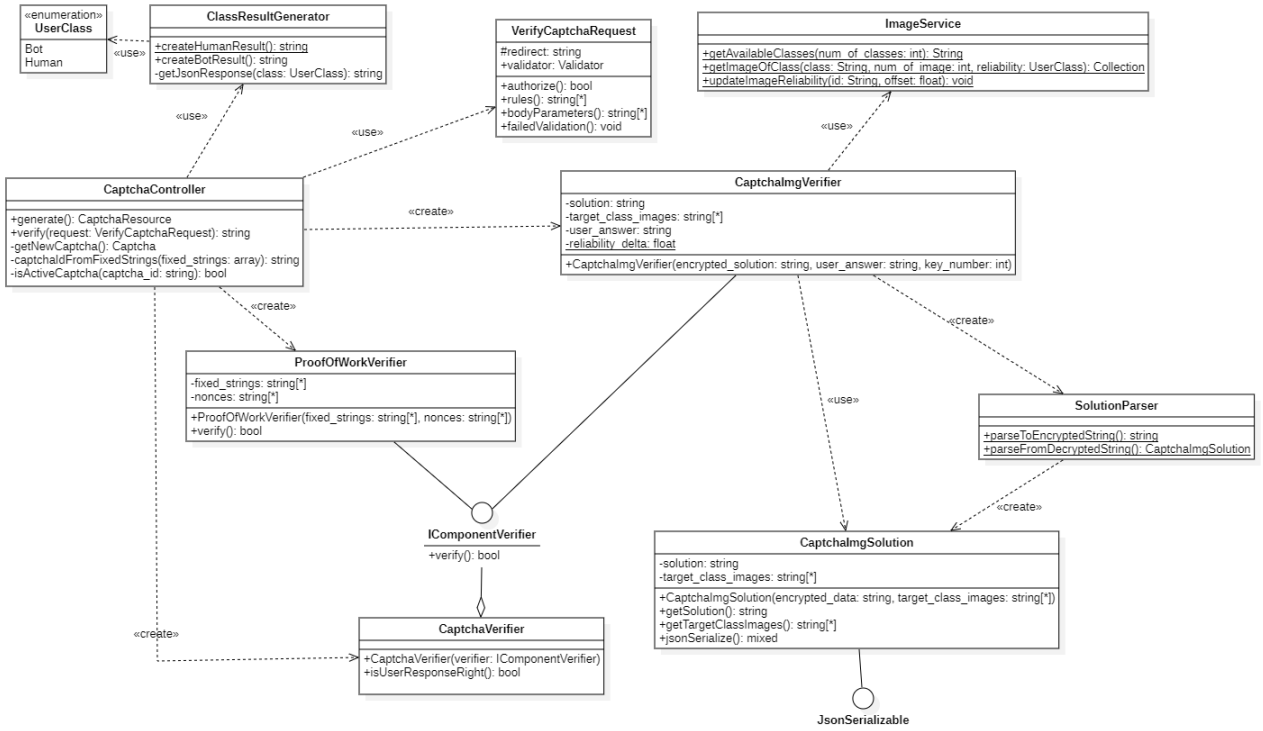

Figura 3.3: Diagramma delle classi. Verifica del Captcha

Nella figura sopra viene mostrata la logica utilizzata per la verifica dei Captcha inviati dagli utenti. *CaptchaController* riceve la richiesta e inizializza il verificatore passandogli i rispettivi verificatori del Captcha immagini e del Proof of Work. In particolare:

- **VerifyCaptchaRequest**: Dopo aver ricevuto la richiesta, Laravel costruisce un oggetto di questa classe la quale estende la classe astratta *Request*, verificando che sia conforme al set di regole specificato. Nel caso in cui il controllo fallisca, viene restituito un messaggio di errore con status code 400;
- **CaptchaController**: Questa classe si occupa di gestire le richieste per la verifica di un Captcha andando a creare i verificatori che andranno a controllare la correttezza effettiva del Captcha;
- **CaptchaVerifier**: Questa classe utilizza dei *ComponentVerifier* per controllare se la risposta dell'utente è corretta o meno;
- **IComponentVerifier**: Questa interfaccia possiede il metodo verify che verrà implementato dai due verificatori;

- **CaptchaImgVerifier**: Classe che verifica la correttezza della soluzione dell'utente per la parte del Captcha composto dalle immagini, andando inoltre ad utilizzare la classe *ImageService* per aggiornare l'affidabilità delle immagini nel DB;
- **ProofOfWorkVerifier**: Classe che verifica la correttezza dei nonce calcolati dall'utente durante il proof of work;
- **SolutionParser**: Classe utilizzata da *CaptchaImgVerifier* per decriptare e costruire un'istanza della classe *CaptchaImgSolution*;
- **CaptchaImgSolution**: Classe che contiene gli attributi della soluzione del Captcha. Creata da *SolutionParser* per fornire la soluzione decriptata a *CaptchaImgVerifier*;
- **ImageService**: Classe utilizzata dal *CaptchaImgVerifier* per aggiornare l'affidabilità delle immagini della classe target all'interno del DB;
- **ClassResultGenerator**: Classe che codifica il risultato criptandolo utilizzando una chiave simmetrica.

3.1.1.4 Gestione delle chiavi e dell'encrypting

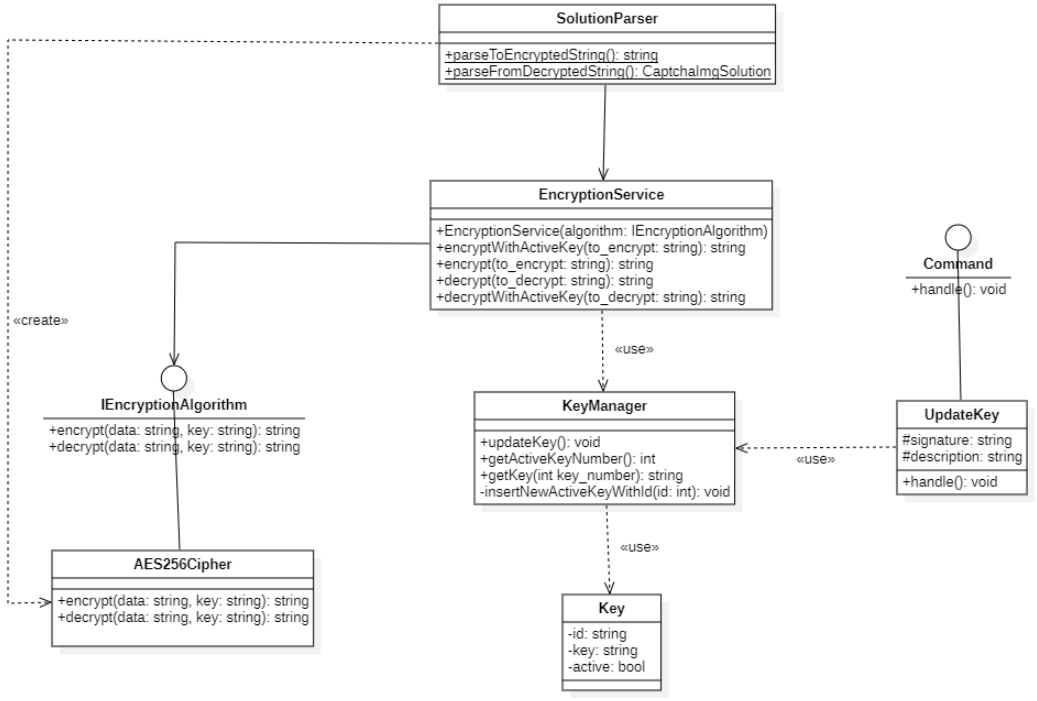

Figura 3.4: Diagramma delle classi. Gestione delle chiavi e dell'encrypting

Nella figura sopra viene mostrata la logica utilizzata per la gestione delle chiavi con cui vengono criptate le soluzioni dei captcha creati, ma anche la struttura in cui alla base c'è l'algoritmo che cripta e decripta le soluzioni.
In particolare:

- **SolutionParser**: Questa classe si occupa di criptare e decriptare le soluzioni dei Captcha utilizzando la classe *EncryptionService*;
- **EncryptionService**: Classe che implementa i metodi crypt e decrypt per le diverse situazioni in cui sono richiesti.
- **IEncryptionAlgorithm**: Questa interfaccia possiede i metodi crypt e decrypt che verranno implementati dalla classe *AES256Cipher*;
- **AES256Cipher**: Classe che implementa i metodi crypt e decrypt attraverso la chiave fornita e l'algoritmo AES256;

- **KeyManager**: Classe che contiene i vari metodi che permettono di inserire, recuperare e aggiornare le chiavi nel DB;
- **Key**: Eloquent Model che contiene i vari attributi che una chiave deve possedere. Utilizzato dalla classe *KeyManager*;
- **UpdateKey**: Derivata dalla classe Command di Laravel. Gestisce la chiamata del metodo per l'aggiornamento di una chiave nel DB presente in *KeyManager*. Questo comando è eseguito ogni minuto dallo scheduler definito dal framework.

3.1.1.5 Download e elaborazione delle immagini

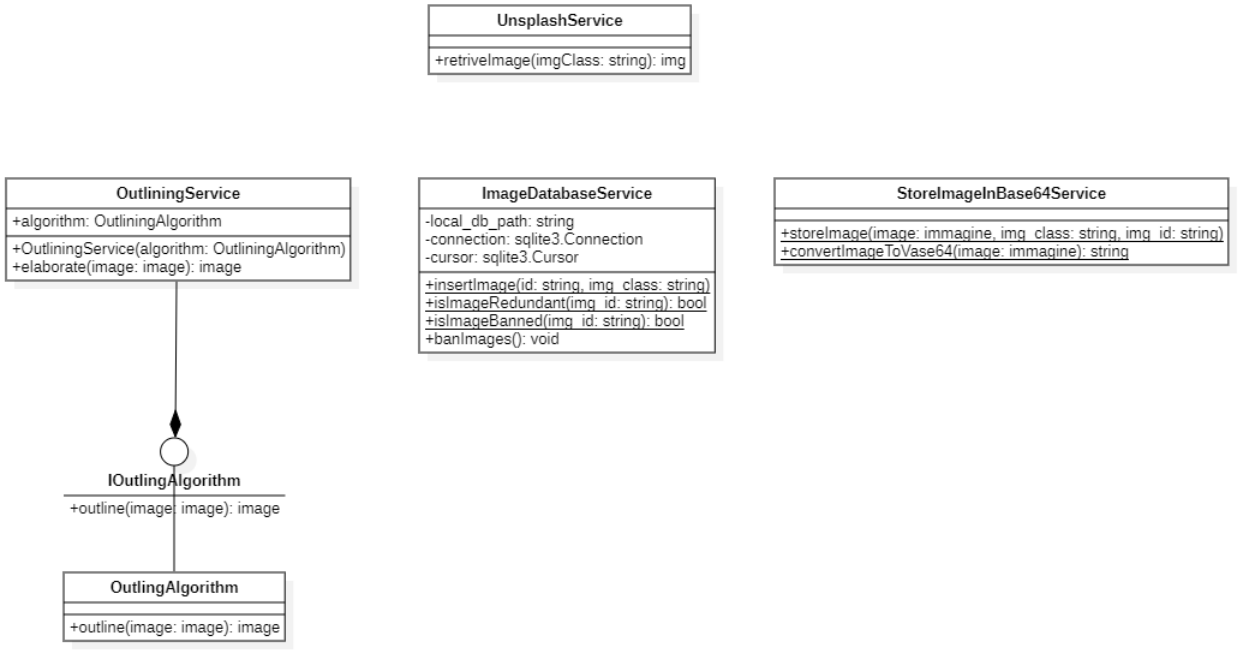

Figura 3.5: Diagramma delle classi. Recupero delle immagini da Unsplash

La figura sopra riportata rappresenta le varie classi e interfacce che sono utilizzate dallo script per il download delle immagini dal sito di Unsplash, la rielaborazione di tali immagini e il salvataggio di esse.
Questo processo segue il seguente ordine:

1. Inizialmente viene fatto un controllo per vedere se ci sono immagini con un'affidabilità troppo bassa e quindi eliminarle dal DB;
2. Dopodiché viene scaricata una nuova immagine di una determina classe da Unsplash;
3. Viene compiuta l'elaborazione dell'immagine eliminando i colori e tenendo solo il contorno;
4. Viene quindi convertita in base64;
5. Infine l'immagine viene salvata e inserita nel DB.

In particolare:

- **UnsplashService**: Classe che si occupa di fare la richiesta di un immagine al servizio di Unsplash;
- **OutliningService**: Classe che si occupa dell'elaborazione dell'immagine;
- **IOutliningAlgorithm**: Interfaccia che contiene il metodo che sarà poi implementato nella classe *OutilningAlgorithm* per l'elaborazione delle immagini;
- **OutilningAlgorithm**: Classe che implementa l'algoritmo di elaborazione dell'immagine;
- **StoreImageInBase64Service**: Classe si occupa della conversione in base64 dell'immagine e del suo salvataggio usando come path *classe/id_immagine*;
- **ImageDatabaseService**: Classe che implementa i vari metodi per eliminare, inserire e fare controlli sulle immagini nel DB.

3.1.2 Front-end

La parte del front-end del progetto, è completamente separata dall'API che offre il servizio Captcha ed è composta da:

- Pagina di login, nella quale l'utente può richiedere un Captcha, compilarlo e fare di conseguenza l'accesso;
- Un file JS che si occupa del calcolo del proof of work;
- Un file JS che si occupa di inserire dinamicamente i dati ricevuti in risposta dal servizio nella pagina e di far iniziare il proof of work;

Il lato client, per poter fare le richieste al servizio, fa utilizzo anch'esso di una classe Controller, la quale possiede due metodi che vengono chiamati nel caso l'utente voglia generare un Captcha o di verificarne uno. Questi metodi si occupano di fare le richieste effettive all'API, ottenendo i dati in risposta che saranno poi gestiti tramite JS, nel caso della generazione o nel caso della verifica di reindirizzare l'utente in base alla correttezza o meno della risposta inviata. Anche questa classe è completamente sconnessa dall'API e si occupa solo di fare le richieste la servizio.

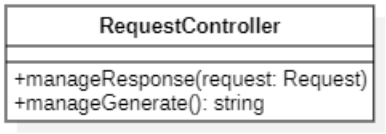

Figura 3.6: Controller per le richieste dell'utente

3.2 Architettura di dettaglio

3.2.1 Strategy pattern

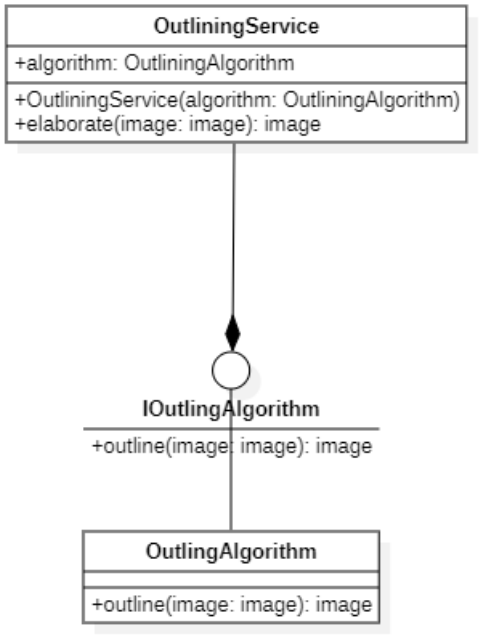

Figura 3.7: Strategy pattern dell'algoritmo di rielaborazione dell'immagine

Attualmente, l'interfaccia *IOutliningAlgorithm* ha una sola classe concreta che rappresenta l'unico algoritmo implementato. Tuttavia, è progettata in modo da consentire l'estensione per l'implementazione di ulteriori algoritmi in futuro. Ciò significa che se si desidera aggiungere nuovi algoritmi di scontorno delle immagini sarà possibile farlo creando nuove classi che estendono l'interfaccia *IOutliningAlgorithm*. Questa flessibilità consente di espandere il sistema per includere più algoritmi di scontorno delle immagini senza dover modificare la struttura di base.

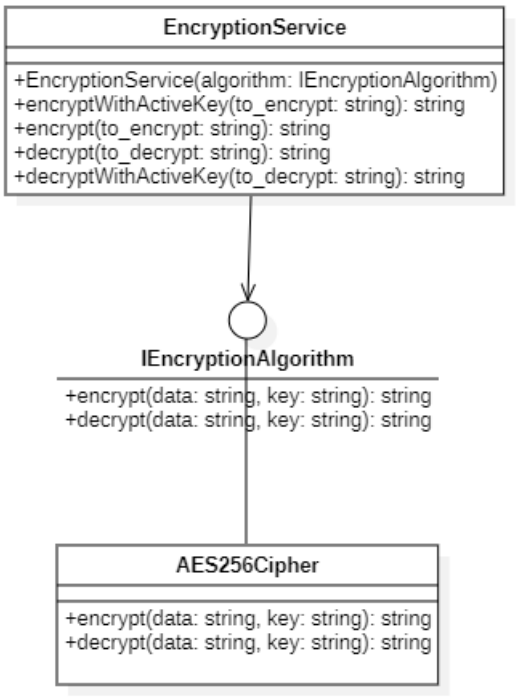

Figura 3.8: Strategy patter dell'algoritmo per l'encrypting

L'interfaccia *IEncryptionAlgorithm* possiede una sola classe concreta per lo stesso motivo citato precedentemente. In questa maniera, l'implementazione di un nuovo algoritmo di crittografia risulterà semplice e non richiederà la modifica di codice esistente.

3.2.2 Dependency Injection

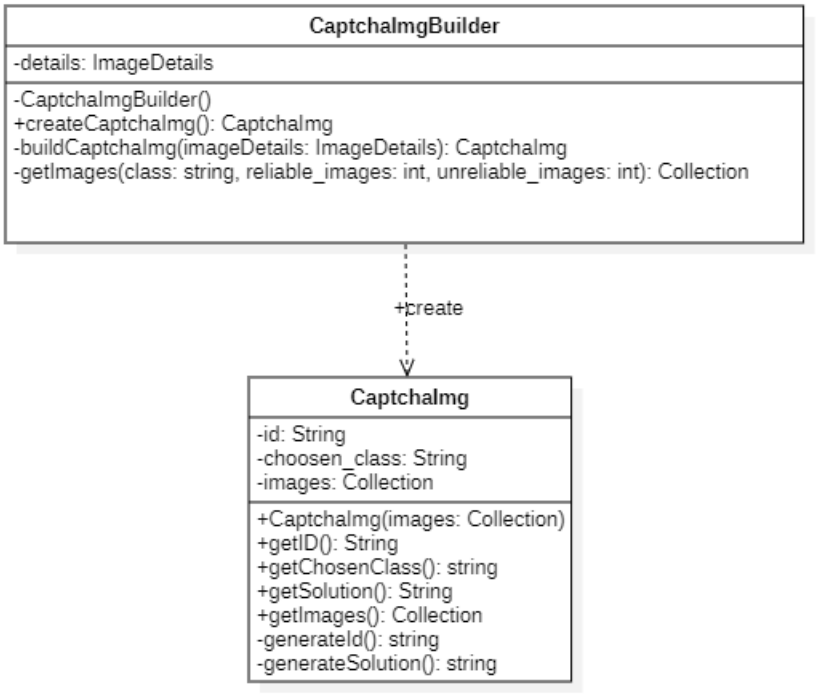

Figura 3.9: Dependency injection pattern

La costruzione di un'istanza di *CaptchaImg* è un'operazione che può essere considerata complessa in quanto bisogna tenere in considerazione diverse variabili, ad esempio il numero di classi di immagine presenti e numero di immagini per classe. Tali variabili devono a loro volta rispettare delle condizioni:

- Numero di classi tra 2 e 4 estremi compresi;
- 9 immagini totali;
- Le immagini della classe target devono essere la metà + 1 affidabili;
- Le immagini delle classi non target devono essere la metà affidabili.

Per quanto elencato sopra si è deciso quindi di spezzare la complessità e creare una classe il cui compito è quello di rifornire il costruttore di *CaptchaImg*.

3.2.3 Model-View-Controller

Il pattern architetturale su cui si basa il framework Laravel è *Model-View-Controller*. Lo sviluppo del servizio API per la generazione e verifica del Captcha sfrutta le classi messe a disposizione dal framework:

- **Illuminate\Database\Eloquent\Model**: Classe astratta per lo sviluppo di modelli la quale fornisce attributi e metodi integrati per la comunicazione con il database;
- **Illuminate\Routing\Controller**: Classe astratta per lo sviluppo di classi il cui compito è la gestione delle route e dei modelli.

Le viste, trattandosi di un API, non sono state utilizzate.

Questo pattern inoltre è alla base dello sviluppo dell'applicazione web utilizzata per testare il servizio Captcha.