UNIVERSITÀ
DEGLI STUDI
FIRENZE

Scuola di Ingegneria

Corso di Laurea Triennale in
Ingegneria Informatica

Sviluppo di un applicazione web per riconoscimento dei dettagli nelle opere d’arte di un museo

Candidato
Gennai Gian Maria

Relatore
Marco Bertini

Anno Accademico 2022/2023

Indice

Struttura della tesi

Capitolo 1

Nel primo capitolo viene introdotto il progetto, verranno spiegate a grandi linee tutte le funzioni e la struttura generale divisa tra back-end e front-end.

Capitolo 2

Nel secondo capitolo viene introdotto il lato front-end, quindi parleremo della progettazione, degli strumenti utilizzati per sviluppare il progetto, tutti i passaggi che avvengono quando l'utente utilizza l'applicazione e tutte le parti del codice che hanno un interesse pratico nel capire perchè sono state fatte in quel determinato modo.

Capitolo 3

Nel terzo capitolo viene spiegato come è stato strutturato il lato back-end e il database, anche qui verrà spiegata la fase di progettazione, gli strumenti utilizzati, i passaggi che il curatore del museo deve compiere per inserire dati e le parti del codice più rilevanti.

Capitolo 1

Introduzione

Il progetto sviluppato in questi mesi e che andremo a visionare durante la lettura di questa tesi è un'applicazione web chiamata SmartLens2 per il riconoscimento dei dettagli all'interno delle opere d'arte nei musei.

Un'applicazione web è un software specifico per essere utilizzato accedendo tramite un browser web. A differenza di una normale applicazione, questa non ha necessità di essere installata sul dispositivo. Nel caso specifico di questo progetto, era fondamentale sviluppare qualcosa di questo tipo perché nessun visitatore di un qualsiasi museo avrebbe ogni volta voglia o possibilità di installare un'applicazione sul dispositivo. L'applicazione web è stata sviluppata tramite:

- **HTML**: HTML (HyperText Markup Language) è il linguaggio di markup standard utilizzato per creare pagine web e definirne la struttura. Utilizza tag per marcare elementi come intestazioni, paragrafi, immagini e link.
- **PHP**: PHP è un linguaggio di scripting ampiamente utilizzato lato server progettato per lo sviluppo web. È incorporato all'interno di HTML e aiuta a generare pagine web dinamiche e interagire con i database. Per aggiungere del codice PHP però

devo avere un file con estensione *.php* perchè se è *.html* non posso mescolare i due linguaggi.

- **JavaScript**: JavaScript è un linguaggio di scripting versatile utilizzato principalmente per lo sviluppo web lato client. Consente di aggiungere interattività, contenuti dinamici e comportamenti ai siti web. È supportato da tutti i moderni browser web e può essere utilizzato anche per lo sviluppo lato server.

- **SQL**: SQL (Structured Query Language) è un linguaggio di programmazione utilizzato per gestire e manipolare i database relazionali. Viene utilizzato per creare, modificare e interrogare le tabelle e i dati all'interno di un database. SQL consente di eseguire operazioni come l'inserimento di nuovi dati, l'aggiornamento dei dati esistenti, la selezione di dati specifici o la cancellazione di dati non più necessari. È uno standard industriale ampiamente utilizzato per l'interazione con i database.

L'applicazione è formata sia da una parte front-end (lato del visitatore del museo) che da un lato back-end (lato del curatore del museo). Queste due parti sono entrambe collegate a un unico database nel quale vengono inseriti dati tramite l'interfaccia fornita al curatore, per poi essere presentati al visitatore del museo. Le due parti lavorano autonomamente, quindi il curatore può continuare a lavorare inserendo dati all'interno del database mentre qualcun altro sta utilizzando l'applicazione dal lato del visitatore.
Possiamo seguire tutti i passaggi che svolge il visitatore del museo e il curatore nei due mockup presenti in *figura 1.1* (per il visitatore) e *figura 1.2* (per il curatore). Seguendo i mockup possiamo vedere che sul lato font-end viene aperta l'applicazione, appena viene cliccato il bottone di avvio si apre quindi una pagina dedicata a leggere il QR Code relativo all'esibizione in corso all'interno del museo, viene aperta la webcam e avviato il riconoscimento, una volta che viene rilevato un dettaglio su un opera d'arte vediamo poi apparire la barra in basso con le icone relative ad esso e cliccandoci sopra si

apriranno tutte le informazioni relative come vediamo nell'ultimo passaggio. Per quanto riguarda il back-end invece partiamo dalla pagina iniziale(*dashboard*) e ci spostiamo, cliccando sopra ai vari nomi delle pagine che troviamo sulla barra di navigazione laterale, per andare sulle pagine dedicate e cliccando su *Add a New ...* per farci inserire dati nella tabella corrispondente nel database tramite l'interfaccia che vediamo nell'ultimo passaggio per poi cliccare su *Save* e concludere il processo.

Per quanto riguarda l'interfaccia generale, questa è stata resa il più semplice possibile per far fronte a un pubblico molto ampio. L'applicazione è formata da una rete che riconosce i dettagli all'interno delle opere d'arte esposte in un'esibizione all'interno di un museo. Per fare questo, la rete viene allenata tramite le immagini che il curatore inserisce all'interno del database. Le reti che vengono utilizzate per riconoscere i dettagli sono tre:

- **Classification** : Classification (Classificazione oggetti): Questo metodo mira a assegnare una classe o una categoria specifica a un oggetto presente in un'immagine o in un video. Invece di individuare e delimitare gli oggetti, la classificazione si concentra sull'etichettatura degli oggetti rilevanti già individuati o su regioni di interesse all'interno dell'immagine. Qui viene utilizzata la rete **MobileNet** che è una famiglia di modelli di reti neurali convoluzionali ottimizzati per l'esecuzione su dispositivi mobili o con risorse computazionali limitate.

- **Retrieval**: Retrieval (Recupero oggetti): Questo metodo riguarda la ricerca di oggetti o immagini simili in un database, a partire da un'immagine di query fornita dall'utente. Utilizzando caratteristiche come colori, forme o texture, il sistema di recupero di oggetti cerca di trovare le corrispondenze più simili all'immagine di query all'interno del database. Qui viene utilizzata la rete **MobileNet Small** che è una variante più leggera di MobileNet, progettata per dispositivi con risorse ancora più

limitate, come dispositivi mobili di fascia bassa o dispositivi IoT.

- **Object Detection**: Object Detection (Rilevamento oggetti): Questo metodo si concentra sull'individuazione e delimitazione degli oggetti all'interno di un'immagine o di un video. L'obiettivo è identificare le posizioni esatte degli oggetti nel contesto dell'immagine e assegnare una classe o una label a ciascun oggetto rilevato. Qui viene utilizzata la rete **SSD/MobileNet** dove SSD (Single Shot MultiBox Detector) è un algoritmo di rilevamento oggetti in immagini e video. L'implementazione di SSD/MobileNet combina la potenza di MobileNet con la capacità di rilevare oggetti in tempo reale.

Tutte e tre queste reti fanno parte della libreria **Tensorflow.js** la quale è una libreria open source di machine learning basata su JavaScript. Consente di eseguire algoritmi di machine learning e addestrare modelli direttamente nel browser o in ambienti JavaScript, senza richiedere la necessità di un server o di un'infrastruttura aggiuntiva.
Come possiamo notare abbiamo che con *Object Detection* oltre a riconoscere il dettaglio riesco a rilevare anche la posizione e quindi quando vado ad inquadrare l'opera d'arte da lontano, questo mi va a racchiudere con dei box tutti i dettagli che riesce a riconoscere, io a quel punto tramite l'applicazione posso sceglierne uno di quelli riconosciuti, cliccarci sopra e leggere tutte le informazioni che sono state inserite dal curatore in un primo momento. Invece per quanto riguarda gli altri due metodi di riconoscimento, non mi viene ritornato alcun dato riferito alla posizione del dettaglio riconosciuto e di conseguenza io non posso andare a riquadrarli tramite i box colorati e questo perché, semplicemente, non sono fatti per questo scopo ma sono io che avvicinandomi all'opera d'arte e usando lo smartphone a mo' di una lente di ingrandimento devo andare a scovare i dettagli e a quel punto quello che prima avevamo chiamato come box colorato che riquadra il dettaglio diventa proprio tutto lo schermo. Poi

procede a stampare le informazioni del dettaglio trovato come con l'*object detection*.

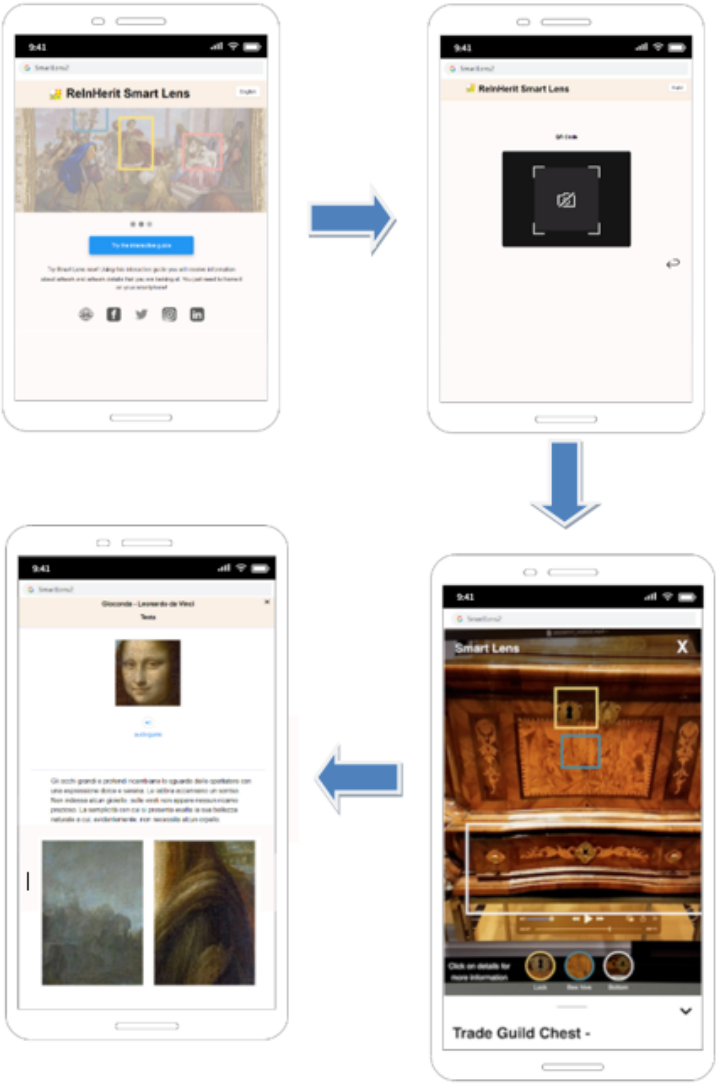

Figura 1.1: Mockup relativo al front-end

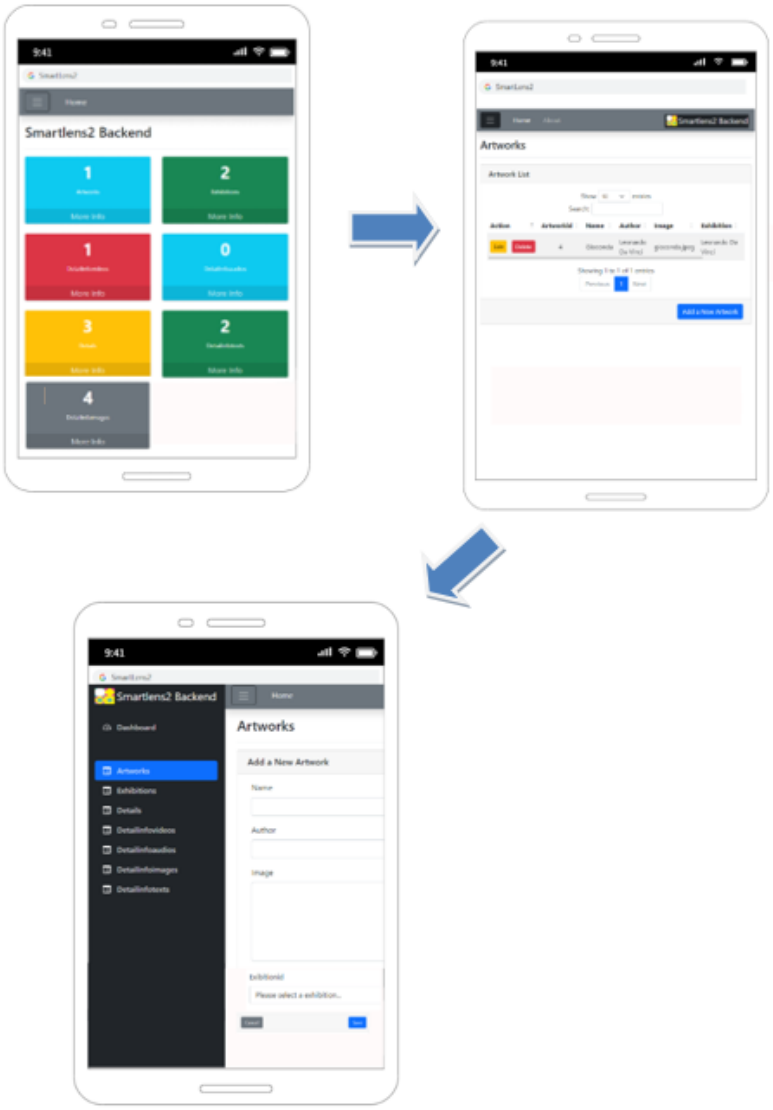

Figura 1.2: Mockup relativo al back-end

Capitolo 2

Front-End

La parte *front-end* di questa applicazione è quella che si interfaccia direttamente con il visitatore del museo.

2.1 Guida all'applicazione SmartLens2 front-end

Siamo nella parte front-end, quindi il visitatore del museo ha in mano lo smartphone con l'applicazione aperta sulla pagina iniziale. L'utente, all'interno della home dell'applicazione web, può subito andare a modificare la lingua a suo piacimento per poi passare ad avviare il riconoscimento.
All'interno della pagina principale abbiamo alcune immagini e due righe introduttive per l'applicazione e poi i vari link social relativi al progetto(*figura 2.1*). Quando l'utente clicca sul bottone per l'avvio del riconoscimento dei dettagli, automaticamente si aprirà la pagina dedicata alla lettura del QR code(*figura 2.2*) e qui dovrà semplicemente inquadrare all'interno del riquadro che appare a video il codice relativo all'esibizione che si vuole vedere in quel momento. Appena inquadrato il QR code, si aprirà la schermata dove vengono rilevati i dettagli delle opere esposte che, dopo una breve attesa dovuta allo scaricamento dei moduli per il riconoscimento, fa partire la webcam dello smartphone. Finché non viene inquadrata nessuna

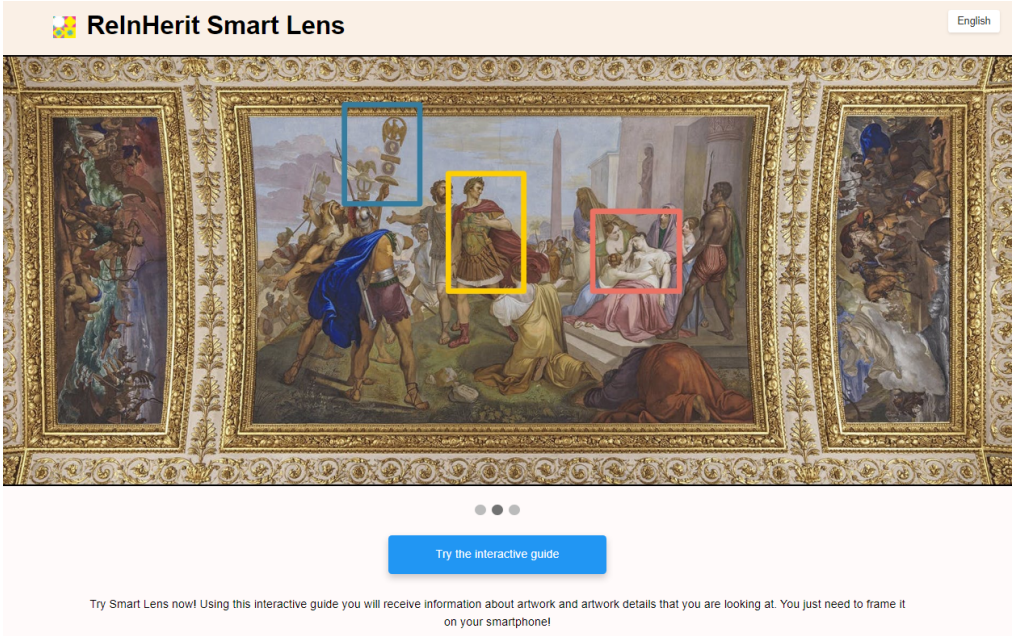

Figura 2.1: Home page(index.php)

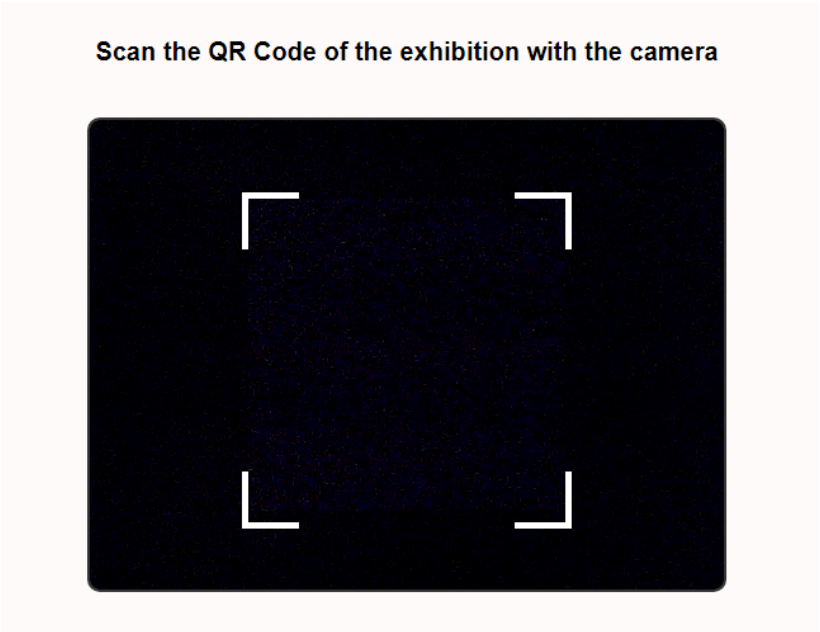

Figura 2.2: Pagina per la scannerizzazione del QR Code

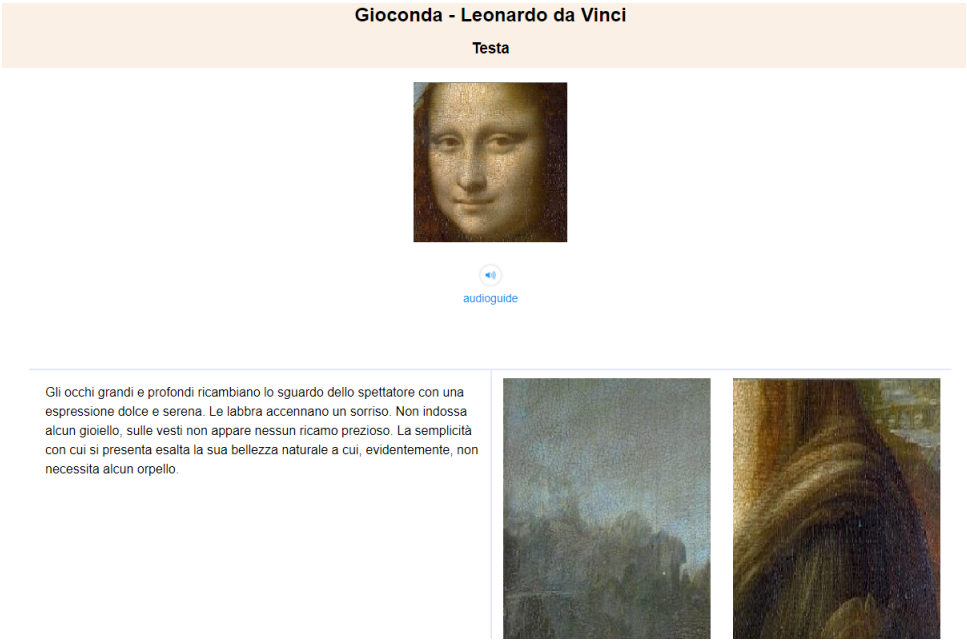

Figura 2.3: Pagina dedicata al dettaglio rilevato sull'opera d'arte

opera, continueremo a vedere soltanto il video fatto dalla webcam perché non viene riconosciuto alcun dettaglio.
Il visitatore inquadra una prima opera d'arte e, dato che viene preso un frame dal video ed elaborata l'immagine ogni secondo, se ho qualche dettaglio presente questo viene inquadrato immediatamente. Appena vengono riconosciuti i dettagli all'interno dell'opera, appare una barra in basso dello schermo e vengono stampate le icone dei dettagli, le quali, se cliccate, aprono a loro volta una pagina dove troviamo eventuali spiegazioni sotto forma di video, testo, immagini e audio(*figura 2.3*).

Le modalità con cui vengono riconosciuti i dettagli sono quelle già spiegate sopra nell'introduzione. Abbiamo l'*Object Detection* che mi va a rilevare tutti i dettagli disponibili inquadrando solamente l'opera d'arte e poi abbiamo gli altri due metodi invece che mi permettono di fare procedimenti più manuali e didattici per scoprire le opere utilizzando lo smartphone come se fosse una lente d'ingrandimento.

2.2 Progettazione

Nelle prime fasi mi sono studiato tutto il codice andando a leggermi tutto quello che c'era da sapere per quanto riguarda le tre reti utilizzate per il riconoscimento dei dettagli e tutto quello che non sapevo rispetto al codice in generale. Dopo aver capito cosa faceva ogni metodo all'interno dell'applicativo ho iniziato ad annotarmi tutti gli errori che ho notato partendo dalla struttura intera del progetto e quindi anche il posizionamento dei file all'interno di cartelle fino ad arrivare ad annotarmi errori e migliorie da fare all'interno di ogni metodo.
La fase più lunga da superare è stata quella del nuovo database. Il database che è stato inserito all'interno dell'applicazione SmartLens2 è stato progettato da me completamente da zero, andando a disegnarne prima la struttura e poi andandolo a creare(come verrà spiegato nel prossimo capitolo), in modo che fosse uno strumento veloce ed efficace per selezionare tutti i dati necessari a far funzionare la *web application*.

2.2.1 Multilingualità all'interno della web application

L'applicazione web è stata creata per ambienti dove ci sono persone provenienti da tutto il mondo e quindi deve essere in grado di poter fornire una grande selezione di lingue differenti.
Il visitatore del museo, all'interno dell'applicazione, può decidere quale lingua selezionare tramite un menù a tendina che si trova in alto a destra della pagina principale o della pagina dove viene inquadrato il QR code. La lingua di default è l'inglese.
All'interno del progetto, la parte di multilinguismo è completamente automatizzata. Abbiamo una struttura dedicata esclusivamente a questo compito, formata da un file chiamato `getText.php`. Tale file ha il compito di richiedere al database le varie parti di testo da stampare nell'HTML e cercare quante lingue sono presenti all'interno del database(nella tabella interna corrispondente) per includerle

direttamente all'interno del menù a tendina.
Per quanto riguarda il file `getText.php`, è formato da due metodi. Il primo metodo riceve la lingua e il nome del testo da inserire (che funge da identificativo) e restituisce il risultato del database. Il secondo metodo, invece, restituisce un vettore contenente tutte le lingue diverse presenti nel database. Il primo metodo viene chiamato all'interno dei file contenenti del testo, in modo che il valore di ritorno e il comando `echo` vengano utilizzati per stampare il testo all'interno del tag `p` dell'HTML. Il secondo metodo serve per aggiungere automaticamente lingue al menù a tendina ogni volta che viene aggiunto un nuovo testo in una lingua diversa nel database.
Qui di seguito possiamo vedere un pezzo di codice preso dal file della pagina principale dove all'interno del tag **<p>** HTML viene stampato il testo tramite il comando `echo` che prende il valore di ritorno di `getTextLang(...)`, che fa la richiesta ad database per farsi restituire il testo chiamato *indexText* e in lingua corrispondente al valore che c'è all'interno della variabile `$selectedLang`.

```
<p id="body-text">
    <?php
    echo getTextLang("indexText", $selectedLang);
    ?>
</p>
```

La tabella interna al database (*figura 2.4*) è molto semplice ed efficace per questo scopo. Infatti, è composta solo da quattro campi: l'`id` del testo, il `name` per identificare di quale testo si tratta e quindi anche dove è posizionato, la `lang` che rappresenta la lingua del testo e `text`, che è il testo da visualizzare.
Come ultima cosa notiamo che i file con codice `HTML` sono con estensione `.php` e questo perché io posso mescolare i due linguaggi `HTML` e `PHP` solo in questo modo sennò quest'ultimo non viene riconosciuto.

Figura 2.4: Tabella webbapptext per il testo nelle varie pagine in lingue differenti

2.3 Strumenti utilizzati

Per lo sviluppo del progetto inizialmente mi sono avvalso di un **IDE Integrated Development Environment**(Ambiente di Sviluppo integrato) fornito da **JetBrains**, *WebStorm* progettato per lo sviluppo di applicazioni web il quale permette di scrivere codice in `HTML`, `CSS` e `JavaScript` ma non riconosce il linguaggio `PHP` e dato che io ne avevo bisogno per mandare delle richieste(*query*) al database sono dovuto passare ad utilizzare **PhpStorm**(sempre fornito da JetBrains) e che permette oltre l'utilizzo dei linguaggi che già stavo utilizzando, anche del `PHP`.
Il codice man mano che veniva sviluppato è stato sempre tenuto aggiornato su GitHub che è una piattaforma di hosting per il controllo delle versioni di progetti software utilizzando il sistema di controllo delle versioni `Git`.

2.4 File .php

Partiamo adesso con i cambiamenti apportati all'interno dei file con estensione `.php`.
All'interno di questi file la priorità più alta iniziale è stata sistemare tutto il sistema per il cambio della lingua e renderlo automatico com'è già stato spiegato nel sottocapitolo che troviamo sopra.
Il file `qrCodeScan.php`, che ho già nominato, è stato aggiunto da me in un momento successivo per andare a far fronte ad un ulteriore problema che si era presentato. Quando il visitatore del museo arriva ed apre l'applicazione, il riconoscitore dei dettagli deve attivarsi scaricando il giusto modello di riconoscimento e dato che ce ne

sono tre disponibili che il curatore può inserire ci deve essere qualcosa che faccia capire all'applicazione quale scaricare. Per risolvere questo ho inserito un lettore di **QR Code** in modo che venga li riconosciuto il nome della mostra, poi associato al nome della mostra all'interno del database io ho il tipo di riconoscimento associato e di conseguenza viene scaricato il modello corretto.
Dopo aver letto il codice QR, viene ritornato dal database il tipo di riconoscimento che c'è associato all'esibizione e quindi viene chiamato, tramite il tag **<script>** HTML, il file JavaScript corrispondente. Qui di seguito possiamo vedere la differenza fra avere i tre tag per i tre differenti file javascript collegati ognuno ad un differente metodo di riconoscimento e tutte le volte per cambiare metodo dover scommentare e commentare le righe e poi possiamo vedere invece, tramite l'utilizzo del database, come viene migliorata la struttura.
Versione senza database:

```
<script src="js/getDetailsObjDet_json.js" type="module"></script>
<!--<script src="js/getDetailsRetrieval.js" type="module"></script>-->
<!--<script src="js/getDetailsClass.js" type="module"></script>-->
```

Versione con database:

```
<?php
$conn = new mysqli(DB_HOST, DB_USER, DB_PASSWORD, DB_DATABASE);
if (!$conn) {
    echo 'Connection error: ' . mysqli_connect_error();
}
$sql = "SELECT * FROM exhibition WHERE name='$selectedValue'";
error_log('SQL query: ' . $sql); // debugging
$result = mysqli_query($conn, $sql);
if (!$result) {
    echo 'Query error: ' . mysqli_error($conn);
}
$exhibition = mysqli_fetch_array($result);
$typeOfDetection = $exhibition['typeOfDetection'];
```

```
// Create an associative array with the variable value
$exhibitionName = array('name' => $exhibition['name']);
$response = array('exhibitionName' => $exhibitionName);

// Encode the array as JSON
$jsonResponse = json_encode($response);

// Echo the JSON-encoded response
echo "<script>var exhibitionName = $jsonResponse;</script>";
$detailsJSUrl = '';
if($typeOfDetection == 'objdetNetwork')
    $detailsJSUrl = 'js/getDetailsObjDet_json.js';
if($typeOfDetection == 'classNetwork')
    $detailsJSUrl = 'js/getDetailsClass.js';
if($typeOfDetection == 'retrievalNetwork 1' ||
    $typeOfDetection == 'retrievalNetwork 2') {
    $detailsJSUrl = 'js/getDetailsRetrieval.js';
    if($typeOfDetection == 'retrievalNetwork 1')
        $version = 1;
    else if($typeOfDetection == 'retrievalNetwork 2')
        $version = 2;
    echo "<script>var version = $version;</script>";
}
?>
<script src="<?php echo $detailsJSUrl?>"
type="text/javascript"></script>
```

Qui sopra nel codice possiamo vedere che per la **$TypeOfDetection** nel caso del *retrieval* abbiamo due versioni e questo perché qaundo vengono preparati i dati io gli posso fornire le immagini divise in due modi differenti per effettuare l'allenamento, ovvero gli passo l'immagine originale divisa in quattro parti uguali oppure gli restituisco l'immagine originale con le quattro parti uguali più una quinta che prende il centro dell'immagine.

2.5 File .js

All'interno dei file JavaScript abbiamo tutte le operazioni per il riconoscimento dei dettagli e per la loro stampa a video.
Il funzionamento generale è molto simile per ogni metodo di riconoscimento, infatti come possiamo vedere dalla cartella dei file *JS* ne abbiamo tre dove ci sono i metodi specifici per ogni modello di rilevamento e sono *getDetailsClass.js, getDetailsObjDet_json.js* e *getDetailsRetrieval.js* e invece abbiamo *getDetails.js* e *displayDetails.js* che sono i file dove all'interno sono contenuti i metodi comuni a tutti e tre i tipi di ricerca anche se all'interno ad alcuni di questi c'è sempre qualche **if** che mi va a fare delle distinzioni per quanto riguarda i tipi che possono rilevare più dettagli alla volta e quindi devono poter andare a disegnare anche il box attorno al particolare trovato per identificarlo all'interno dell'opera d'arte e altri invece devono tralasciare tutta questa parte.
Parlando dei metodi di rilevazione che possiamo utilizzare: *Classification*, *Retrieval* e *Object Detection*, so che ogni volta che viene preso un frame del video e viene elaborato e restituito un vettore(tensore) contenente le varie predizioni fatte dalla rete allenata. Quando viene restituito il vettore all'interno dell'object detection c'è il problema che per ogni modello differente restituisce i valori tutti in una posizione diversa e quindi dato che io ho bisogno di prendere i box per riquadrare il dettaglio, la classe e la precisione, ho dovuto trovare una soluzione. La soluzione è relativamente semplice, controllando come è strutturato il vettore che viene restituito, possiamo notare che ci sono in tutto 8 elementi che potrebbero essere quelli che noi cerchiamo e ognuno di questi ha almeno una differenza di struttura rispetto agli altri e quindi tramite una selezione rigorosa riesco a trovare la posizione all'interno del vettore senza dover controllare a mano tutte le volte. All'interno di questo vettore possiamo trovare, per esempio, vettori con un solo elemento, vettori di vettori con un numero n di elementi, vettori con elementi di tipo intero o float ecc.
Di seguito vediamo un pezzo del codice che ci ha permesso selezio-

nare le posizioni giuste all'interno del vettore di predizione restituito dall'*Object detection*:

```
let boundingBoxes = undefined;
let classes = undefined;
let probabilities = undefined;
for(let i = 0; i < predictions.length; i++) {
    predictions[i] = predictions[i].arraySync();
    if(predictions[i][0][0] != undefined) {
        if (!Number.isInteger(predictions[i][0][0]) &&
        predictions[i][0].length == 100 &&
        predictions[i][0][0].length != 26 &&
        predictions[i][0][0].length != 4) {
            probabilities = predictions[i][0];
        }else if(!Number.isInteger(predictions[i][0][0]) &&
        predictions[i][0].length == 100 &&
        predictions[i][0][0].length != 26 &&
        predictions[i][0][0].length == 4){
            boundingBoxes = predictions[i][0];
        }else if(Number.isInteger(predictions[i][0][0]) &&
        predictions[i][0].length == 100 &&
        predictions[i][0][0]<1000){
            classes = predictions[i][0];
        }
    }
}
```

Come possiamo vedere abbiamo un ciclo **for** che mi serve per scorrere tutte le posizioni del vettore restituito dalla predizione che è stato chiamato **predictions** e poi abbiamo tutta una serie di **if** che mi vanno a verificare cosa c'è all'interno di ogni posizione nel vettore delle predizioni. Una volta controllato che il valore restituito è quello desiderato allora andiamo ad inserirlo all'interno dei vettori definiti *undefined* sopra.
Il file *getDetails.js* è stato molto utile per fare tutte le richieste dei dati verso il database, in particolare io non posso fare direttamente delle *query* passando dal JavaScript al database ma de-

vo sempre passare prima per il PHP. Per mandare richieste dal JS al PHP mi sono aiutato tramite la libreria **jQuery** usando il comando `$.ajax(...);` funzione che mi permette di fare richieste **AJAX**(Asynchronous JavaScript and XML) verso il server. jQuery è una popolare e ampiamente utilizzata libreria JavaScript. Semplifica ed ottimizza l'interazione con i documenti HTML, la gestione degli eventi, la realizzazione di richieste AJAX, le animazioni degli elementi e la manipolazione del DOM (Document Object Model). Qui di seguito possiamo vedere l'esempio di una richiesta utilizzando i metodi appena descritti.

```
    $.ajax({
  url: 'https://example.com/api/data',
  type: 'GET',
  dataType: 'json',
  success: function(response) {
    // Handle the successful response
    console.log(response);
  },
  error: function(xhr, status, error) {
    // Handle any errors
    console.error(error);
  }
});
```

Qui sopra quindi possiamo vedere una semplice richiesta di esempio verso il file che si trova in `https://example.com/api/data`, la richiesta è fatta da un `GET` e il dato è di tipo `json`. Se io ho successo e vengono ritornati i dati allora verrà utilizzato il metodo che si trova su *success* sennò quello che si trova su *error*. Gli altri file che troviamo all'interno della cartella js servono per fare operazioni di poco rilievo o comunque per stampare a video le informazioni sui dettagli.

2.6 Modifiche di stile

All'interno del progetto ho anche apportato delle modifiche estetiche per far si di avere un qualcosa di carino e funzionale.
In tutte le pagine che abbiamo ho fatto si di distaccare la testata della pagina con il resto del corpo andando a farli di colorazione differente senza però andare a inserire alcuna linea divisoria tra il sopra e il sotto. I colori che ho utilizzato sono tutti molto tenui ma sono tendenti al giallo chiaro in modo che se la mostra a cui stiamo partecipando si trova all'interno di una sala buia o comunque con poca luce, il contrasto quando prendiamo il dispositivo è minore.

Capitolo 3

Back-End

La parte *back-end* di questa applicazione è quella che si interfaccia con il curatore del museo.

3.1 Guida all'applicazione SmartLens2 back-end

Siamo adesso sul lato back-end e quindi lo smartphone è in mano al curatore del museo.
Come prima cosa il curatore apre l'applicazione e si trova davanti la pagina principale formata da una *Dashboard* (la schermata iniziale *figura 3.1*) dove sono rappresentate con dei riquadri colorati tutte le tabelle del database che il curatore può vedere (le abbiamo tutte escluso quelle relative ai dati per il riconoscimento dei dettagli) e ognuno di questi riquadri colorati ha un numero che indica quanti elementi interni alla tabella ci sono. Da qui è possibile visualizzare all'interno dei riquadri il numero di elementi che ci sono dentro la tabella ed è possibile cliccarci sopra e andare direttamente alla pagina relativa alla tabella selezionata.

Sia che il curatore apra una pagina relativa ad una tabella tramite la dashboard (ogni riquadro relativo ad una tabella è anche un link) oppure tramite il menu laterale, il risultato è lo stesso e ci troviamo in entrambi i casi davanti alla pagina scelta.

Figura 3.1: Pagina iniziale Back-End(*Dashboard*)

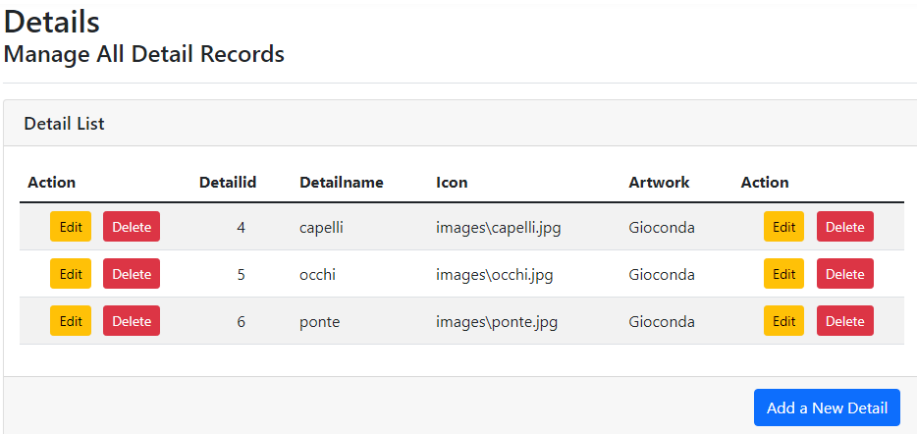

Figura 3.2: Pagina riassuntiva di tutti gli elementi inseriti all'interno di una tabella nel database(in questo caso *details*)

All'interno di una qualsiasi pagina dedicata alla tabella(*figura 3.2*) troviamo inizialmente una lista di tutti gli elementi già inseriti e da dove possiamo, oltre che leggere quello che c'è scritto, anche modificarli o eliminarli. Per aggiungerne di nuovi possiamo procedere cliccando sul bottone dedicato in basso a destra.

Cliccando su **Add a New ...** si aprirà una nuova pagina, formata da una serie di campi che servono per inserire i valori all'interno delle tabelle nel database(*figura 3.3*). Dobbiamo stare attenti perché per inserire tutti i dati in una tabella con una *foreign key*, non vedremo apparire alcun campo al di fuori di quello relativo alla chiave esterna finché non avremo selezionato il valore relativo a quella chiave. Dopo aver fatto ciò, appariranno tutti gli altri campi. Questa modifica è

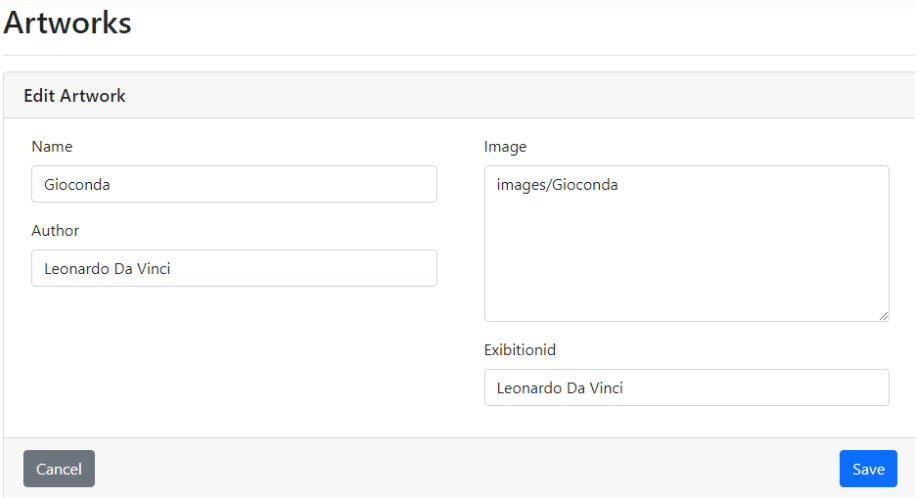

Figura 3.3: Pagina fornita al curatore del museo per inserire i dati e poi salvarli sul database(esempio con *artwork*)

stata aggiunta per evitare problemi con l'applicazione sul front-end, poiché se il curatore si dimentica di inserire quel valore, potrebbero verificarsi inconsistenze e bug nell'applicazione sulla parte del visitatore del museo.
Dopo aver inserito tutti i dati nei campi desiderati, possiamo cliccare sul bottone **Save** per salvare il tutto e automaticamente verrà inserito sul database nella tabella corrispondente e incrementato il numero che mi indica quanti elementi sono presenti all'interno della dashboard.
In sintesi, il curatore può da qui soltanto inserire le informazioni relative ai dettagli e alle opere, eliminarle o modificarle, e successivamente può visualizzare tutte le informazioni che sono state inserite nel tempo nelle varie tabelle.

3.2 Progettazione

La progettazione della parte back-end parte dallo studio di un database ben strutturato.
Il database utilizzato per lo sviluppo di questa applicazione è stato creato completamente da zero, è composto da dieci tabelle ed è di

tipo relazionale. La struttura del database può essere associata a un piccolo albero, in cui tutto parte dall'esibizione e si espande. Partendo dalle tabelle più esterne, abbiamo le tre *classmapping, objectmapping, retrievalmapping* per il riconoscimento, ognuna delle quali ha una colonna dedicata alla chiave esterna per il dettaglio. Lo stesso vale per le tabelle *detailinfoaudio, detailinfovideo, detailinfoimage* e *detailinfotext*. Troviamo poi la tabella del dettaglio (*detail*), che è collegata alla tabella delle opere d'arte (*artwork*), e quest'ultima è collegata all'esibizione (*exhibition*). Data la struttura gerarchica appena descritta, quando un'opera d'arte viene eliminata, viene eliminato anche il dettaglio e tutte le informazioni ad esso collegate. Lo stesso vale per tutte le righe nelle tabelle che si riferiscono ad altri elementi e questa operazione è chiamata **eliminazione a cascata**.
Dopo aver progettato il database, ho inserito tutto all'interno di un software per generare le operazioni di **C.R.U.D.** (Create, Read, Update, Delete), in modo da evitare di dover fare tutto manualmente e così sono passato subito a sistemare i dettagli mancanti per rendere il sistema il più semplice e sicuro possibile nelle mani del curatore del museo.

3.2.1 Database Relazionale

Un database relazionale è un tipo di database che organizza i dati in tabelle strutturate in righe e colonne, i dati sono organizzati in relazioni o tabelle, dove ogni tabella rappresenta una specifica entità o concetto, ogni riga rappresenta una singola istanza o record dei dati, mentre ogni colonna rappresenta un attributo specifico dei dati.

3.3 Strumenti utilizzati

Per la parte back-end ho dovuto utilizzare diversi strumenti, considerando che dovevo mettere insieme diverse componenti.
Come prima cosa ho utilizzato **StarUML**, un software per creare

diagrammi di flusso, tabelle, schemi, ecc., per disegnare l'Entity Relationship del database. Dopo aver realizzato il diagramma e inserito in tutte le colonne con il tipo, una *Primary Key* e, se necessario, la *Foreign Key* con l'eliminazione a cascata, sono passato a creare tutte le tabelle aiutandomi con il software sviluppato da **JetBrains**, **DataGrip**, che mi permette di eseguire operazioni su tabelle, come creazione, modifica ed eliminazione, tramite un'interfaccia grafica, senza dover scrivere alcuna riga di codice.
DataGrip, utilizzato per creare il database, era collegato a **XAMPP**, un pacchetto software gratuito e open source che fornisce un ambiente di sviluppo web completo e ottimo per la gestione dei database in locale. XAMPP combina diversi componenti popolari utilizzati nello sviluppo web, inclusi il server web Apache e il sistema di gestione dei database MariaDB/MySQL. All'interno di XAMPP ho dovuto aggiungere un certificato SSL (Secure Sockets Layer), che è un file digitale utilizzato per stabilire una connessione sicura tra un server web e un browser web (*https://*).
Dopo aver creato il database con le tabelle, come anticipato in precedenza, ho collegato a quel database un altro software per la generazione dei C.R.U.D., chiamato **CodeIgniter Wizard**. Questo software semplifica notevolmente la vita di chi lo utilizza, poiché crea autonomamente un'interfaccia grafica minima in pochi minuti, che può poi essere modificata a nostro piacimento. Tuttavia è un software disponibile solo per macOS.
Per quanto riguarda l'IDE utilizzato per la modifica del codice, è sempre stato utilizzato **PhpStorm**.

3.4 Struttura del database

Le tabelle create sono le seguenti(in tutte le tabelle è presente l'ID di ogni elemento che è formato da un intero):

- **exhibition**: tabella per gestire tutte le esibizioni che sono nel museo, abbiamo un campo per il nome, uno per il tipo di rile-

vamento dei dettagli e tre campi dove viene salvata la posizione del modello da scaricare.

- **artwork**: tabella formata da tutte le opere d'arte presenti nel museo. Al suo interno troviamo un campo per il nome dell'opera, uno per il nome dell'autore, uno per l'URL dove si trova l'immagine e l'ID dell'esibizione corrispondente.
- **detail**: nella tabella dei dettagli troviamo tutti i dettagli di tutte le opere d'arte di tutte le esibizioni. Questa è formata da un campo per il nome del detaglio, uno per l'URL dell'icona del dettaglio e uno per l'ID dell'opera d'arte a cui appartiene.
- **detailinfoaudio, detailinfovideo, detailinfoimage, detailinfotext**: queste quattro tabelle invece mi vanno a rappresentare le informazioni relative al dettaglio all'interno di un opera d'arte. Per ogni singola tabella troviamo un campo per l'URL che mi indica dove è salvato il file, uno per la lingua(fatta eccezione per detailInfoImage che contenendo immagini non ha bisogno di lingue differenti) e uno per l'ID del dettaglio a cui corrispondono.
- **classmapping, objectmapping, retrievalmapping**: le tabelle servono per salvare i dati relativi al metodo di riconoscimento dei dettagli tramite *classification, object detection* e *retrieval*. Qui troviamo un campo per l'Id della rete, uno per la confidenza e uno per l'ID del dettaglio per quanto riguarda classmapping e objectmapping, invece per retrievalmapping ne abbiamo uno per l'ID del dettaglio, uno per la feature e uno per la distanza.

Troviamo lo schema Entity Relationship del database in *figura 3.4*.

3.5 Sviluppo

Entriamo adesso all'interno del codice del back-end. Il codice creato è già abbastanza completo, io ho soltanto apportato alcune

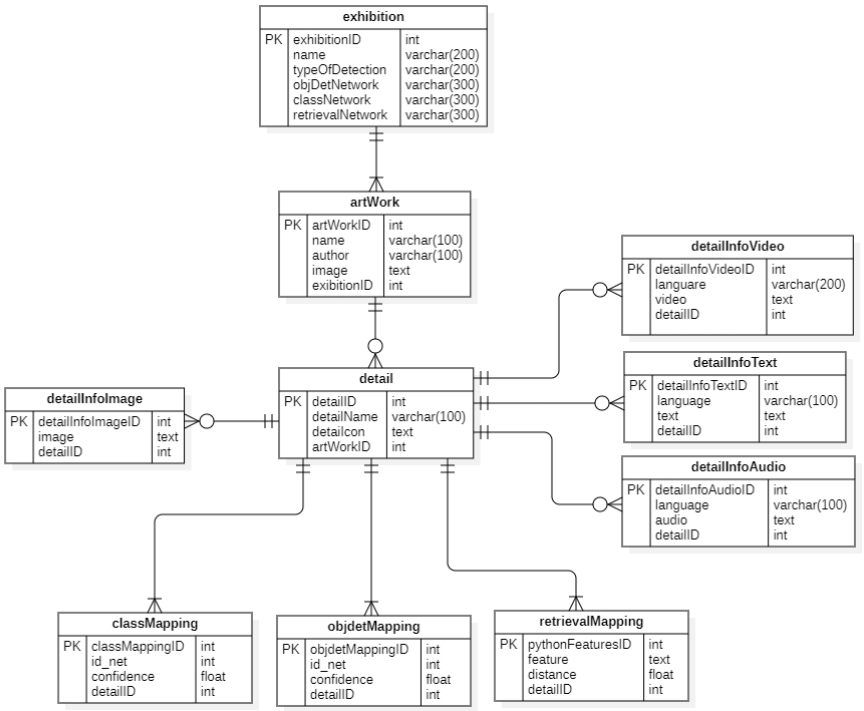

Figura 3.4: Diagramma Entity Relationship del database per SmartLens2 fatto con StarUML

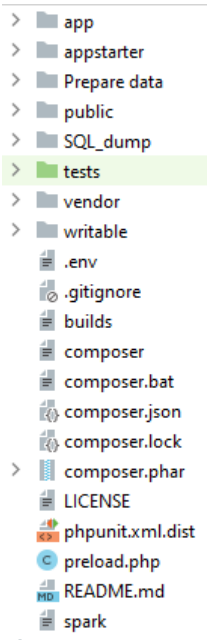

Figura 3.5: Struttura dei file sulla parte Back-End

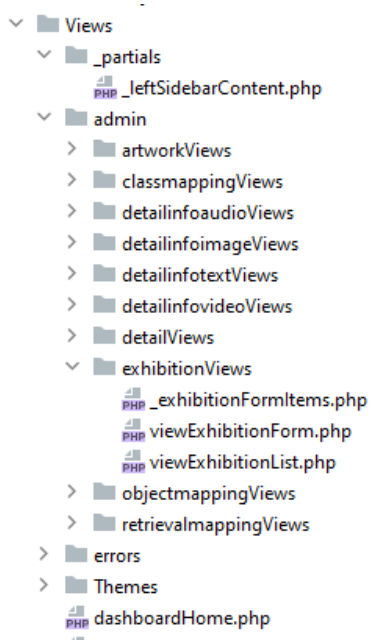

Figura 3.6: Struttura interna della cartella Views contenuta nella cartella App

modifiche estetiche e sulla pagina delle singole tabelle per renderle più solide rispetto al database e ai dati ripescati dal front-end. La parte più complicata all'interno di questi file è stata capire la loro struttura e come venivano utilizzati per poi modificarli. Ho eseguito tutti i passaggi per cambiare la radice (root) del progetto e l'URL che doveva avere il lato back-end, intervenendo sul file `App.php` che si trova in `app/config/App.php`, e sul file `.env` per impostare le informazioni principali (URL dell'app e database) dell'ambiente. Successivamente, mi sono spostato nella cartella `View` e ho iniziato ad aggiungere alcune parti e modificarne altre (nella *figura 3.5* è mostrata la struttura del progetto). La cartella `Views` che si trova all'interno della cartella `app` è strutturata come mostrato nella *figura 3.6*, e contiene altre sottocartelle con file PHP che descrivono la struttura generale dell'interfaccia della pagina web.

Come prima cosa, ho modificato il file `leftSidebarContent.php` in modo da eliminare dalla barra laterale tutte le pagine che il curatore non doveva toccare, ovvero quelle relative ai tre metodi di rilevamento dei dettagli. Inoltre, ho rimosso tali pagine anche dalla

dashboard iniziale, in modo che non fosse più possibile accedervi da nessuna parte per il curatore. Questa modifica è stata semplice, in quanto ho eliminato completamente dall'HTML dei file pertinenti le parti in cui compariva il link al dettaglio, ovvero quelli con il tag `<a>` e un riferimento `href` alla pagina che non volevo fosse modificata dal curatore.

Dopo aver eseguito queste due modifiche, sono entrato in ogni singolo file PHP relativo alle altre tabelle e ho iniziato a modificarli. Come anticipato in precedenza, ho avuto la necessità di aggiungere un ulteriore livello di sicurezza, in modo che ogni volta che il curatore carica e salva i dati nel database, nel caso in cui si dimentichi di inserire la chiave esterna, non si verifichino problemi nel front-end. Per risolvere questo problema, avevo almeno due possibilità: la prima sarebbe stata visualizzare un messaggio di avviso a schermo, spiegando il problema al curatore; la seconda, che ho effettivamente utilizzato, è stata quella di modificare in modo interattivo, all'interno del codice CSS tramite JavaScript, il campo `display` da `none` a `block` e viceversa, facendo apparire e scomparire i blocchi delle informazioni.
Passando a modifiche meno rilevanti, ho modificato il logo della pagina mettendo al posto di quello di CodeIgniter quello di SmartLens2 e poi ho fatto in modo che il titolo e il logo che sono posti sulla la barra laterale, quando questa viene chiusa, rimangono presenti in alto a destra nell'header della pagina. Per la prima modifica mi è bastato cambiare l'URL dell'immagine del logo a cui era collegata e per la seconda mi è bastato andare a modificare il codice CSS del titolo rendendolo visibile sull'header.

Conclusioni

Per concludere facciamo delle piccole considerazioni per quanto riguarda il progetto ottenuto.
Il processo che ci ha permesso di arrivare ad avere SmartLens2 è stato abbastanza lungo, sia per quanto riguarda la fase di progettazione e sviluppo sia anche per la fase di apprendimento. L'applicazione web finale però è perfettamente funzionante, semplice ed efficace sia per quanto riguarda il visitatore che per quanto riguarda il curatore del museo. Avendo una web application ho dovuto fare i conti anche con il trasporto del formato in qualcosa di adattabile ad uno schermo ristretto di uno smartphone e quindi rimodellare tutta l'applicazione dato che la parte back-end si può utilizzare su pc ma spesso potrebbe essere utilizzata, per far prima, su smartphone e lo stesso vale per la parte front-end, la quale però verrà utilizzata quasi esclusivamente su smartphone.
All'interno della struttura manca ancora una parte molto importante, ovvero quella che permette di effettuare l'allenamento della rete dopo che gli sono state fornite tutte le informazioni, l'ideale sarebbe utilizzare un qualcosa che prendendo un set di immagini ne riesce a ricavare i dettagli in modo automatico ma dare anche la possibilità al curatore di andare a segnare nuovi dettagli non riconosciuti all'interno dell'opera d'arte o anche eliminarne qualcuno che è stato rilevato dalla rete in automatico ma non è di alcun interesse. In questo caso, aggiungendo l'allenamento automatico, avremmo che l'applicazione funzionerebbe automaticamente e senza bisogno di un allenamento manuale da parte del programmatore come invece avviene adesso e

il curatore avrebbe quindi la possibilità di inserire quanti dettagli vuole all'interno dell'opera d'arte, selezionandoli da solo e facendo partire il riconoscimento per poi aver aggiunto un nuovo modello allenato all'interno dell'applicazione.
Durante lo sviluppo di tutto il progetto ho potuto imparare ad utilizzare molto meglio di quanto sapevo fare il JavaScript, PHP, HTML e CSS, dato che l'applicazione si basava esclusivamente su questi linguaggi. Lo studio del database per progettarlo dall'inizio mi ha dato maggiore consapevolezza di come questi funzionano, di come sono fatti e degli strumenti necessari per utilizzarli. Ho potuto utilizzare nuovi strumenti come per esempio CodeIgniter che prima non sapevo come funzionasse ampliando anche su questo aspetto nuove conoscenze sempre utili in questo ambito.

Ringraziamenti

Vorrei adesso ringraziare tutti coloro che mi sono stati vicini e mi hanno supportato durante tutto il percorso di studio.

Prima di tutto mi faccio i complimenti da solo per la forza di volontà, la voglia e la determinazione che ho impiegato per portare a termine questo percorso. Grande Giammi!!

Adesso vorrei ringraziare i miei genitori, Massimo e Laura e mio fratello Antonio che mi hanno dato sempre un sostegno(economico e morale) costante e fiducia, durante tutta la durata della triennale(quadriennale) a Firenze. Ringrazio poi tutto il resto della mia famiglia per il supporto dato durante questi anni e in particolare partendo dai nonni Giovanni, Franco, Anna, Doriana e Severina(Melisenda), gli zii Enrico, Ilaria e Alessandro e i cugini Giovanna e Vittorio.

Ringrazio poi tutti gli amici che mi sono stati vicini e hanno contribuito a sostenermi e rendere meno faticoso questo percorso di studi, tra tutti ringrazio particolarmente Elia, Matteo(il Taddy) e Ludovico.

Un ringraziamento va al professore/relatore Marco Bertini che mi ha seguito durante tutti questi mesi per portare avanti il progetto, facendomi avvicinare e appassionare a questa branca dell'Ingegneria Informatica.