Relazione progetto Tecnologie Web

Anno Accademico 2017-2018

careghetole

Referente gruppo: Nodari Leonardo (leonardo.nodari@studenti.unipd.it)

Indirizzo sito: http://tecweb.studenti.math.unipd.it/lnodari/

Gruppo composto da:

- Nodari Leonardo 1123441
- Pozzan Paolo 1121339
- Rebellato Elia 1122639
- Travasci Gianluca 1120260

Login utente amministratore:

- Username: admin
- Password: admin

Login utente semplice:

- Username: user
- Password: user

Indice

Abstract

L'idea alla base di CareghеTole è quella di sopperire alla mancanza, nel mercato e-commerce, di una azienda che venda arredamento in territorio europeo. Lo scopo del sito è quello di offrire all'acquirente un'ampia scelta di mobilio, di alta qualità, a prezzi ragionevoli. Le categorie di oggetti proposti sono principalmente: divani per il soggiorno, tavoli e sedie per la cucina. L'utente, volendo, potrà creare un account personale e, tramite questo, aggiungere al carrello gli oggetti desiderati e, successivamente, effettuare l'ordine.

Accessibilità

Il sito sviluppato rispetta gli standard W3C, la separazione tra struttura, presentazione, comportamento e le regole di accessibilità per rendere il sito fruibile da qualsiasi tipo di utente e dispositivo. La struttura è realizzata tramite file HTML 5. Questi ultimi utilizzano fogli di stile in CSS e script realizzati tramite JavaScript che gestiscono la parte di comportamento. Questi script sono realizzati in modo da garantire una elegante trasformazione, al fine di migliorare l'esperienza utente, senza ridurre le capacità del sito in caso JavaScript fosse disattivato.

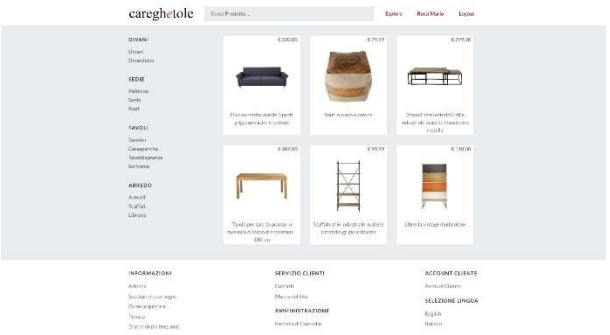
Si è tentato di rendere il sito accessibile anche per persone con disabilità visiva. É infatti navigabile tramite l'uso di screen reader: tutte le immagini presenti sono dotate di un campo alt, che descrive il contenuto dell'immagine, è possibile viaggiare nel menù tramite l'utilizzo del Tab ed è stato inserito, all'inizio della pagina, un link che permette di saltare la parte di menù e andare direttamente alla parte di contenuto della pagina, rendendo più veloce la navigazione.

È stato scelto, poi, di non utilizzare il tag "accesskey" nei link per evitare eventuali conflitti che si potrebbero avere fra utenti che utilizzano diversi software con accesskey già preimpostate.

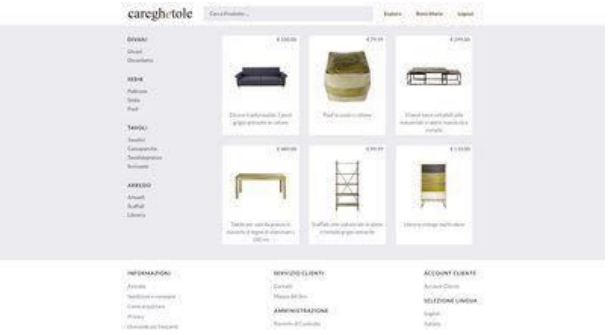
Si è cercato di utilizzare uno schema di colori ottimale, per garantire una facile lettura del contenuto anche a persone affette da varie tipologie di daltonismo (ciò non può essere garantito completamente per le immagini raffiguranti i prodotti).

I link, inoltre, sono di un colore diverso rispetto al testo normale, per favorirne il riconoscimento.

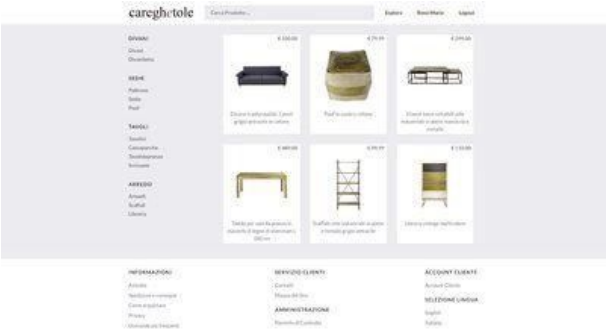
Seguono alcuni screenshot della homepage vista da utenti affetti da varie forme di daltonismo effettuati tramite l'utilizzo del sito vischeck.com.



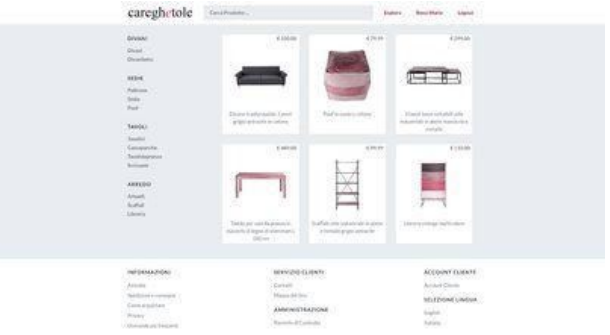
Pagina originale



Deuteranopia



Protanopia



Tritanopia

Usabilità

La homepage del sito risponde alle seguenti delle sei W:

- Where: Avendo progettato una homepage contenente direttamente i prodotti e le categorie viene mostrato all'utente che visita il sito per la prima volta ciò che il sito offre;
- Who: Se l'utente che accede per la prima volta al sito desidera conoscere maggiori informazioni riguardo careghetole può trovarle nel footer dove sono presenti i link che forniscono i dettagli del sito e dell'azienda;
- Why: Nella prima versione del sito non vi è risposta ma si potrebbe facilmente aggiungere alla homepage i link ai prodotti al momento in offerta, oppure l'aggiunta di banner che garantiscono sconti o spedizioni gratuite sopra una determinata cifra;
- What: Un utente giunto nella homepage si fa un'idea delle tipologie di prodotti offerti guardando i link del menù di navigazione;
- When: Nella homepage vengono riportati dei prodotti selezionati dall'amministratore, questi possono essere ad esempio gli ultimi aggiunti al catalogo;
- How: La barra di navigazione laterale permette all'utente di navigare fino alle pagine delle singole categorie di arredamento offerte dal sito, è inoltre presente una barra di ricerca che permette una ricerca più specifica.

Scelte Strutturali

Per la realizzazione del sito si è voluto mantenere uno stile minimalista. Si è evitato quindi colori troppo appariscenti, testo lampeggiante o in movimento, ecc. È stato inoltre scelto di rimuovere il menù dalle pagine interne del sito e di mantenerlo solamente per la homepage in modo tale da avere un design più pulito. Questa scelta non compromette il facile utilizzo del sito in quanto è comunque presente la barra di ricerca.

Per la stampa si è scelto di rimuove header e footer, inoltre è stato scelto di lasciare il logo, altrimenti non si capirebbe di che sito si tratta, e di lasciare il menù laterale nel caso in cui un utente volesse stamparsi le varie categorie di oggetti offerte dal sito.

Per quanto riguarda la navigazione da smartphone (e phablet) si è deciso di non mostrare il menù in modalità portrait in quanto occuperebbe una quantità elevata di spazio. Passando alla modalità landscape il menù viene normalmente mostrato. In tutti gli altri dispositivi (esclusi alcuni tablet) il menù compare sempre nella homepage.

Interazione dell'Utente

Un utente che giunge nel sito è in grado di navigare tranquillamente e, quando trova qualcosa di suo interesse, può effettuare il login, se già possiede un account, oppure registrarsi al sito inserendo i propri dati. I dati richiesti per la registrazione sono l'essenziale per poter ricevere direttamente a casa, viene richiesto l'inserimento di un indirizzo e-mail e la creazione di una password che permettono l'accesso all'area personale e l'indirizzo a cui gli acquisti devono essere spediti. I dati inseriti dall'utente sono salvati sul database MySQL.

L'amministratore del sito invece può, tramite dashboard, creare nuove una nuova pagina prodotto dove deve inserire un'immagine, una descrizione e un prezzo e successivamente pubblicarla in modo tale da renderla disponibile a tutti gli utenti del sito. Per effettuare modifiche è necessario accedere come utente amministratore, nel footer quindi comparirà la voce "Pannello di controllo", premendo qui si entrerà nella parte riservata all'amministratore. Subito si potrà vedere alcune statistiche del sito come il profitto, andando su "Prodotti" si potrà modificare prodotti già esistenti o aggiungere nuovi oggetti al catalogo. Andando invece su "Ordini" si potrà controllare o modificare lo stato degli ordini in corso.

Problemi e Limitazioni Conosciute

Il sito è stato sviluppato inizialmente in lingua italiana ma nel footer è presente anche un link per andare ad una versione del sito in inglese. La traduzione però è stata lasciata incompleta a causa delle scadenze della consegna. È comunque possibile aggiungere nuove lingue, nel caso in cui il sito voglia espandersi in nuovi paesi, o modificare quelle già presenti.

La dashboard di amministrazione, tenendo conto del fatto che un sito è solitamente amministrato da computer e non da smartphone, non è stata sviluppata in modo ottimale per essere utilizzata da dispositivi mobile.

Su mobile (portrait) la pagina degli ordini effettuati sfora leggermente il layout.

Specifiche Tecniche

Package Manager

Per accelerare lo sviluppo e per assicurarsi che tutti i requisiti software fossero aggiornati e mantenuti ad una versione comune a tutti i membri sono stati scelti dei Package Manager. Questi software sono richiesti anche per l'installazione dell'applicativo in produzione.

- PHP - Composer
- Javascript - NPM + Yarn

Supporto allo Sviluppo

Per semplificare e uniformare lo sviluppo sono stati scelti alcuni software, tra i quali: Macchine Virtuali, Linters. Questi software non sono necessari per pubblicare l'applicativo in produzione, ma sono stati inclusi nella relazione per completezza.

Vagrant + Laravel Homestead

Vagrant è un sistema di gestione di macchine virtuali, permette di creare un sistema di sviluppo costante e comune tramite l'utilizzo di macchine virtuali (VirtualBox, VMWare, ecc).

Laravel Homestead è una macchina preconfigurata adibita allo sviluppo di applicativi PHP; mette a disposizione uno stack LEMP (e molto altro) configurabili tramite uno script YAML.

PHP CS Fixer

PHP CS Fixer è un software da riga di comando che controlla ed eventualmente sistema lo stile del codice PHP. Questo permette ai programmatori di non doversi preoccupare dell'allineamento, indentazione o anche di pacchetti importati e non utilizzati, in quanto il tool li rimuove in automatico.

Si è scelto di seguire le linee guida PSR-2.

ESLint

ESLint la contro parte Javascript di PHP CS Fixer.

Si è scelto di seguire le linee guida di Google.

PHP Autoloader - Composer

Composer è stato scelto anche perché mette a disposizione un autoloader per applicativi PHP basato su PSR-4. Questo permette di scrivere codice molto più elegante e stabile, evitando di raggruppare e disseminare i propri file di istruzioni `require` o `include`; lo stile finale si avvicina molto al sistema di gestione dei pacchetti di Java.

SASS

Per poter gestire più facilmente e ordinatamente fogli di stile, si è fatto uso di SASS. Tra i vari benefici, permette la personalizzazione di Bootstrap tramite variabili, il caricamento di altri fogli di stile (dividendo in maniera logica il codice) e la possibilità di scrivere in maniera "ramificata" (semplificando la lettura e la complessità dei selettori).

Build System

Per poter utilizzare SASS e per minimizzare il codice prima di pubblicarlo, è stato scelto di utilizzare un build system. La scelta è ricaduta su Webpack, un build system scritto in Javascript di ultima generazione. Per semplificare ulteriormente il suo utilizzo si è scelta la libreria Laravel Mix, che offre un layer di astrazione alle configurazioni di Webpack più che sufficiente ai fini di questo progetto.

Inoltre sono stati creati degli shortcut per npm in modo da eseguire rapidamente i comandi essenziali:

- `npm run dev`: Compila gli asset in modalità sviluppo, quindi rapidamente e senza minificazione
- `npm run watch`: Compila gli asset in modalità sviluppo, ma rimane in ascolto per eventuali modifiche (Live Compiling)
- `npm run prod`: Compila gli asset in modalità produzione, concatenando e compattando i fogli di stile e gli script

PHPdotenv

PHPdotenv è una libreria PHP che permette di caricare le proprie configurazioni da un file `.env` e utilizzarle all'interno dell'applicativo come fossero variabili d'ambiente. Non è pensato per l'utilizzo in produzione e normalmente dovrebbe essere affiancato da un layer di cache, ma visto che si tratta di un prototipo si è deciso di evitare la complessità aggiuntiva ed utilizzare la libreria as-is.

Bootstrap

Bootstrap è forse il framework frontend più conosciuto e utilizzato sul web. Proprio per questo motivo si è scelto di utilizzarlo il meno possibile, solamente per i componenti meno estetici e di utilità (allineamento, griglie, ecc) per non reinventare la ruota e realizzare tutto il resto dell'interfaccia a mano. Tutti i colori e alcuni effetti sono stati modificati per essere più "personali".

Inoltre si è scelto di evitare completamente tutto l'aspetto "interattivo" di Bootstrap, ovvero tutti i componenti che richiedono moduli Javascript. Questo principalmente per due ragioni: jQuery è una dipendenza necessaria per tutti questi moduli ed è considerata antiquata e troppo pesante; per avere una buona accessibilità e compatibilità si rendono necessari ancora più moduli esterni.

Tra i moduli di stile utilizzati indichiamo:

- Reboot
- Immagini
- Griglie
- Tabelle
- Bottoni (personalizzati)
- Paginazione
- Alert

Compatibilità

Backend

L'applicativo è stato scritto per PHP 7.0 su ambiente Linux, utilizzando come database relazionale MySQL. Inoltre si fa utilizzo delle sessioni native di PHP. Molti componenti sono stati astratti e migrare da un RDBMS ad un altro o ad uno store di sessioni diverso (Redis, Database, Memcache, ecc) non richiederebbe la modifica di tutta la logica applicativa, ma non sono stati realizzati "adattatori" alternativi.

Frontend

Il frontend è stato realizzato per supportare più del 90% dei browser in circolazione (basati sulle statistiche italiane) basandosi sul sito CanIUse. La componente più "recente" utilizzata (estensivamente) è Flex Box, supportata dal 93.45% dei browser in Italia. Non si garantisce la compatibilità con Internet Explorer in quanto la sua implementazione di Flex Box è ricca di bug, tuttavia il sito dovrebbe essere utilizzabile anche da IE nonostante la presenza di alcuni glitch.

Sprinkle some Javascript

Javascript è stato considerato un abbellimento del sito, e strettamente non necessario. Viene utilizzato per semplificare l'ordinamento dei prodotti durante la ricerca, permettendo l'auto-submit di un form, e per migliorare le "bounding box" di alcuni link (come il menu e le card dei prodotti) e renderli più facilmente cliccabili.

E' stato scelto quindi un approccio "enhance", il sito è completamente funzionante senza Javascript e mostra un determinato stile. Quando il codice Javascript esegue va ad aggiungere la classe enhanced ai componenti specifici, permettendo quindi uno stile più accurato (hover, cursor pointer, ecc) di quest'ultimi e migliorare l'esperienza utente.

MVC

L'architettura del backend segue il paradigma Model-View-Controller, basato suoi più popolari web framework disponibili (Ruby on Rails, Express, Ember, Laravel, Yii, ecc).

Struttura delle cartelle

L'applicazione presenta una struttura fatta nel seguente modo:

```
.
├─── app          # Business logic dell'applicazione
├─── database     # Dump SQL del database e struttura
├─── framework    # Codice del framework
├─── node_modules       # Dipendenze di NPM
├─── public             # Web Root
├─── resources    # Assets, Immagini e Views
├─── storage      # Storage dinamico
└─── vendor             # Dipendenze di Composer

8 directories
```

Framework

```
framework
├─── Auth              # Componenti per la gestione degli utenti
├─── Controllers       # Struttura base di Controllers e Middleware
├─── Database          # ORM per la gestione dei Model
├─── Helpers                 # Set di helpers globali
├─── Localization      # Gestione delle traduzioni e multi-lingua
├─── Request                 # Astrazione delle richieste e sessioni
├─── Responses         # Astrazione delle risposte
├─── Router            # Router per le richieste
└─── Views             # Sistema di rendering delle view

9 directories
```

Router

Il sistema di routing è alla base del funzionamento dell'applicazione. Tutte le richieste vengono inoltrate tramite mod_rewrite allo script index.php, che passa la richiesta al Router per determinare quale Controller dovrà "rispondere".

Per fare ciò al Router viene "caricato" un file contentente le configurazioni in base a path e HTTP method.

A seconda della configurazione `PRETTY_LINKS` verranno utilizzati due formati per l'URL:

- `PRETTY_LINKS=false` `/subdirectory/index.php?r=/product&id=3`
- `PRETTY_LINKS=true` `/subdirectory/product?id=3`

Il primo ha il vantaggio di non richiedere configurazioni aggiuntive ed è abilitato di default. Non ha bisogno di `mod_rewrite` e funziona nativamente su qualsiasi sottocartella si vada a posizionare l'applicativo. Però è molto brutto da vedere, soprattutto perché molti caratteri vengono "url encoded" e il risultato finale è difficilmente comprensibile.

Il secondo è molto più in linea con le moderne applicazioni. Tuttavia richiede un sistema di riscrittura delle richieste (come `mod_rewrite`) e una configurazione manuale (tramite la voce `SUBDIRECTORY`) della cartella di destinazione.

ORM

Per semplificare le interazioni con il database e per evitare i problemi più comuni di distrazione (SQL-Injection) è stato sviluppato un ORM (Object-relational mapping). Questo ha portato alla scrittura di codice molto più espressivo, di cui riportiamo un esempio per tipo.

Insert:
```
$category = new Category($input);
$category->save();
```
Update:
```
$category = Category::find($id);
$category->setAttribute($input);
$category->save();
```
Select:
```
Category::where('parent_id', null)->orderBy('name', SORT_ASC)->all();
```
Delete:
```
$category = Category::find($id);
$category->destroy();
```

Controllers e View

Il sistema di Controller è l'"output" del Router, che identifica una classe Controller e un corrispondente metodo a cui delegare la risposta per l'azione corrente. Ad esempio alla richiesta `POST /product?id=3` può corrispondere la coppia classe-metodo `ProductController::update()`.

Il controller deve rispondere o con una stringa contenente il codice da ritornare, o per maggior comodità un istanza di `Response`.

Le seguenti istanze di `Response` sono disponibili e hanno un corrispettivo helper nella classe base Controller:

- `BaseResponse`: ritorna direttamente il codice da mostrare
- `RedirectResponse`: reindirizza l'utente ad un altra pagina
- `ViewResponse`: renderizza una view con i parametri passati

Un Controller ha inoltre a disposizione un riferimento all'oggetto `Request` contenente tutti i parametri necessari (GET, POST, ecc).

Una View viene identificata dalla "dotted-version" del suo path rispetto alla cartella `resources/views`. Ovvero la view `admin.orders.index` corrisponderà al file `resources/views/admin/orders/index.php`.
Ad una View può essere passato un array associativo contenente le variabili che poi avrà a disposizione. Ovvero se io richiamo una View passando come parametro

`["something" => "abc123"]` la View avrà a disposizione una variabile `$something` con il valore `"abc123"`.

Localizzazione

E' stato realizzato un sistema rudimentale di traduzione e selezione della lingua, che permette di tradurre l'applicazione in un numero potenzialmente infinito di lingue. Il sistema è composto da una sola classe, `Translator`, e da un helper globale `trans()`.

Le traduzioni sono organizzate in files nelle cartelle `resources/lang/{LANG_CODE}`. Ogni file restituirà un array associativo a cui ad ogni identificativo corrisponderà il valore tradotto, in questo modo all'interno delle view è possibile scrivere semplicemente `trans("orders.my_orders")` e la stringa risultante corrisponderà a quella della lingua selezionata dall'utente.

E' inoltre messo a disposizione un sistema di "bindings" delle lingue, la funzione `trans()` accetta come secondo parametro un array associativo contenente dei valori da sostituire.
Per esempio, data la chiamata `trans("order.order_title", ["id" => 12])`, se il contenuto della stringa tradotta è `"Ordine #:id"`, il risultato finale sarà `"Ordine #12"`.

Flash Message/Input

Per mantenere una separazione logica tra le funzioni, invece di ritornare la View nel caso un'azione venisse rifiutata, si è preferito reindirizzare l'utente alla pagina di partenza. Questo presenta un problema tecnico: senza ulteriori strutture, non è possibile mostrare all'utente i dati inseriti in precedenza o un messaggio di errore. Per ovviare a questo problema è stato creato un sistema di messaggi basato sulla sessione.

Quando si vuole reindirizzare un utente al form iniziale, gli viene agganciato alla sessione un array contenente tutti i dati inseriti in precedenza ed eventualmente un messaggio di errore/successo. Questo array viene preso e rimosso alla richiesta successiva e mostrato all'utente. Per facilitare la cosa viene inoltre fornito un helper `old($key, $value)` che mette a disposizione i dati precedentemente inseriti (ora salvati in sessione) per ripopolare il form.

Vengono inoltre forniti i seguenti helpers per reindirizzare con l'input e un messaggio di errore (all'interno di un Controller):

```
return $this->redirect('/admin/categories')
            ->withFlash([
                 “error” => “some_error_message”,
            ])
            ->withInput();
```

Middleware

Un Controller può dichiarare uno o più Middlewares da essere eseguiti prima della richiesta effettiva. Questi possono anche interrompere il “flusso d'esecuzione” senza che la richiesta giunga al Controller.

L'utilizzo principale è il controllo dei permessi, assegnare a tutti i Controllers di richieste d'amministrazione un Middleware che interrompa l'utente qualora non fosse autorizzato ad eseguire quell'azione.

Applicazione

```
app
├─── Controllers       # Controllers
├─── Data              # Dati di supporto
├─── Middlewares       # Middleware per le richieste
├─── Models            # ORM Models
├─── Queries                 # Helpers per Query complesse
├─── routes.php        # File di routing
└─── view_data.php           # Variabili globali a tutte le Views

5 directories, 2 files

resources
├─── assets            # Assets (JS, Styles, Images)
├─── lang              # File di localizzazione
└─── views             # Views

3 directories, 0 files
```

Il funzionamento di base dell'applicazione è uguale ad un normalissimo e-commerce, per cui nelle seguente sezione verranno trattate solo le decisioni opinabili o che hanno richiesto più tempo nella progettazione. Tutte le altre verranno eventualmente solamente accennate.

Amministrazione

E' disponibile una pagina di amministrazione all'indirizzo `/admin`, con la possibilità di modificare categorie, prodotti e ordini. Inoltre fornisce una primitiva panoramica dei guadagni attuali.

Il controllo degli accessi è effettuato tramite un Middleware chiamato AdminOnly, che verifica che l'utente sia effettivamente un amministratore prima di procedere con la richiesta. In caso contrario un errore 403 è restituito.

Storage

Per permette agli amministratori di caricare le immagini dei propri prodotti è stato realizzato un sistema di storage e upload di file. La funzione di upload è molto elementare e basilare, quindi non verrà trattata.

Le immagini sono salvate in una cartella storage esterna alla web root. Questa scelta è stata causata dal desiderio di utilizzare la cartella storage anche per cache di codice PHP (da tenere fuori dalla web root). Nonostante questa funzione non sia mai stata implementata, la struttura iniziale è rimasta tale.

Per servire le immagini sono stati pianificati due sistemi, in base alle funzioni messe a disposizione dal web server.

Direct Storage

Abilitato configurando `DIRECT_STORAGE=true`, è il metodo più prestazionale e pulito di servire i contenuti dello storage agli utenti. Consiste nel creare un symlink chiamato storage all'interno della cartella public (document root del web server) che punti alla cartella storage nella root del progetto, in questo modo sarà il web server stesso a servire le richieste.

Reflected Storage

Non essendo sempre possibile utilizzare un symlink per servire il contenuto, è stato scritto uno `StorageController` che legga le immagini dal disco e le serva. Questo meccanismo è più inefficiente del precedente, ma è sempre disponibile ed è stato considerato un fallback accettabile.

N+1 Problems

Vista la natura basilare dell'ORM realizzato per questo progetto, vi sono vari casi in cui si può notare il cosiddetto "n+1 problem". Ovvero viene preso dal database un insieme di oggetti, e per ogni oggetto si accede ad un oggetto collegato; in questo modo lo script eseguirà una query al DBMS alla prima richiesta e poi una query per ogni oggetto collegato, che si sarebbero potute evitare recuperando in blocco tutti gli oggetti collegati.

La realizzazione di un sistema di "Eager Loading" è stata considerata eccessiva.

Sitemap

E' stata realizzata una Sitemap XML disponibile all'indirizzo `/sitemap.xml`.

L'host del sito deve essere configurato tramite il parametro `HOST`.

Istruzioni per l'installazione

Si è preferito valorizzare stabilità ed efficienza dell'applicativo, tuttavia la questo ha pesato sulla semplicità d'installazione. Le istruzioni che seguono cercano di essere il più lineari possibili per ottenere un ambiente di produzione stabile.

Prerequisiti

Per lo scaricamento dei pacchetti richiesti sono necessari i seguenti software: Composer, NodeJS e NPM, Yarn.

Si rimanda ai rispettivi siti per l'installazione.

Preparazione

Dopo aver scompattato l'archivio consegnato nella cartella desiderata, spostarsi con il terminale alla root del progetto ed eseguire i seguenti comandi per installare le dipendenze richieste:

```
composer install --no-dev
yarn --prod
```

Se il download avrà successo, si potrà notare la creazione delle cartelle `node_modules` e `vendor`.

Per compilare tutti gli assets eseguire il seguente comando:

```
npm run production
```

Configurazione

Copiare il file `.env.example` e chiamarlo `.env` .

Modifica il file con le configurazione desiderate. Per un setup di base è sufficiente modificare utente, password e nome del database.

Se si vuole pubblicare il sito, per il corretto rendering della Sitemap è necessario aggiustare il campo `HOST`.

Database

Se si vuole un database con solamente la struttura e nessun valore, caricare il file `database/structure.sql`.

Se si preferisce avere anche i dati d'esempio, importare il file `database/dump.sql`. Questo file è stato modificato a mano per essere compatibile con MySQL 5.6, altrimenti è necessaria la versione 5.7 (o compatibile).

Per importare le immagini, copiare il contenuto della cartella `database/seed/images` all'interno della cartella `storage/images`:

```
cp database/seed/images/* storage/images/
```

Il sito ora dovrebbe essere pienamente funzionante. Ulteriori istruzioni sono disponibili di seguito per le funzioni aggiuntive.

Pretty Links

Per abilitare le url decenti, è sufficiente cambiare le configurazioni nel seguente modo (considerato il sito installato nella sottocartella lnodari):

```
PRETTY_LINKS=true
HOST=http://tecweb.studenti.math.unipd.it
SUBDIRECTORY=/lnodari/
```

Si sottolinea che il parametro SUBDIRECTORY deve iniziare e terminare con uno slash.

Fast Storage

Per abilitare lo storage più efficiente è necessario creare un symlink storage all'interno della cartella public che punti alla cartella storage del progetto:

```
cd public/ && ln -s ../storage storage
```

Possiamo controllare ora se è possibile accedere ad un immagine, per esempio: http://tecweb.studenti.math.unipd.it/lnodari/storage/images/divaniletto4.jpg

In caso affermativo, è possibili abilitare la seguente configurazione:

```
DIRECT_STORAGE=true
```

Server TecWeb

Si ritiene opportuno specificare una particolarità dell'installazione messa a disposizione all'indirizzo http://tecweb.studenti.math.unipd.it/lnodari/.

Vista la struttura dell'applicazione si riteneva di cattivo gusto trasformare la home dell'utente nella root del progetto e rinominare public in public_html. Per evitare questo l'applicativo è stato installato nella cartella ~/project e la cartella ~/public_html è stata trasformata in un symlink a ~/project/public.

Divisione dei compiti

Il lavoro è stato distribuito come segue:

- Nodari Leonardo: Costruzione del framework e Database;
- Pozzan Paolo: Views e scrittura degli script JavaScript;
- Rebellato Elia: Logica applicativa;
- Travasci Gianluca: Scrittura dei fogli di stile e Immagini.

La divisione qui sopra riportata è puramente indicativa in quanto vi è stata "contaminazione" tra i compiti, è difficile quindi determinare il lavoro svolto dal singolo.

Ognuno ha scritto la relativa parte della relazione.