Laboratorul numarul 1

Baze de date

Dragomir Țurcanu
MI-191

Decembrie 25, 2020

Contents

Sarcina

Deoarece sarcina per-se nu a fost menționată în cadrul aplicației else, o menționez la fel cum este indicată pe website. Deci urmează de realizat însărcinarea în următorul mod.

> PREZENTAREA VARIANTEI FINALE A APLICAȚIEI WEB, COMPONENTA FRONT-END SI BACK-END, PASII 1-7 CE URMA SA FIE PREZENTAT LA LABORATORUL PRECEDENT

Obiecivele

- Raportul PDF
- Prezentarea PPTX
- Prezentarea și zip-ul bazei de date

Abordarea Proprie

Pentru a *simplifica* și în același moment de a *dezvolta* ideea sarcinii, am decis să i-au ca bază laboratorul nr 6 la obiectul "Tehnologii Web", deoarece toate conceptele necesare, ba chiar mai multe sunt folosite și explicate.

Deci am mfoodificat un pic sarcina, pentru a satisface condiția. Varianta lucrării la TW, presupunea crearea unei aplicații de monitorizare și selectare a datelor referitoare la cursul valutar. Și sună în următorul mod.

> De creat baza de date cu datele despre cursurile valutelor pe fiecare zi. De realizat interfaţa on-line la baza de date cu următoarele funcţii: selectarea şi vizualizarea datelor în baza cîmpurilor diferite (denumire, data, perioada de timp), indicarea creşterii sau scăderii cursului în perioada perioadă indicată de utilizator.

Rezolvarea

Pentru a realiza sarcina, am decis să folosesc un set de tehnologii ce mi-ar fi interesante în lucru, și benefice pentru dezvoltarea skill-setului. De asemenea acestea au contribuit imens le menținerea interesului în dezvoltare :).

Stack-ul Tehnic

- PHP 7.4
- MySQL 5.6
- Slim Framework [1]
- Doctrine ORM [2]
- Git [3]

Apropos de Git, proiectul este disponibil deschis pe pagina mea proprie GitHub, pentru citire integrală și instrucțiuni de instalare. Accesați aici, sau pe url-ul `https://github.com/dragomirt/lab6tw`.

Concepția

Ideea aplicației este crearea unei interfețe interactive pentru vizualizarea informației. Deoarece caracterul acestei informații este *dinamic*, este logică folosirea tehnologiei **AJAX** [4].

Ținând acest fapt în considerare, am creat o singur loc de intrare, fișierul **index.html**, ce și conține toată partea vizuală, informația fiind încărcată prin requesturile **fetch** realizate de către client, pentru a trage informația din **endpointurile** ale **API-ului** [5] intern.

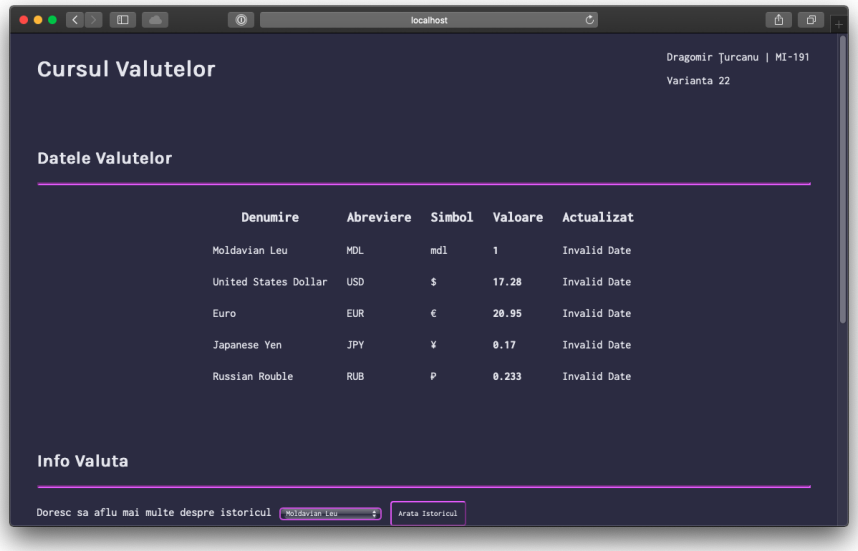
[1] `https://www.slimframework.com/`
[2] `https://www.doctrine-project.org/projects/orm.html`
[3] `https://git-scm.com/`
[4] Asynchronous Javascript and XML
[5] Application Programming Interface

Supranumitul API conține adrese pentru inserarea, modificarea, selectarea și ștergerea informației.

Explicații Tehnice

Front End

Așa deci, odată ce am discutat teoria, este timpul de a demonstra un pic de practică *as well*. Mai jos este demonstrată aplicația în starea sa normală, pe unica pagină grafică a websiteului :D



Pe această pagină putem admira locul primar de interacțiune a unui utilizator de rând cu aplicația în cauză. Aici este posibilă monitorizarea și selectarea valorilor pentru citire.

Majoritatea requesturilor sunt transmise de către Javascriptul din browser în formatul următor.

```
async function getCurrencyData() {
    let response = await fetch("/api/currency");
    return await response.json();
}
```

Fiecare request fiind modificat pentru endpoint-ul dorit. Deci requestul de sus întoarce informații generale despre toate valutele.

Funcția întreagă ce conține requestul arată în modul următor.

```
function getAvailableCurrencies() {
    let currencySelect =
    document.querySelector('select#currencySelect');
    let currencySelectGraph =
    document.querySelector('select#currencySelectGraph');

    if (!currencySelect || !currencySelectGraph) {
        return;
    }

    currencySelect.innerHTML = "";
    currencySelectGraph.innerHTML = "";

    getCurrencyData().then(data => {
        for (const row of data) {
            availableCurrencies[row.id] = {full_name:
            row.full_name, symbol: row.symbol};
            currencySelect.innerHTML += `<option
            value="${row.id}">${row.full_name}</option>`;
            currencySelectGraph.innerHTML += `<option
            value="${row.id}">${row.full_name}</option>`;
        }
    });
}
```

Aceasta ia răspunsul requestului în format **JSON** și înscrie în pagină. La fel sunt realizate celelalte blocuri din pagină.

Baza de date

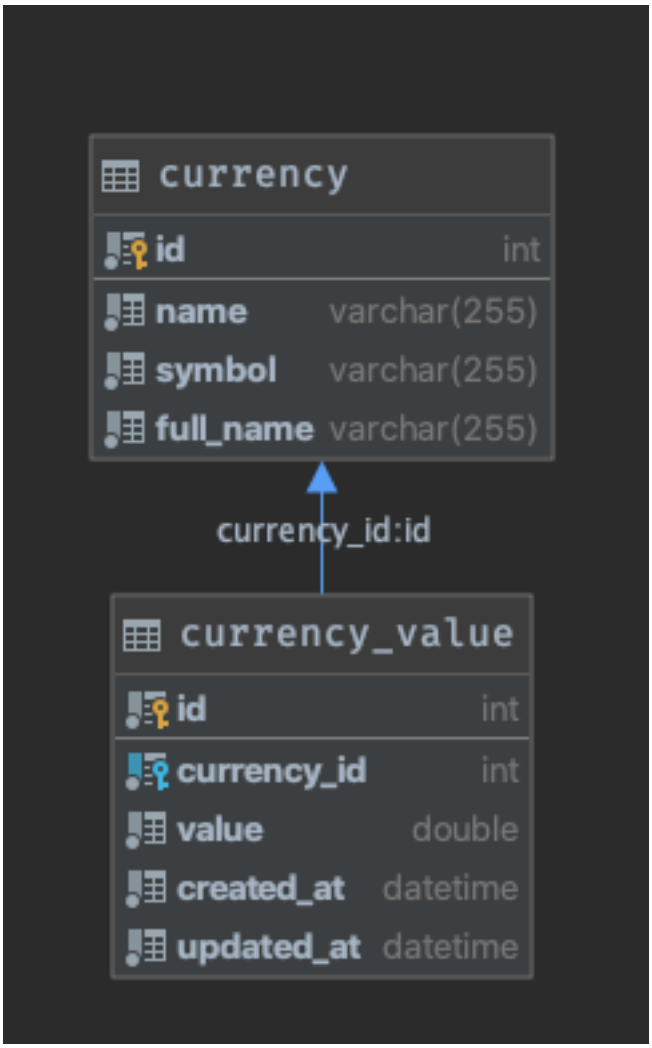
Baza de date în acest exemplu a fost realizată într-un mod maximal simplistic. Există 2 tabele *currency* și *currency_value*.

"currency" presupune un tabel ce conține informație generală despre fiecare valută. Denumirea, simbolul, abrevierea și de genul.

"currency_value" la rândul său conține informație ce ține de valoarea valutei la o dată anumită. Sunt folosite chei străine secundare pentru a indica corelația între aceste 2 tabele.

Există relația OneToMany la *currency.id* spre***currency_value*** și invers, ManyToOne pe aceleași fielduri, pentru a extrage cu ușurință informația în ambele părți.

Tot arată în modul următor.



Back End

Cum deja am menționat, alegerea a căzut pe **Slim Framework**. Aceasta a permis crearea rapidă a rutelor pentru API și descrierea simplă a logicii aplicației.

Spre exemplu, ruta ce întoarce datele despre valori, arată în modul următor.

```php
<?php
// Get all values
    $group->get('/currency/value', function(Request $request,
    Response $response) use (&$entityManager) { // Este funcția
    părinte ce encapsulează o funcție anonimă ce preia
    entityManager pentru a modifica datele cu ajutorul ORM
```

```
        $values = $entityManager->getRepository(CurrencyValue::c
        lass)->findBy([], ['created_at' => 'DESC']); // Sunt
        datele provenite din DB
        if ($values === null) { // Dacă valorile sunt nule,
        sfarșește funcția și întoarce un array gol ca http
        răspuns
            $response->getBody()->write(json_encode([]));
            return $response;
        }

        $responseFormatted = array();
        foreach ($values as $value) { // Pentru fiecare
        înscriere în DB, scrie datele în array-ul decodat pentru
        răspuns
            $responseFormatted[$value->getCurrency()->getId()][]
            = array(
                'value' => $value->getValue(),
                'date' => $value->getCreatedAt()
            );
        }

// Codifică răspunsul în format JSON și întoarce clientului
        $response->getBody()->write(json_encode($responseFormatt
        ed));
        return $response;
    });
?>
```

Mai multe detalii sunt disponibile pe pagina GitHub a proiectului disponibilă mai sus.

Concluzie

Dezvoltarea acestui laborator a contribuit la îmbunătățirea abilităților de creare a aplicațiilor de tip API + AJAX, și a fost un proces total interesant, ce presupun că este reflectat în entuziasmul cu care prezint lucrarea. *Apreciez feedbackul!*