Universitatea Tehnică a Moldovei

Serviciul monitoring valutar

LUCRARE DE AN
la disciplina
BAZE DE DATE

utilizând SGBD MySQL, PHP și stack-ul front-end la alegere
pentru utilizarea potențială de către o agenție specializată.

Dragomir Țurcanu
MI-191

Chișinău 2020

Contents

Conceptul SGBD

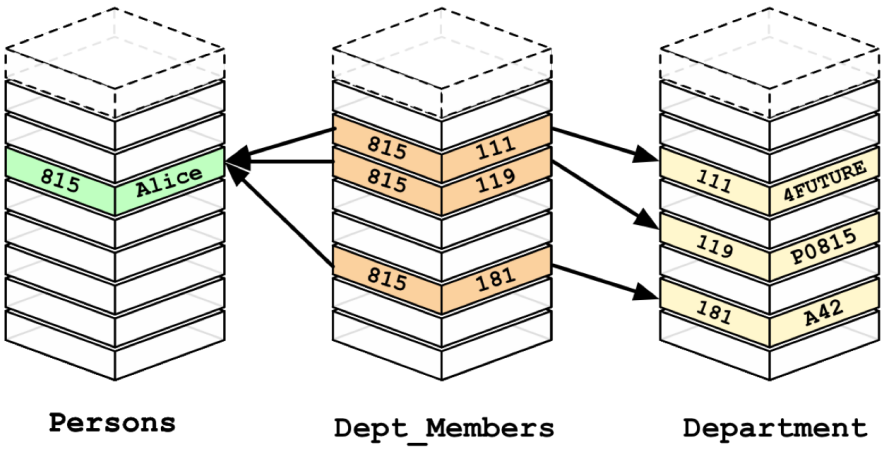
Definiția SGBD

SGBD deabreviat sună ca ***S**istemă de **G**estiune a **B**azelor de **D**ate*. Aceasta este o bază de date digitală bazată pe modelul relațional de date, propusă de către E.F. Codd[1] în 1970. Sistemă softwre folosită la menținerea bazelor de date relaționale este un **SGBD**. Majoritatea sistemelor bazelor de date relaționale folosesc pentru comunicarea internă, interogări și modificări, limbajul ***SQL*** [2].

Baze de date relaționale

O bază de date relaționale se referă la o bază de date ce conține informația salvată într-un mod structurat, folosind *randuri* și *coloane*. Astfel devine simplă localizarea și accesul valorilor în cadrul bazei de date. Este numită *"relațională"* deoarece valorile în fiecare ***tabel*** sunt inter-conectate. Tabelele pot la fel fi conectate către alte tabele. Structura relațională creează posibilitatea de a executa ***operații*** asupra a o multitudine de tabele în același moment.

[1] `https://en.wikipedia.org/wiki/Edgar_F._Codd`
[2] `https://en.wikipedia.org/wiki/SQL`



SGBD în practică

SGBD-urile sunt folosite foarte intensiv în practica de zi cu zi atât a dezvoltatorilor soluțiilor software, atât și de către personalul *data entry*, unitățile de management ale organizației, sau chiar *stakeholder-ii* companiei. Majoritatea sistemelor moderne permit accesarea și vizualizarea datelor în format ușor accesibilă, cu funționalități performante de exportare pentru eventuală analitică folosind instrumente dezvoltate pentru însărcinarea propusă.

Datele pot fi exportate în o multitudine bogată de formate pentru operațiuni diferite cu datele propuse. Exemple exacte sunt ce urmează.

- ***JSON***[3] pentru includerea în aplicații web sau scripturi, perfect pentru dezvoltatorii de soluții softare, ce au nevoie de un format portabil pentru integrarea datelor în ***API***-uri[4] și interfețe vizuale.
- ***XLS***[5], pentru includerea în aplicații de tip *spreadsheet*, de tip Microsoft Excel, sau Google Sheets. Este perfect potrivit pentru managerii sau contabilii unei companii pentru analiza și prognozartea informației pe baza datelor existente.
- ***CSV***[6], perfect pentru integrarea în scripturi și sisteme automatizate, de tipul instrumentariului pentru *machine learning*[7]. Este formatul perfect pentru experții domeniului *data science* ce conlucrează cu dezvoltatorii pentru determinarea **pattern**-urilor în date, și prin urmare exploatarea parametrilor datelor pentru maximizarea profitabilității.

[3] `https://en.wikipedia.org/wiki/JSON`
[4] `https://en.wikipedia.org/wiki/API`
[5] `https://en.wikipedia.org/wiki/Microsoft_Excel`
[6] `https://en.wikipedia.org/wiki/Comma-separated_values`
[7] `https://en.wikipedia.org/wiki/Machine_learning`

Cele mai răspândite sisteme de SGBD la momentul actual sunt următoarele.

- Oracle DB
- MySQL
- PostreSQL
- SQLite
- Microsoft SQL Server
- IBM DB2

Ultimii ani, tot mai populare au inceput să devină SGBD bazate pe baze de date non-relaționale, așa numitul ***NoSQL***[8]. Acestea permit un nivel de flexibilitate a datelor mult mai înalt. Faptul dat este motivat prin lipsa unei structuri bine definite, ce prin folosirea sistemei ***cheie-valoare***.

Lipsa structurii induce o pierdere în performanță, condiționată prin complexititate indexării datelor, dar, beneficiul de bază este posibilitatea modificării formatării, mărimii sau a encodării datelor, ***"on the fly"***[9], ce este foarte benefic pentru o sistemă software în creștere. De aia acest tip de baze de date a devenit foarte popular în cadrul ***startup***-urilor, deoarece această alegere tehnică le permite avansarea rapidă și modificarea datelor fără riscul de a strica datele.

Exemple ale astfel de SGBD sunt următoarele.

- MongoDB
- Redis
- Amazon DynamoDB
- Oracle NoSQL DB

[8] `https://en.wikipedia.org/wiki/NoSQL`
[9] În mișcare

Modelarea Datelor

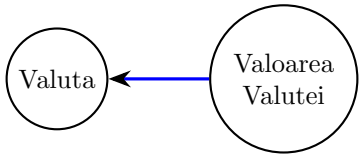
Domeniul de studiu

Un concept bine cunoscut oricărui investor, economist profesional, ba și unei persoane de rând, este următorul fapt: banii, bine-investiți aduc profit. Prin urmare, o persoană co-interesată, este întâlnită cu o multitudine de opțiuni pentru a o obține rezultatul dorit. În joc apar investițiile imobiliare, hârtii de preț, obiecte de epocă, dar ce-a mai răspândit în spațiul R.M., este cu siguranță inveția în valută. Diversificarea portfoliului valutar, aduce o încredere în viitorul apropiat și chiar îndelungat, fiind amplificat pe spațiul CSI de către pierderile usturătoare din anii 90'.

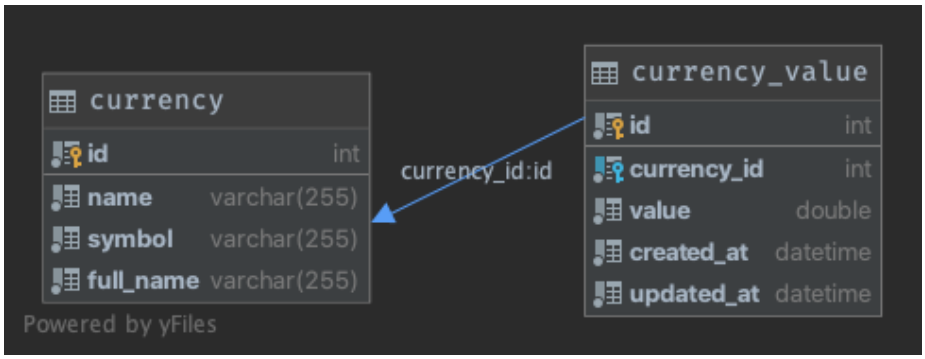
Ca și orice serviciu bazat pe cantitatea și calitatea datelor, alegerea valutei, monitorizarea momentului optimal de cumpărare și vânzare și calcularea tendenței de creștere și scădere a valutei este o necesitate vitală pentru orice ***investor valutar****.*

Exact pentru acest scop a fost luată decizia să se creeze un sistem ușor-accesibil, bazat pe o interfață reactivă, ce ar permite o interacțiune cu fricție minimală, pentru a îndruma utilizatorii să colecteze toată informația necesară în timpul minimal, luând în considerare ritmul **supra-accelerat** a vieții cotidiene.

Modelul Logic



Probabil modelul logic al aplicației curente pare foarte simplistic, dar secretul în cazul dat este faptul, ca acesta își îndeplinește misiunea într-un mod rapid și pragmatic. În continuare prezint machetul bazei de date.



În cazul dat, toată structura aplicației presupune 2 tabele, **currency** și **currency_value**, primul ce conține datele despre valutele disponibile, și al doilea tabel respectiv, despre valorile fiecărei valute într-un moment anumit de timp.

Este aplicată relația ***One-To-Many*** în referință cu **currency.id**, ce este *primary key* și **currency_value.currency_id** ce este *foreign key*.

Aspecte Tehnice

Stack-ul Tehnologic

- 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].

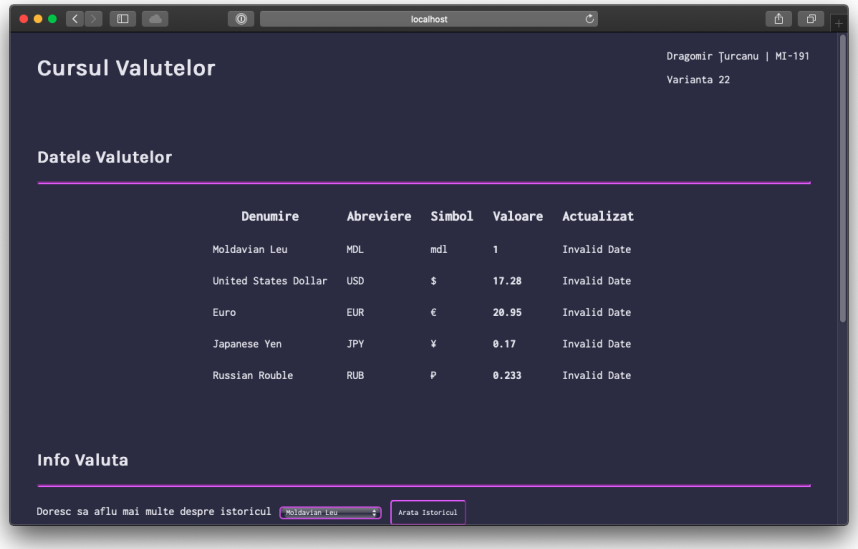
Ținând acest fapt în considerare, am creat o singur loc de intrare, fișierul **index.html**, ce și conține toată partea vizuală, informația fiind încărcată prin requesturile **fetch** realizate de către client, pentru a trage informația din **endpointurile** ale **API-ului** [5] intern.

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

[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

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.

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
// 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,
        sfarsește funcția și întoarce un array gol ca http
        răspuns
            $response->getBody()->write(json_encode([]));
            return $response;
```

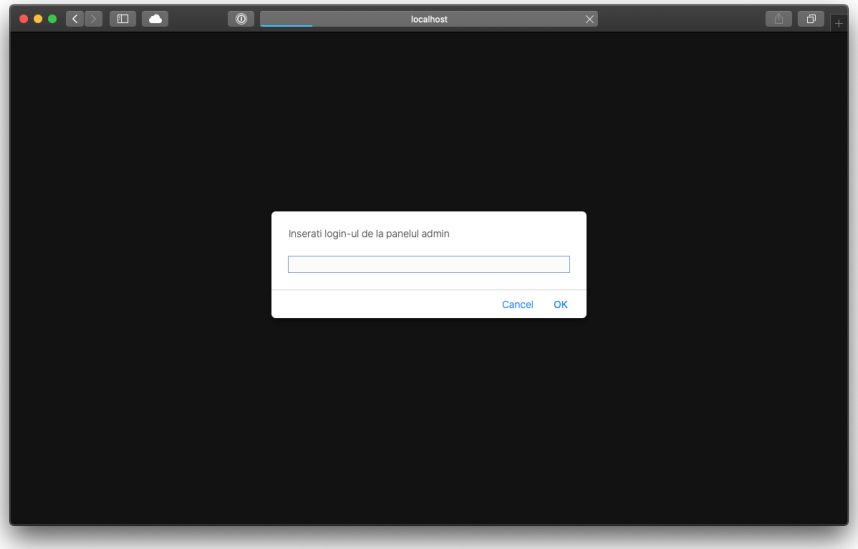
```
        }

        $responseFormatted = array();
        foreach ($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.

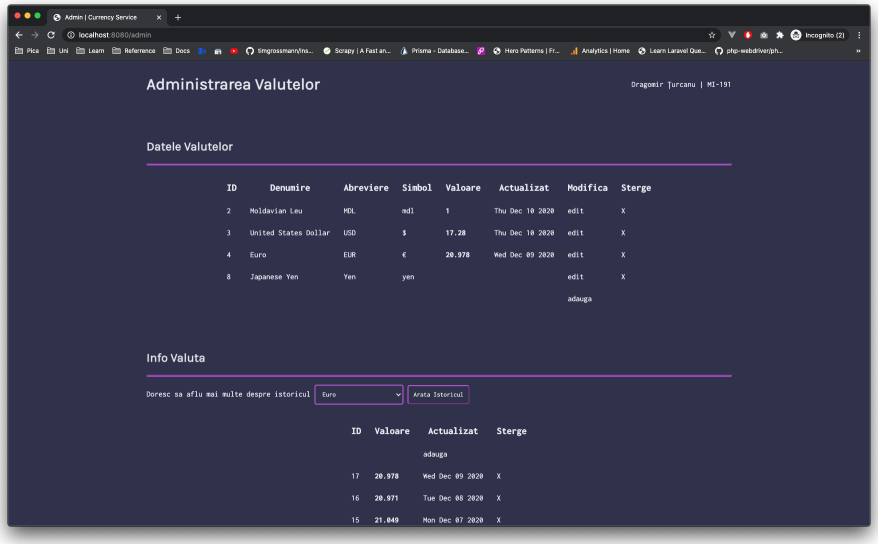
Adminstrare

Pentru a permite independența managementului aplicației de la accesul direct către **SGBD**, au fost create un număr de endpointuri ale aplicației care pot fi accesate doar folosind accesul prin combinația ***login + pass***, indicată în headerul requestul spre API. Toate requesturile spre operații Create / Update / Delete sunt filtrare prin intermediul ***middleware***-ului "AdminMiddleware", ce acceptă sau revocă requestul din considerentul existenței și coincidenței headerului de authenticare cu loginul și parola de control. Astfel această aplicație exemplu primește un mic boost de securitate.

Pentru a accesa pagina de administrare, este nevoie de a naviga spre `/admin`. Odată încărcată, pagina va întâlni utilizatorul cu un **popup** ce va interoga utilizatorul în privința credențialelor de acces.



După inserarea datelor în interogări, administratorul este întâlnit cu următoarea interfață. Aceasta este în practică pagina de start a aplicației, cu o serie de moficiări pentru adăgurea și modificarea datelor cu ușurință.



Astfel apar butoanele de tip **"adaugă"**, **"edit"** și **X (șterge)**. Administrarea datelor în așa caz devine foarte rapidă și luând în considerare că aici se

păstrează arhitectura bazată pe **AJAX**, toată interfața este foarte interactivă și nu necesită refresh pentru a privi cele mai noi date.

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!*