

Documentație EvoCMS/InfoTest

Sistem de Management al Conținutului
(platforma de testare probleme de informatică)

Cuprins

Documentație EvoCMS/InfoTest.....	1
Tehnologii utilizate în dezvoltarea părții de evaluator	1
Tehnologii PHP utilizate	2
1) Paradigma programării orientate pe obiecte (POO).....	2
2) Despre paradigma de programare Model-View-Controller.....	3
3) Modul de funcționare a cadrului de lucru (framework) Laravel 6.....	3
Documentație proiect EvoCMS/InfoTest.	5
1) Structura dosarului (namespace) App.	5
2) Prezentarea claselor definite în cadrul proiectului.....	6
3) Plug-ins și Teme: Adăugarea de funcții noi platformei.	7
Prezentare platformă InfoTest.....	9
Bibliografie	13

Tehnologii utilizate în dezvoltarea părții de evaluator

Mecanismul de evaluare este un script Python care rulează tot timpul în fundal în containerul său propriu care scanează în permanență după existența unui fișier de intrare „entry.txt”.

```
import os
import subprocess
import re
import random as rand
from signal import signal, SIGINT, SIGTERM
from sys import exit
```

Scriptul este scris în totalitate folosind biblioteci de funcții predefinite în Python precum: os, subprocess, re (regular expressions), random (generare aleatoare de numere), signal (pentru semnale linux precum SIGINT și SIGTERM) și funcția sys.exit() (pentru ieșirea din proces la oprirea containerului)

Bucula principală scanează directorul comun /opt/solutions după existența unui fișier „entry.txt” care conține toate instrucțiunile necesare pentru a cunoaște cine a încărcat o soluție și asupra cărei probleme trebuie efectuată evaluarea.

```
1 USER 7568dfbb-60f5-47b4-975f-5985f6c9731b
2 PROBLEM suma
3 DEFINE <solution> AS <user>_<problem>.c
4 EVAL <solution> <problem>/stages.txt
```

USID generat la înregistrarea utilizatorului (arata cine este logat)
PROBLEM arată ce problemă se evaluează. DEFINE definește o variabilă solution de tipul user_problem.c (evaluarea se face pe cod C asupra stagiilor de evaluare descrise in stages.txt)

```
def main():
    signal(SIGINT, signalHandler)
    signal(SIGTERM, signalHandler)

    ssid = pickRandomString()
    while True:
        if os.path.isfile("entry.txt"):
            os.rename("entry.txt", "{0}_entry.txt".format(ssid))
            print("[Evaluator] Interpreting entry file with session id {0}".format(ssid))
            interpretEntry(ssid, {"defs": {}, "stages": ""})
            ssid = pickRandomString()

if __name__ == "__main__":
    main()
```

Cat timp procesul este pornit se caută fișierul „entry.txt” iar atunci când a fost găsit este redenumit cu un cod de 10 caractere generat aleatoriu dintr-un amestec de caractere alfanumerice, apoi se pornește procesul de interpretare și evaluare. Evaluarea fișierului sursă constă în mutarea acestuia în dosarul solutions aferent problemei pentru care se face evaluarea, compilarea codului sursă și rularea executabilului asupra dosarului asociat stagiului de evaluare. Un stagiul de evaluare reprezintă o probă de date de intrare peste care are loc evaluarea efectivă a problemei.

```
with open("{0}_eval.log".format(variables["user"].strip()), "w") as log:
    scoreTotal = 0
    process = subprocess.Popen(["gcc", "-o", compiled, solutionFile], stdout=log, stderr=log, universal_newlines=True)
    return_code = 0
    log.write("Code compiled! \n")
    while True:
        return_code = process.poll()
        if return_code is not None:
            break
    if return_code == 0:
        with open(stages, "r") as stagesFile:
            line = stagesFile.readline()
            while line:
                stage, score = [ k.strip() for k in line.split(" ") ]
                folder = "{0}/{1}".format(variables["problem"].strip(), stage)
                os.chdir(folder)
                run = subprocess.Popen(["../{0}".format(compiled)], stdout=log, stderr=log, universal_newlines=True)
                log.write("Stage {0}: File evaluated. Score: ".format(stage))
                while True:
                    return_code = run.poll()
                    if return_code is not None:
                        break
                with open("{0}.ok".format(variables["problem"].strip()), "r") as ok:
                    lines_ok = ok.readlines()
                    ok.close()
                    lines_out = []
                    with open("{0}.out".format(variables["problem"].strip()), "r") as out:
                        lines_out = out.readlines()
                        out.close()
                    if lines_out == []:
                        log.write("No output read from file\n")
                    else:
                        if len(lines_out) == len(lines_ok):
                            ok = True
                            for i, line in enumerate(lines_ok):
                                if line != lines_out[i]:
                                    ok = False
                        else: ok = False
                        if ok:
                            log.write("{0}\n".format(score))
                            scoreTotal = scoreTotal + int(score)
                    os.remove("{0}.out".format(variables["problem"].strip()))
                os.chdir("../..")
                line = stagesFile.readline()
        log.write("Total score obtinut: {0}\n".format(scoreTotal))
        os.remove(compiled)
        log.close()
    return return_code
```

Tehnologii PHP utilizate

1) Paradigma programării orientate pe obiecte (POO)

Limbajul PHP la bază este un limbaj de programare procedural dar introduce și alte paradigme de programare utile printre care și cel Orientat pe Obiecte (POO). Această paradigmă folosește ca tip de date de bază clasa. O clasă reprezintă o colecție de atribute și metode ce definesc instanțele acesteia. Aceste instanțe se numesc obiecte. Obiectele sunt niște variabile definite de tipul de clasă din care fac parte acestea.

În PHP o clasă se definește în felul următor unde:

- <vizibilitate> poate fi una din cuvintele: public (vizibil atât în interior cât și în exterior), private (vizibile doar în interiorul clasei), protected (folosit în clase derivate din clasa părinte)
- [static] este opțional și prezintă dacă tipul atributului sau metodei este statică sau nu. (Adică, poate fi accesat independent de obiect)
- <tip> reprezintă un tip de date (ce poate fi dedus automat de interpretor)
- <atribuire> este o expresie ce poate fi asociată unui atribut la inițializarea unui obiect din clasă

```
class <nume-clasă> {
    <vizibilitate> [static] $<nume-atribut> [= <atribuire>] ...
    <vizibilitate> [static] function <nume-funcție>({[<tip>] $<nume-param> [,]}) {
        <instrucțiuni>
    } ...
}
```

În continuare proiectul a fost realizat folosind 100% programarea orientată pe obiecte.

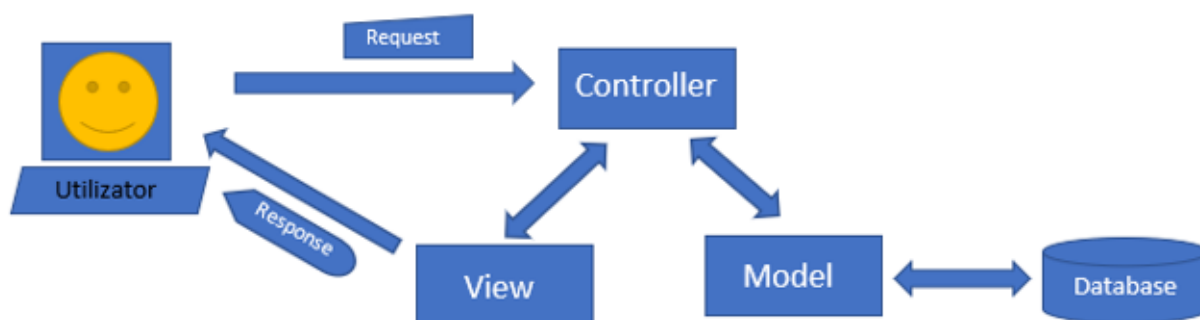
2) Despre paradigma de programare Model-View-Controller

Paradigma Model-View-Controller este un caz particular de programare orientată pe obiecte folosită adesea în realizarea de aplicații cu interfața grafică și baze de date pentru gestiunea informației prelucrate de program.

Modelul reprezintă entitatea ce creează legătura cu bazele de date. Acesta preia informația, o convertește în format ușor de înțeles de utilizator și o predă Controller-ului care încearcă să se folosească de aceasta pentru a putea fi transmisă ușor View-ului care o afișează utilizatorului final. Tot modelul se ocupă și cu actualizarea tabelii dar și adăugarea de informații noi în baza de date.

Controllerul preia cererile (request-urile) de la utilizator, le interpretează și pe baza acestora face cerere la Model sau întoarce un răspuns (response) adesea reprezentând pagina la care s-a făcut cerere. (care reprezintă un View).

View-ul este de fapt pagina care este afișată de browser utilizatorului. Prin aceasta el poate efectua alte cereri (requests) către Controller.



3) Modul de funcționare a cadrului de lucru (framework) Laravel 6.

a) *Ce este un framework și la ce folosește Laravel 6. (Definiția unui Framework - Wikipedia, 2018)*

Un framework este o structură conceptuală și reprezintă o arhitectură de software care modelează relațiile generale între componentele de bază ale unui program. Acesta reprezintă un ansamblu standardizat de concepte ce stă la baza rezolvării unei categorii de probleme. Laravel 6 este un framework web ce implementează elementele de bază a funcționării corecte a unui website PHP conform paradigmei conceptuale de Model-View-Controller (MVC).

b) *Structura unui proiect Laravel 6 (Structura Directoarelor - Laravel Documentation, n.d.)*

Un proiect este structurat pe mai multe directoare. Acestea au un rol în felul acestora pentru a compune în final pagina web văzută de către utilizatorii finali.

- The Root Directory

- The **app** Directory

- Acest director conține codul de bază a aplicației dezvoltate. Majoritatea claselor din aplicația web se afla în acest director.

- The **bootstrap** Directory

- Acest director conține fișierul app.php care este primul executat de către framework și inițializează aplicația web.
 - Tot aici se găsește un folder cache unde sunt salvate fișiere ce permit rularea rapidă a aplicației, fișiere folosite de către framework.

- The **config** Directory

- Aici se găsesc fișierele de configurație a aplicației web. Dacă este necesară înregistrarea unor elemente ale aplicației web aici se adaugă.
 - (De exemplu un Service Provider ce permite rularea unui serviciu specific aplicației)

- The **database** Directory

- Aici se găsesc fișierele ce compun Baza de date (scripturi, migrări, etc). Dacă este una SQLite tot aici poate fi salvată.

- The **public** Directory

- Acest folder conține fișierele publice ale aplicației. Fișierul index.php reprezintă rădăcina aplicației. Tot aici se găsesc imagini, fișiere JavaScript și CSS finale.

- The **resources** Directory

- Acest folder are toate fișierele raw ale website-ului (fișiere CSS SASS neacompile, cod Node.js JavaScript ajutător) dar și fișierele View. (scrise în format template engine blade.php)

- The **routes** Directory

- Aici sunt fișiere ce definesc căile de acces la paginile aplicației web. Tot aici pot fi definite rute specifice unui REST API (Representational State Transfer Application Programming Interface)

- The **storage** Directory

- La cererea unui client la o pagină web aici se găsesc fișierele PHP ce compun un View compile.

- The **tests** Directory

- Cod pentru testarea unitară a aplicației web. (directorul este opțional și poate lipsi din proiect)

- The **vendor** Directory

- Bibliotecile de bază ale Laravel-ului. Tot ce este descărcat și instalat cu ajutorul a managerului de pachete Composer.

- The App Directory

- The **Http** Directory

- În acest director se află toate Controllele ce compun funcționalitățile aplicației web. Tot aici se mai găsesc și clasele ce intervin în controlul prealabil al unor rute ale aplicației. (de exemplu controlul accesului restricționat la anumite pagini web prin autentificarea unui utilizator)

- The **Providers** Directory

- Aici se găsesc toate serviciile oferite de către aplicație.

- Spre exemplu: manipularea unor date transmise de utilizator către baza de date dar și controlul accesului la teme (Themes) și cod adițional (Plugins)

c) Modul de implementare a Modelelor. Comunicarea cu baza de date

Modul de implementare a modelelor în Laravel se face abstract, ascuns în implementarea clasei Model ce aparține bibliotecii Eloquent. Această clasă mamă dispune de metode predefinite ce permit execuția de query statements în SQL.

La momentul efectuării cererii la un câmp din tabelă, PHP are o metoda magică (magic method) denumită `__get()` care în Laravel se apelează metoda `getAttribute()`. Această metodă caută atributul într-un array de relații (tabele) obținut prin folosirea funcțiilor `mysqli_fetch_*` existente în PHP. Același procedeu se efectuează și la realizarea unei atribuirii numai ca se folosește de o implementare a metodei `__set()` din PHP.

Pentru definirea unor legături între modele, Laravel definește o colecție de clase specifice fiecărei relații în parte. Pe scurt, există metode ce se pot apela dinamic de către metodele definite de către programator. Aceste metode sunt: `hasOne()`, `belongsTo()` (one-one relationship); `hasOne()`, `hasMany()`, `belongsToMany()` sau `hasMany()`, `belongsTo()` (many-one relationship); `hasMany()`, `hasMany()` (many-many relationship). ([Relații - Laravel Documentation](#), n.d.)

d) Dispatcherul, rutele și comunicarea cu clasele Controller.

Orice proiect Model-View-Controller se folosește de un element central denumit Dispatcher. Acesta preia toate cererile efectuate de către server, le despachetează (acele pachete HTTP) și prelucrează adresa web primită pentru a recunoaște ruta la care s-a realizat referirea. Pe baza acesteia se uită într-o clasă mamă denumită Route (clasă abstractă, nu pot exista obiecte din această clasă) și caută ruta la care a vrut să se refere utilizatorul. Pe baza acesteia se descoperă Controllerul care se ocupă de acea pagină web. Este apoi creat un obiect de tip controller și este apelată metoda asociată rutei pe baza tipului de cerere efectuată. (GET, POST, DELETE, UPDATE/PUT)

Documentație proiect EvoCMS/InfoTest.

Proiectul EvoCMS a fost realizat folosind tehnologia oferită de către Laravel 6. Absolut tot codul PHP al proiectului a fost scris în dosarul „app” care definește spațiul de nume „App”. Aici am șase dosare (namespaces) fiecare conținând cod pentru fiecare componentă specifică MVC dar și independentă de proiect (Plugins și Themes).

1) Structura dosarului (namespace) App.

a) Dosarul Http:

a. Controllers:

- Acest spațiu de nume conține toate clasele de tip Controller de bază ale proiectului.
- Clasa `LoginController` (ce realizează conectarea unui utilizator la aplicație și afișează formularele de log-in și sign-up)
- Clasa `UserController` (ce permite înregistrarea unui utilizator la aplicație)

b. Middleware:

- Aici se găsește clasele middleware (clase intermediare ce efectuează verificări prealabile de restricționare al accesului la o anumită rută)
- Se găsește o singură clasă `UserTokenMiddleware` care verifică existența token-ului utilizatorului și determinarea acestuia.

b) Dosarul Models:

- Conține toate modelele ce comunică cu baza de date în vederea preluării de informații.

c) Dosarul Observers:

- Care conține clase intermediare ce verifică datele primite de la utilizator și le prelucrează în vederea menținerii integrității datelor.

d) Dosarul Plugins:

- Aici se găsesc dosare pentru fiecare plugin creat de alți programatori în vederea extensibilității platformei

e) Dosarul Providers:

- Dosarul acesta conține serviciile de bază ale Laravel-ului dar și altele noi create de programator.

- i. PluginServiceProvider este clasa care încearcă se activeze și să încarce fiecare plugin nou instalat de client
 - ii. ThemesServiceProvider face o cerere la baza de date pentru a prelua tema curentă a utilizatorului și încearcă să o găsească pentru a o încărca și ai permite schimbarea aspectului paginilor web văzute de utilizatorul final (persoana străină care vizitează site-ul).
- f) Dosarul Themes:
- a. Aici se găsesc toate temele noi instalate ce urmează spre a fi încărcate de către aplicație.

2) Prezentarea claselor definite în cadrul proiectului

- a) **Clasa LoginController:** conține tot codul ce are grijă de procesul de Login al unui utilizator la aplicație.

```
public function submit(Request $request) { // Metoda submit primește cererea utilizatorului
    // Se creează o instanță a clasei Validator ce verifică câmpurile oferite
    $validator = Validator::make($request->all(), [
        'Email' => 'required', // Dacă câmpul email este introdus
        'Password' => 'required', // Și dacă câmpul parolă a fost introdus
    ]);
    // Dacă una dintre câmpuri nu există atunci se cer introducerea lor
    if ($validator->fails()) {
        return redirect('/admin/login')
            ->withErrors($validator)
            ->withInput();
    }
    // Se caută utilizatorul în baza de date după câmpul Email
    $user = User::find($request->all('Email')['Email']);
    $hash = $user->Password; // Și se preia câmpul Password hashed
    // Se verifică dacă avem user și parola dacă cea tastată corespunde cu hash-ul din
    // baza de date
    if(isset($user) && password_verify($request->all('Password')['Password'], $hash)){
        // Dacă utilizatorul vrea să fie ținut minte se creează un Cookie valabil 7 zile
        // altfel valabil 5 minute
        // cu un cod unic de identificare preluat din baza de date (Token)
        $logged_in = $request->all('logged-in')['logged-in'];
        if($logged_in) setcookie('token', $user->auth_token, strtotime('+7 Days'), '/');
        else setcookie('token', $user->auth_token, strtotime('+5 Minutes'), '/');
        // Se face trimitere la pagina admin
        return redirect('admin');
    } else {
        // Se anunță utilizatorul că Email nu a fost găsit sau parola nu este corectă
        if(!isset($user))
            $validator->errors()->add('Email', 'The specified email was not found! Are you
registered?');
        else if(!password_verify($request->all('Password')['Password'], $hash))
            $validator->errors()->add('Password', 'The specified password was invalid!');
        // Se face trimitere înapoi la formularul de login și se afișează erorile.
        return redirect('/admin/login')->withErrors($validator)->withInput();
    }
}
```

Această clasă definește două metode care afișează utilizatorului formularele de login și signup și o metodă ce efectuează delogarea clientului. În continuare se va prezenta metoda submit care realizează verificarea utilizatorului și conectarea efectivă a acestuia la aplicație.

- b) **Clasa UserController:**

Aici se definește o singură metodă, store care efectuează înregistrarea utilizatorului în baza de date. Metoda aceasta corespunde formularului de Signup.

```
public function store (Request $request) {
    // Validare câmpuri
    $validator = Validator::make($request->all(), [
        'Username' => 'required|unique:users',
```

```

        'Email' => 'required|unique:users',
        'Password' => 'required',
    ]);
    // Trimitere la register cu erori.
    if ($validator->fails()) {
        return redirect('register')
            ->withErrors($validator)
            ->withInput();
    }
    User::create($request->all()); // Această singură instrucțiune face tot farmecul.
    return redirect('/admin/login');
}

```

c) Clasa UserTokenMiddleware:

Această clasă stă la mijlocul tuturor rutelor din aplicație și verifică existența unui utilizator conectat la aplicație.

```

class UserTokenMiddleware
{
    /**
     * Handle an incoming request.
     * Metoda verifică existența unui cookie token și dacă tokenul există în baza de date
     *
     * @param Request $request
     * @param Closure $next
     * @return mixed
     */
    public function handle($request, Closure $next) {
        if(!isset($_COOKIE['token'])) return redirect('admin/login');
        $user = \App\Models\User::all()->where('auth_token', '=', $_COOKIE['token']);
        if(!isset($user)) return redirect('admin/register');

        return $next($request);
    }
}

```

d) Rutele de bază ale aplicației:

```

// Prefixul root sub ruta admin
Route::prefix('/')->group(function () {
    Route::prefix('admin')->group(function () {
        // Ruta /admin/ ce apelează login.submit cu metoda POST (accesată de formular)
        Route::post('/', 'Admin\LoginController@submit')->name('login.submit');
        Route::get('/login', 'Admin\LoginController@login')->name('login.login');
        Route::get('/register', 'Admin\LoginController@register')->name('login.register');
    });
    // Ruta user.create este cea folosită aici
    Route::apiResource('/user', 'Resources\UserController');
    // Rutele /admin/ pe get (afișează pagina admin)
    // și ruta /admin/logout șterge cookie-ul și redirecționează la /admin/login
    Route::middleware('admin-auth')->prefix('admin')->group(function () {
        Route::get('/', function () {return view('admin');})->name('admin.index');
        Route::get('/logout', 'Admin\LoginController@logout')->name('login.logout');
    });
});

```

3) Plugins și Teme: Adăugarea de funcții noi platformei.

Platforma EvoCMS de management al conținutului, proiect realizat la DAW (dezvoltarea aplicațiilor web) pe sem I a fost gândită să permită încărcarea de noi funcții platformei cu ajutorul unor Plugin-uri. Un plugin permite instalarea de noi rute, controllere, modele și chiar secțiuni în panoul de control fără a intervenii în mod direct în codul sursă al platformei.

Platforma atunci când este accesată are multiple clase de tip servicii (ServiceProvider) care permit rularea de funcționalități noi la rulare. O astfel de clasă a fost implementată pentru încărcarea de pluginuri noi definite în fișierul PHP „app/Plugins/plugins.php”. Acest fișier PHP întoarce la incluziune un array (tablou unidimensional) de căi

către clase derivate de tip abstract Plugin. Aceste clase au comune funcția load() care încarcă pluginul și este apoi înregistrat în aplicație pentru cazul în care pluginul respectiv permite încărcarea de noi clase de tip serviciu. (ServiceProvider)

```
namespace App\Providers;

use Illuminate\Support\ServiceProvider;

class PluginServiceProvider extends ServiceProvider {
    public $loadedPlugins = [];

    /**
     * Bootstrap services.
     *
     * @return void
     */
    public function boot() {
        $plugins = include base_path('app/Plugins/plugins.php');
        foreach ($plugins as $key => $plugin) {
            $this->loadedPlugins[$key] = $plugin::load()->registerPlugin($this->app);
        }
    }
}
```

```
return [
    'Probleme' => \App\Plugins\Probleme\ProblemePlugin::class,
    'AdminUser' => \App\Plugins\AdminUser\AdminUserPlugin::class,
    'UserRoles' => \App\Plugins\UserRoles\UserRolesPlugin::class
];
```

```
use Illuminate\Contracts\Foundation\Application;
use Illuminate\Foundation\Auth\Access\AuthorizesRequests;
use Illuminate\Foundation\Bus\DispatchesJobs;
use Illuminate\Foundation\Validation\ValidatesRequests;
use Illuminate\Routing\Controller as BaseController;
use Illuminate\Support\ServiceProvider;

/**
 * Class Plugin
 * @package App\Plugins
 * @var Plugin $plugin
 * @var NamespacedClass[] $classes
 * @var NamespacedClass[] $dependencies
 * @var Plugin[] $loadedDependencies
 * @var ServiceProvider[] $serviceProviders
 */
class Plugin extends BaseController {
    use DispatchesJobs, ValidatesRequests, AuthorizesRequests;

    protected static $plugin;
    public $serviceProviders = [];
    public $classes = [];
    public $dependencies = [];
    public $loadedDependencies = [];
    protected function __construct() {}
    public static function load() {
        if(is_null(self::$plugin)) {
            self::$plugin = new Plugin();
        }
        return self::$plugin;
    }
    public function registerPlugin(Application &$app) { }
```

Clasa „Plugin” este o clasa abstracta care implementează un controller. Această clasă sta la baza tuturor claselor plugin încărcate pe platformă. Pluginuri precum: UserRoles, care evidențiază cele trei roluri asociate platformei InfoTest (Student, Teacher, Administrator); Probleme, care permite adăugarea de noi probleme pe platformă și AdminUser care adaugă acces direct la un utilizator Administrator cu parolă Admin. Se pot adăuga modificări să se poată schimba parola dacă se dorește acest lucru.

Pe această platformă se pot adăuga și noi teme, ideea de temă este aceea de a permite o ușoară utilizare a platformei de către studenți care nu au acces la panoul de administrator, aceștia putând vedea doar ce probleme noi sunt încărcate pe platformă, ce punctaje au obținut alți colegi și încărca soluții (rezolvări) la o problemă încercată. O temă funcționează pe același principiu ca și un plugin fiind și ea la rândul ei un plugin dar cu mai multe privilegii. O problemă este aceea că nu se pot folosi mai mult de o temă simultan, fiind încărcată o dată doar o temă. Dacă se dorește implementarea de noi funcții în temă se recomandă implementarea de noi pluginuri instalate separat.


```
/**
 * Class Theme
 * @package App\Themes
 * @var Theme $themeController
 * @property string $themeName
 * @property ServiceProvider[] $serviceProviders
 */
class Theme extends BaseController {
    use DispatchesJobs, ValidatesRequests, AuthorizesRequests;

    protected static $themeController = null;
    public static $themeName = '';
    public static $serviceProviders = [];

    public function registerServiceProviders(Application $app) { return self::$themeController; }
    public function boot() {}
    public static function load(array $themes) {
        $theme = Setare::all()->where('Obtiune', '=', 'current_theme');
        if(!$theme->isEmpty() && is_null(self::$themeController)) {
            self::$themeName = $theme->first()->Valoare;
            self::$themeController = new Themes[self::$themeName];
            return self::$themeController;
        }
        return new Theme();
    }
}
```

Tema este încărcată la accesarea platformei folosind baza de date accesând opțiunea current_theme care are ca valoare cheia asociată clasei derivate Theme din array \$themes transmis funcției load.

```
class ThemesServiceProvider extends ServiceProvider {

    public $themes = [
        'Snipp'=> \App\Themes\Snipp\Snipp::class
    ];

    /**
     * Bootstrap services.
     *
     * @return void
     */
    public function boot() {
        $loadedTheme = Theme::load($this->themes->registerServiceProviders($this->app);
        if(!is_null($loadedTheme) && class_basename($loadedTheme) != Theme::class)
            $loadedTheme->boot();
    }
}
```

Prezentare platformă InfoTest

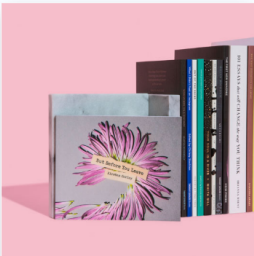
Platforma InfoTest este o platformă simplă ce permite rezolvarea de probleme încărcate de profesori și administratori. Soluțiile la probleme sunt încărcate atât de către studenți cât și de profesori.

Pe prima pagina a platformei se pot vedea o colecție de probleme puse la dispoziție spre a fi rezolvate. Fiecare problemă este unic reprezentată printr-o scurtă cerință și o imagine.

INFOTEST - PLATFORMA DE EVALUARE A SOLUTIILOR UNOR PROBLEME

Home User Page

Probleme

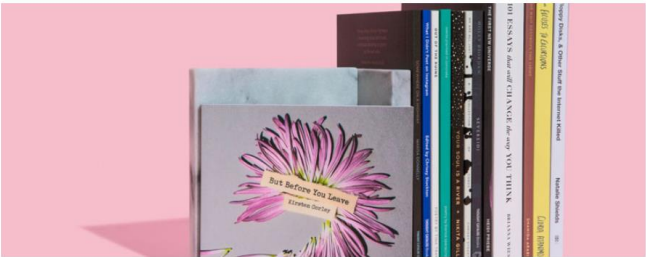


2020-07-30 Admin

Sa se calculeze suma unor numere date.

Copyright ©2020 All rights reserved | This template is made with by Colorlib

La apăsarea uneia dintre imagini studentul este trimis către prezentarea cerinței problemei și un scurt exemplu. De acolo se poate alege „încarcă o soluție” sau vezi punctaj. O soluție este încărcată în dosarul soluții al evaluatorului numai dacă se poate identifica studentul (dacă acesta este conectat pe platformă).



#1. Problema: Suma

Sa se calculeze suma unor numere date.

suma.in	suma.out
4 2 3 1 7	13

Vezi punctaj

Încarca soluția

În următoarea imagine se poate vedea cum că această problemă a fost rezolvată atât de administrator cât și de un student de exact două ori.



#1. Problema: Suma

Utilizator	Scor
Administrator	100
Petrickah	100
Petrickah	100

Sa se calculeze suma unor numere date.

suma.in	suma.out
4 2 3 1 7	13

Vezi punctaj

Încarca soluția

Iar în această imagine se poate vedea că este suficienta încărcarea unui fișier C ce urmează a fi compilat și apoi evaluat pentru acordarea scorului.

INFOTEST - PLATFORMA DE EVALUARE A SOLUTIILOR UNOR PROBLEME
Home
User Page

#1. Problema: Suma

Incarca solutie: No file chosen

Sa se calculeze suma unor numere date.

suma in	suma out
4 2 3 1 7	13

Pentru conectare sau creare cont de student se accesează meniul dropdown „User Page” și se alege Login sau Register. Pe pagina „/admin/login” se completează numele de utilizator și parola. Iar pe pagina „/admin/register” se realizează contul.

Purple

Hello! let's get started
Sign in to continue.

SIGN IN

☐ Keep me signed in

Don't have an account? [Create](#)

Purple

New here?
Signing up is easy. It only takes a few steps

☐ I agree to all Terms & Conditions

SIGN UP

Already have an account? [Login](#)

Pe pagina Administrator se pot vedea numarul total de comentarii (în caz de implementare funcție de comentarii), postari(probleme) și cuvinte cheie (în caz de implementare funcție de căutare) Lista cu toate problemele recent încărcate și posibilitatea de schițare rapidă de postare/problemă (fără categorie, imagine de thumbnail sau fișiere de evaluare).

Dashboard
Probleme
Solutii
Utilizatori

Dashboard

Comentarii probleme
0 Comentarii

Probleme postate
1 Postari

Keywords
0 Keywords

Probleme Recente

Author	Title	Category	Create
Administrator	Suma	Incarcare	2020-07-08 13:11:07

August 2020

Su	Ma	Ti	Mi	Jo	Vi	Sa
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	1	2	3	4

Problema Rapida

Aici se poate vedea formularul de adăugare problemă nouă. O problemă este dată de un titlu, categorie, cuvinte cheie (folosite pentru a putea căuta o problemă), fișiere evaluator (o arhivă ce conține dosare de stagiu de evaluare cu fișiere .in și .ok respectiv un fișier text stages.txt care descrie câte stagii sunt și câte puncte reprezintă fiecare) și Imagine thumbnail care poate fi orice format de poză .jpg sau .png) respectiv conținutul problemei (cerința și exemplu).

The screenshot shows the 'Adaugare Problema Noua' form. It has a sidebar with 'Probleme' and 'Solutii' sections. The main form has the following fields:

- Title:** A text input field.
- Category:** A dropdown menu.
- Keywords:** A text input field.
- Evaluator File:** A file upload button labeled 'Choose File'.
- Thumbnail:** A file upload button labeled 'Choose File'.
- Content:** A large text area for the problem description.

A 'Problema' button is located at the bottom of the form.

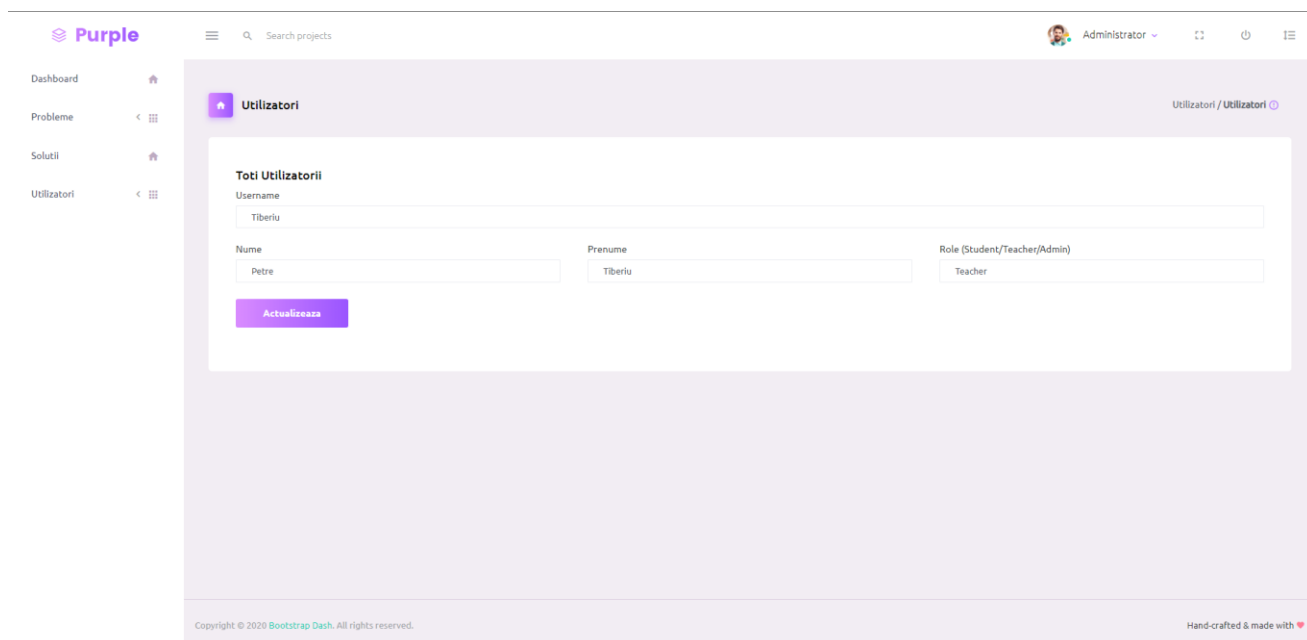
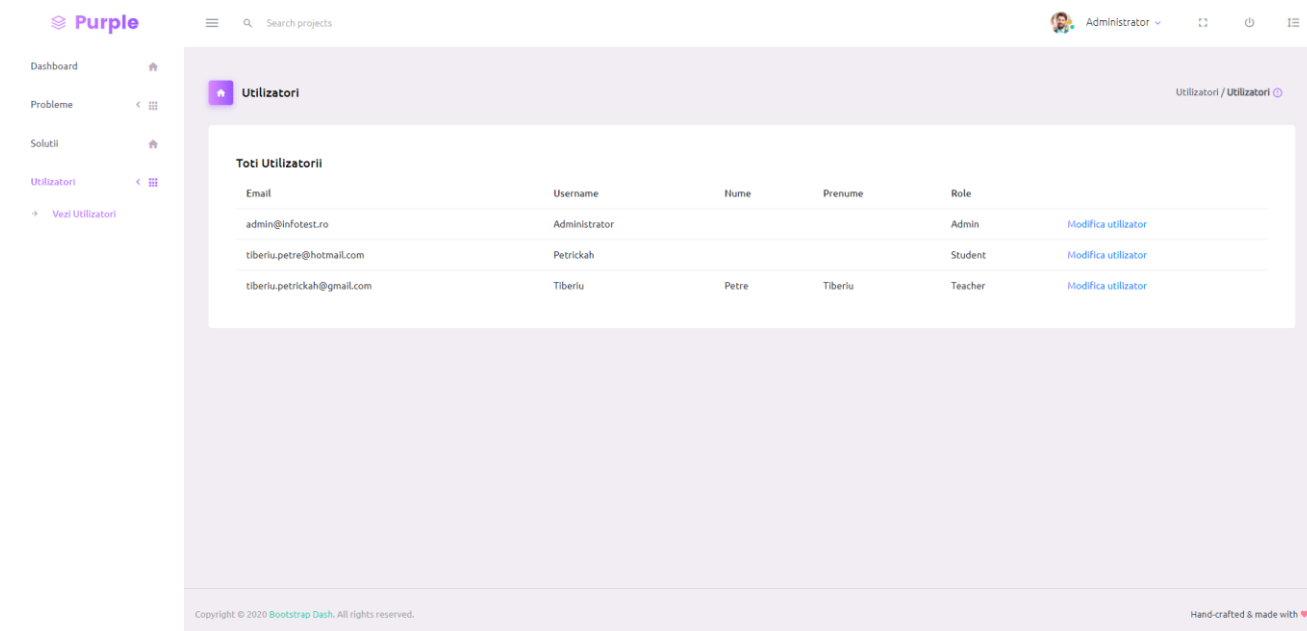
La meniul Soluții se pot vedea cine a încărcat o soluție, la ce problemă și ce punctaj au obținut. Tot de aici se poate vedea descrierea problemei, edita respectiv șterge problema. Probleme pot fi văzute și accesând Probleme > Vezi problema.

The screenshot shows the 'Toate Solutiile' page. It contains a table with the following data:

Problema	Student	Scor	
suma	Administrator	100	Vezi problema
suma	Petrickah	100	Vezi problema
suma	Petrickah	100	Vezi problema

The footer of the page includes the text: 'Copyright © 2020 Bootstrap Dash. All rights reserved.' and 'Hand-crafted & made with ❤️'.

Din meniu se poate accesa lista cu toți utilizatorii. Un utilizator are un email, nume de utilizator, nume, prenume și rol. Parola este specifică fiecărui utilizator și nu poate fi schimbată. Un utilizator o dată creat, nu poate fi șters. Se poate implementa această funcționalitate dacă se dorește.



Bibliografie

Definiția unui Framework - Wikipedia. (2018). Preluat de pe Wikipedia: <https://ro.wikipedia.org/wiki/Framework>

Relații - Laravel Documentation. (fără an). Preluat de pe Laravel Documentation: <https://laravel.com/docs/6.x/eloquent-relationships>

Structura Directoarelor - Laravel Documentation. (fără an). Preluat de pe Laravel Documentation: <https://laravel.com/docs/6.x/structure>