# Login

Στην κλάση “UserService” υπάρχει μία μέθοδος με το όνομα “login” περιμένει ως παράμετρο το LoginDto που το λαμβάνει από το HTML που έχει κάνει post.

HTML Code:

if(isset($\_POST['email']) && isset($\_POST['password'])){

      $loginDto = new LoginDto();

      $loginDto->email = $\_POST['email'];

      $loginDto->password = $\_POST['password'];

      $loginDto->isRememberMe = isset($\_POST['remember\_me']);

      $mRet = $userService->login($loginDto);

      if($mRet->status == 0){

        header('location: dashboard');

      }

}

User Service -> Login Method

public function login(LoginDto $loginDto) : ResponseObject {

        global $userRepository;

        $validator = new LoginDtoValidator();

        if (!$validator->validate($loginDto))

            return new ResponseObject(1, $validator->validation\_error);

        $user = $userRepository->fetchWithEMail($loginDto->email);

        if(!isset($user) || $user == NULL){

            return new ResponseObject(1, "User does not exist.");

        }

        if(!comparePasswords($loginDto->password, $user->password)){

            return new ResponseObject(1, "You typed wrong password.");

        }

        if($loginDto->isRememberMe){

            setcookie("LoginCredentials", $loginDto, time() +2592000, "/");

        }

        $user->password = "";

        $\_SESSION['user'] = serialize($user);

        return new ResponseObject(0, "Successfull Login.");

    }

Προτού κάνει κάποιο Select ελέγχει μέσο μιας Custom κλάσης αν είναι Valid το Object που μας έχει στείλει.

UserDtoValidator

class LoginDtoValidator extends BaseValidator{

    public function Validate($loginDto): bool

    {

        if(!isset($loginDto->email) || $loginDto->email == ''){

            settype($validation\_error, "string");

            $validation\_error = "E-Mail must not be null or empty.";

            return false;

        }

        if(!isset($loginDto->password) || $loginDto->password == ''){

            settype($validation\_error, "string");

            $validation\_error = "Password must not be null or empty.";

            return false;

        }

        return true;

    }

}

Έπειτα καλεί από το Repository την fetchWithEMail με όρισμα το e-mail του χρήστη:

public function fetchWithEMail(string $email): User

    {

        global $conn;

        $stmt = $conn->prepare("SELECT u.id, u.address\_id, u.contact\_id, u.role\_id, u.username

        , u.password, u.first\_name, u.last\_name, u.enable

        , r.description, r.can\_write, r.can\_read, r.can\_delete, r.can\_update, r.is\_manager, r.is\_admin

        , a.street, a.area, a.city, a.postal\_code, a.state, a.country

        , c.phone\_one, c.phone\_two, c.phone\_three, c.phone\_four, c.mobile, c.email

        FROM users AS u

        INNER JOIN roles AS r ON r.id = u.role\_id

        INNER JOIN addresses AS a ON a.id = u.address\_id

        INNER JOIN contacts AS c ON c.id = u.contact\_id

        WHERE UPPER(c.email) = UPPER(:email)");

        $stmt->bindParam(":email", $email, ***PDO***::PARAM\_STR);

        if(!$stmt->execute()){

            return NULL;

        }

        return $stmt->fetchAll(***PDO***::FETCH\_FUNC, array("VacationPortal\Data\Model\Security\User", "mapping"))[0];

    }

Αφού βρει χρήστη κάνει Map με το αποτέλεσμα με την κλάση και μας επιστρέφει το User Class.

Έπειτα η UserService->Login ελέγχει αν το Password που μας έδωσε ο χρήστης είναι ίδιο με αυτό της βάσης και αν είναι ίδιο τότε διαγράφω από το object το password (για να μην υπάρχει πουθενά και γίνει κάποια κλοπή από κάποιο session) προσθέτω τον χρήστη στο Session και επιστρέφω success code.

# Register

Η διαδικασία μοιάζει πολύ με την Login.

Από το HTML στέλνω το POST στο UserService στην μέθοδο registerUser. Εδώ όμως χρησιμοποιώ ένα άλλο DTO με όνομα UserDTO που διαφέρει ως προς το ότι είναι Flat και δεν έχει κάποια άλλα Object εσωτερικά αυτό με βοηθάει καλύτερα στο να είναι πιο απλό αυτό που διαβάζω στον κώδικα.

Οπότε στέλνοντας το UserDto στο UserService->Register κάνω και εδώ ένα Validate με τον ίδιο τρόπο που κάνω στο login και αρχίζω να αρχικοποιώ το αντικείμενο του User έπειτα το στέλνω στο Repository όπου αυτό αναλαμβάνει να κάνει το Insert στη βάση.

RegisterUser (UserService Method)

public function registerUser(UserDto $userDto): ResponseObject{

        global $userRepository;

        $validator = new UserDtoValidator();

        if (!$validator->validate($userDto))

            return new ResponseObject(1, $validator->validation\_error);

        $user = new User();

        $user->address = new Address();

        $user->contact = new Contact();

        $user->username = $userDto->username;

        $user->password = hashPassword($userDto->password);

        $user->first\_name = $userDto->first\_name;

        $user->last\_name = $userDto->last\_name;

        $user->date\_of\_birth = $userDto->date\_of\_birth;

        $user->role\_id = $userDto->role\_id;

        $user->address = new Address();

        $user->address->street = $userDto->street;

        $user->address->area = $userDto->area;

        $user->address->city = $userDto->city;

        $user->address->postal\_code = $userDto->postal\_code;

        $user->address->state = $userDto->state;

        $user->address->country = $userDto->country;

        $user->contact = new Contact();

        $user->contact->phone\_one = $userDto->phone\_one;

        $user->contact->phone\_two = $userDto->phone\_two;

        $user->contact->phone\_three = $userDto->phone\_three;

        $user->contact->phone\_four = $userDto->phone\_four;

        $user->contact->mobile = $userDto->mobile;

        $user->contact->email = $userDto->email;

        if (!$userRepository->add($user)) {

            return new ResponseObject(1, "error");

        }

        return new ResponseObject(0, "User Register Success");

    }

Validator

class UserDtoValidator extends BaseValidator{

    public function Validate($userDto) : bool{

        if(!isset($userDto->username) || $userDto->username == '') {

            // settype($validation\_error, "string");

            $validation\_error = "Username must not be null or empty.";

            return false;

        }

        if(!isset($userDto->password) || $userDto->password == ''){

            // settype($validation\_error, "string");

            $validation\_error = "Password must not be null or empty.";

            return false;

        }

        if(!isset($userDto->first\_name) || $userDto->first\_name == ''){

            // settype($validation\_error, "string");

            $validation\_error = "First Name must not be null or empty.";

            return false;

        }

        if(!isset($userDto->last\_name) || $userDto->last\_name == ''){

            // settype($validation\_error, "string");

            $validation\_error = "Last Name must not be null or empty.";

            return false;

        }

        return true;

    }

}

Add (Repository Method)

public function add(User $user) : bool {

        global $conn;

        $conn->beginTransaction();

        $stmt\_user =  $conn->prepare("INSERT INTO users(username, `password`, first\_name, last\_name, role\_id)

        VALUES(:username, :password, :first\_name, :last\_name, :role\_id)");

        $stmt\_user->bindParam(":username", $user->username, ***PDO***::PARAM\_STR);

        $stmt\_user->bindParam(":password", $user->password, ***PDO***::PARAM\_STR);

        $stmt\_user->bindParam(":first\_name", $user->first\_name, ***PDO***::PARAM\_STR);

        $stmt\_user->bindParam(":last\_name", $user->last\_name, ***PDO***::PARAM\_STR);

        $stmt\_user->bindParam(":role\_id", $user->role\_id, ***PDO***::PARAM\_INT);

        if(!$stmt\_user->execute())

        {

            $conn->rollBack();

            return false;

        }

        $user\_id = $conn->lastInsertId();

        $stmt\_address = $conn->prepare("INSERT INTO addresses(street, area, city, postal\_code, state, country, user\_id)

        VALUES (:street, :area, :city, :postal\_code, :state, :country, :user\_id)");

        $stmt\_address->bindParam(":street", $user->address->street, ***PDO***::PARAM\_STR);

        $stmt\_address->bindParam(":area", $user->address->area, ***PDO***::PARAM\_STR);

        $stmt\_address->bindParam(":city", $user->address->city, ***PDO***::PARAM\_STR);

        $stmt\_address->bindParam(":postal\_code", $user->address->postal\_code, ***PDO***::PARAM\_STR);

        $stmt\_address->bindParam(":state", $user->address->state, ***PDO***::PARAM\_STR);

        $stmt\_address->bindParam(":country", $user->address->country, ***PDO***::PARAM\_STR);

        $stmt\_address->bindParam(":user\_id", $user\_id, ***PDO***::PARAM\_STR);

        if(!$stmt\_address->execute()){

            $conn->rollBack();

            return false;

        }

        $address\_id = $conn->lastInsertId();

        $stmt\_contact = $conn->prepare("INSERT INTO contacts(phone\_one, phone\_two, phone\_three, phone\_four, mobile, email, user\_id)

        VALUES (:phone\_one, :phone\_two, :phone\_three, :phone\_four, :mobile, :email, :user\_id)");

        $stmt\_contact->bindParam(":phone\_one", $user->contact->phone\_one, ***PDO***::PARAM\_STR);

        $stmt\_contact->bindParam(":phone\_two", $user->contact->phone\_two, ***PDO***::PARAM\_STR);

        $stmt\_contact->bindParam(":phone\_three", $user->contact->phone\_three, ***PDO***::PARAM\_STR);

        $stmt\_contact->bindParam(":phone\_four", $user->contact->phone\_four, ***PDO***::PARAM\_STR);

        $stmt\_contact->bindParam(":mobile", $user->contact->mobile, ***PDO***::PARAM\_STR);

        $stmt\_contact->bindParam(":email", $user->contact->email, ***PDO***::PARAM\_STR);

        $stmt\_contact->bindParam(":user\_id", $user\_id, ***PDO***::PARAM\_STR);

        if(!$stmt\_contact->execute()){

            $conn->rollBack();

            return false;

        }

        $contact\_id  = $conn->lastInsertId();

        $stmt\_user = $conn->prepare("UPDATE users SET address\_id = :address\_id, contact\_id = :contact\_id WHERE id = :id");

        $stmt\_user->bindParam(":address\_id", $address\_id, ***PDO***::PARAM\_INT);

        $stmt\_user->bindParam(":contact\_id", $contact\_id, ***PDO***::PARAM\_INT);

        $stmt\_user->bindParam(":id", $user\_id, ***PDO***::PARAM\_INT);

        if(!$stmt\_user->execute()){

            $conn->rollBack();

            return false;

        }

        $conn->commit();

        return true;

    }

# Application Submission

Η διαδικασία μοιάζει πολύ με την διαδικασία που περιγράψαμε πιο πάνω. Με την διαφορά εδώ δεν χρειάζομαι κάποιο Dto λόγο του ότι ήδη το αντικείμενο μου είναι Flat και δεν έχει κάποιες εσωτερικές κλάσεις. Οπότε από το POST του HTML γεμίζω το αντικείμενό μου και το στέλνω σε αντίστοιχο Service (ApplicationService) σε μέθοδο submitApplication η οποία αυτή με την σειρά της κάνει κάποιο Validate και έπειτα το περνάει στο Repository το οποίο κάνει INSERT στην βάση.

HTML

if(isset($\_POST['date\_from']) && isset($\_POST['date\_to']) && isset($\_POST['reason\_text'])){

        $application = new Application();

        $application->date\_from = $\_POST['date\_from'];

        $application->date\_to = $\_POST['date\_to'];

        $application->reason = $\_POST['reason\_text'];

        $application->status = ApplicationStatus::Pending;

        $application->user\_id = $user->id;

        $applicationService->submitApplication($application);

    }

Sumbit Application (Application Service Method)

public function submitApplication(Application $application) : ResponseObject{

        global $applicationRepository;

        $validator = new ApplicationValidator();

        if(!$validator->Validate($application)){

            return new ResponseObject(1, $validator->validation\_error);

        }

        if(!$applicationRepository->add($application)){

            return new ResponseObject(1, "Something goes wrong and cannot submit your application");

        }

        return new ResponseObject(0, "Application has submitted successfully.");

    }

Validate

public function Validate($entity): bool

    {

        if(!isset($entity->date\_from) || $entity->date\_from == ""){

            settype($validation\_error, "string");

            $validation\_error = "From Date cannot be null or empty.";

            return false;

        }

        if(!isset($entity->date\_to) || $entity->date\_to == ""){

            settype($validation\_error, "string");

            $validation\_error = "To Date cannot be null or empty.";

            return false;

        }

        if(!isset($entity->reason) || $entity->reason == ""){

            settype($validation\_error, "string");

            $validation\_error = "The reason cannot be null or empty.";

            return false;

        }

        if(!isset($entity->status)){

            settype($validation\_error, "string");

            $validation\_error = "The status cannot be null or empty.";

            return false;

        }

        if($entity->status < ApplicationStatus::Pending || $entity->status > ApplicationStatus::Rejected){

            settype($validation\_error, "string");

            $validation\_error = "The status is not valid.";

            return false;

        }

        if(strtotime($entity->date\_from) > strtotime($entity->date\_to)){

            settype($validation\_error, "string");

            $validation\_error = "The Date From cannot be greater than Date To";

            return false;

        }

        return true;

    }

Insert (Repository Method)

public function add(Application $application): bool{

        global $conn;

        $stmt = $conn->prepare("INSERT INTO applications (date\_from, date\_to, reason, status, inserted\_date, user\_id)

        VALUES(:date\_from, :date\_to, :reason, :status, NOW(), :user\_id)");

        $stmt->bindParam(":date\_from", $application->date\_from, ***PDO***::PARAM\_STR);

        $stmt->bindParam(":date\_to", $application->date\_to, ***PDO***::PARAM\_STR);

        $stmt->bindParam(":reason", $application->reason, ***PDO***::PARAM\_STR);

        $stmt->bindParam(":status", $application->status, ***PDO***::PARAM\_INT);

        $stmt->bindParam(":user\_id", $application->user\_id, ***PDO***::PARAM\_INT);

        $conn->beginTransaction();

        if(!$stmt->execute()){

            $conn->rollBack();

            return false;

        }

        $application->id = $conn->lastInsertId();

        $conn->commit();

        return true;

    }

# Application Approve/Reject

Σε αυτό το σημείο δεν έχουμε ακριβώς την ίδια διαδικασία με πριν ως προς το HTML εδώ δράση αναλαμβάνει η JavaScript κάνοντας Ajax Call σε PHP Rest API.  
 Ο λόγος πίσω από αυτή την υλοποίηση είναι ότι θα χρειαζόταν να γραφτεί εκ νέου κώδικας σε κάθε περίπτωση οπότε αποφάσισα να μην το κάνω και αν ‘αυτού API.

API php file

<?php

use VacationPortal\Data\Model\Communication\NotificationHeader;

use VacationPortal\Helpers\Enumerations\ApplicationStatus;

use VacationPortal\Helpers\Enumerations\NotificationAction;

if(file\_exists("../autoload.php")){

    include\_once '../autoload.php';

}

else{

    include\_once '../../autoload.php';

}

    header('Content-Type: application/json; charset=utf-8');

    if($\_SERVER['REQUEST\_METHOD'] == 'GET'){

        if($\_REQUEST['method'] == 'approve'){

            echo approveApplication($\_REQUEST['uuid']);

        }

        else if($\_REQUEST['method'] == 'reject'){

            echo rejectApplication($\_REQUEST['uuid']);

        }

        else{

            echo json\_encode(array("status"=>false, "message"=>"The method does not exist."));

        }

    }

    else{

        echo json\_encode(array("status"=> false, "message"=>"You try make a request with a method that not supported."));

    }

    function approveApplication(string $uuid){

        global $applicationService;

        global $notificationService;

        global $user;

        if($uuid == ''){

            return json\_encode(array("status" => false, "message" => "Application with this ID does not exist."));

        }

        $applicationService->changeApplicationStatus($uuid, ApplicationStatus::Approved);

        $notificationHeader = new NotificationHeader();

        $notificationHeader->notification\_action = NotificationAction::AnswerToApplication;

        $notificationHeader->application\_id = $applicationService->getApplicationByUuid($uuid)->object->id;

        $notificationHeader->sender\_user\_id = $user->id;

        $notificationService->setNotification($notificationHeader);

        return json\_encode(array("status"=> true, "message" => "You have approve the application with ID: " . $uuid));

    }

    function rejectApplication(string $uuid){

        global $applicationService;

        global $notificationService;

        global $user;

        if($uuid == ''){

            return json\_encode(array("status" => 1, "message" => "Application with this ID does not exist."));

        }

        $applicationService->changeApplicationStatus($uuid, ApplicationStatus::Rejected);

        $notificationHeader = new NotificationHeader();

        $notificationHeader->notification\_action = NotificationAction::AnswerToApplication;

        $notificationHeader->application\_id = $applicationService->getApplicationByUuid($uuid)->object->id;

        $notificationHeader->sender\_user\_id = $user->id;

        $notificationService->setNotification($notificationHeader);

        return json\_encode(array("status"=> true, "message" => "You have reject the application with ID: " . $uuid));

    }

Οπότε ανάλογα της κλήσης που έρχεται καλώ και διαφορετικό function και από εδώ και πέρα είναι η ίδια διαδικασία με τις προηγούμενες. Καλώ το ApplicationService με την μέθοδο changeApplicationStatus στέλνοντας ένα Unique Identifier που δημιουργείτε αυτόματα κατά την δημιουργία της εγγραφής στη βάση με Trigger και έπειτα η μέθοδος του Service καλεί το Repository και κάνει Update την εγγραφή.

Όπως θα μπορούμε να παρατηρήσουμε μπαίνουν και τα Notifications εδώ τα οποία με τον ίδιο τρόπο και αυτά με κάποιο Service αναλαμβάνει όλη την διαδικασία.

Ajax Call

$('button[name="approve"]').click(function (e) {

***e***.preventDefault();

    const uuid = $(this).attr('data-uuid');

***$***.ajax({

      url: `/api/application-approve/${uuid}`,

      type: 'get',

      contentType: 'application/json; charset=utf-8',

      success: function (response) {

        if (***response***.status) {

***Swal***.fire(

            'Success',

            'You have approve the application.',

            'success'

          ).then(function (dialogue\_result) {

            if (***dialogue\_result***.isConfirmed) {

***document***.***location***.reload();

            }

          });

        }

        else {

***Swal***.fire(

            'Error',

            'Something went wrong. Please try again or contact with a administrator.',

            'error'

          );

        }

      },

      error: function (err) {

***Swal***.fire(

          'Error',

          'Something went wrong. With error: ' + err,

          'error'

        );

      }

    });

  });

  $('button[name="reject"]').click(function (e) {

***e***.preventDefault();

    const uuid = $(this).attr('data-uuid');

***$***.ajax({

      url: `/api/application-reject/${uuid}`,

      type: 'get',

      contentType: 'application/json; charset=utf-8',

      success: function (response) {

        if (***response***.status) {

***Swal***.fire(

            'Success',

            'You have reject the application.',

            'success'

          ).then(function (dialogue\_result) {

            if (***dialogue\_result***.isConfirmed) {

***document***.***location***.reload();

            }

          });

        }

        else {

***Swal***.fire(

            'Error',

            'Something went wrong. Please try again or contact with a administrator.',

            'error'

          );

        }

      },

      error: function (err) {

***Swal***.fire(

          'Error',

          'Something went wrong. With error: ' + err,

          'error'

        );

      }

    });

  });

Change Status (Service Method)

public function changeApplicationStatus(string $applicationId, int $applicationStatus):ResponseObject{

        global $applicationRepository;

        return new ResponseObject(0, "Application Status has changed", $applicationRepository->changeStatus($applicationId, $applicationStatus));

    }

Change Status (Repository Method)

public function changeStatus(string $applicationId, int $status) : bool{

        global $conn;

        $stmt = $conn->prepare("UPDATE applications

        SET status = :status

        WHERE uuid = :uuid");

        $stmt->bindParam(":status", $status, ***PDO***::PARAM\_INT);

        $stmt->bindParam(":uuid", $applicationId, ***PDO***::PARAM\_STR);

        $conn->beginTransaction();

        if(!$stmt->execute()){

            $conn->rollBack();

            return false;

        }

        $conn->commit();

        return true;

    }