### <?php

### /\*\*

### \* Connects to a database and allows accessing tables as properties. eg. `(new Database())->jobs->select('\*');`

### \* where 'jobs' is the name of a table in the database. Methods such as select, insert, update, etc. are available on the table object.

### \*/

### 

### namespace *Classes*;

### 

### use *\PDO*;

### use *\stdClass*;

### use *\PDOException*;

### 

### class Database extends *stdClass* {

### // In PHP 8.2+, dynamic properties are deprecated and removed in PHP 9 according to https://wiki.php.net/rfc/deprecate\_dynamic\_properties.

### // stdClass is an empty class that allows dynamically adding properties. Inheriting it is a work around (https://php.watch/versions/8.2/dynamic-properties-deprecated)

### private $pdo;

### 

### public function \_\_construct($user, $password, $dbname, $host = "mysql") {

### try {

### *$this*->pdo = new *PDO*("mysql:host=$host;dbname=$dbname", $user, $password);

### } catch (*PDOException* $e) {

### *$this*->pdo = new *PDO*("mysql:host=127.0.0.1;dbname=$dbname", $user, $password);

### }

### 

### // Using PDO::FETCH\_OBJ instead of PDO::FETCH\_ASSOC allows accessing columns as properties instead of array keys.

### *$this*->pdo->setAttribute(*PDO*::ATTR\_DEFAULT\_FETCH\_MODE, *PDO*::FETCH\_OBJ);

### }

### 

### /\*\*

### \* Instead of manually creating properties for all the tables in the database,

### \* I use the \_\_get magic method to create a new Table object when a property is accessed that doesn't exist.

### \*/

### public function \_\_get($name) {

### if (property\_exists(*$this*, $name) && isset(*$this*->$name))

### return *$this*->$name;

### 

### // I save the new Table object to the Database object so that it doesn't have to be created again.

### *$this*->$name = new *Table*(*$this*->pdo, $name);

### return *$this*->$name;

### }

### }

### <?php

### 

### namespace *Classes*;

### 

### class EntryPoint {

### public *Database* $db;

### public array $uriSegments;

### 

### public function \_\_construct(*Database* $db = null) {

### *$this*->db = $db ?? new *Database*('student', 'student', 'job');

### *$this*->uriSegments = explode('/', explode('?', $\_SERVER['REQUEST\_URI'])[0]);;

### }

### 

### public function run() {

### $location = *$this*->uriSegments[1] ?? '';

### $controllerPath = '\\Controllers\\' . ucfirst($location);

### 

### if (!class\_exists($controllerPath))

### // Load default controller if no controller exists for current location.

### $controller = new *\Controllers\Home*(*$this*->db);

### else

### // Otherwise, load the controller for the current location.

### // If path is "/admin/jobs", then $location is "admin" and controller class name is most likely "Admin"

### $controller = new $controllerPath(*$this*->db, *$this*->uriSegments);

### 

### $controller->run();

### 

### // Cannot forget the footer.

### include '../templates/footer.html.php';

### }

### 

### /\*\*

### \* Truncate a string if it exceeds a certain length

### \*

### \* @param string $str The string to be truncated

### \* @param int $len The maximum length of the string, defaults to 20

### \* @return string The truncated string with "..." appended if necessary

### \*/

### public static function sub(string $str, int $len = 20): string {

### return strlen($str) >= $len ? substr($str, 0, $len) . '...' : $str;

### }

### }

### 

### 

### <?php

### 

### /\*\*

### \* Page class contains methods related to pages eg. methods to create the head and footer of a page or a

### \* methods to check if a param is set on a page's URL, or to redirect to different page, or to check if a user is logged in,

### \* etc.

### \*

### \* @param *Database* $db - Database object for interacting with the database.

### \* @param array $categories - All categories in the database. Used for the navigation bar.

### \* @param boolean $testing - Makes the class and it's methods eg. staffOnly() work in testing mode.

### \*/

### 

### namespace *Classes*;

### 

### use *stdClass*;

### 

### class Page {

### public $db;

### public $categories;

### public $testing;

### 

### public function \_\_construct(*Database* $db, $testing = false) {

### *$this*->db = $db;

### *$this*->testing = $testing;

### }

### 

### /\*\* @return bool Checks whether the user is logged into an account on the page. \*/

### public function loggedIn(): bool {

### return isset($\_SESSION['loggedIn']);

### }

### 

### public function userInfo(): *stdClass* {

### $info = *$this*->db->account->select(['id' => $\_SESSION['loggedIn']]);

### // !$info && $this->logout();

### 

### return $info;

### }

### 

### /\*\* @return bool Checks whether the user is logged in as staff. \*/

### public function isStaff(): bool {

### return *$this*->loggedIn() && *$this*->userInfo()->isAdmin == true;

### }

### 

### /\*\* Logs the user out and redirects to the homepage. \*/

### protected function logout() {

### $\_SESSION = [];

### session\_destroy();

### *$this*->redirect('/');

### }

### 

### /\*\*

### \* Redirects to a URL and exits with an optional message.

### \*

### \* @param string $url - The URL to redirect to.

### \* @param string $message - Optional message to display upon redirect.

### \*/

### public function redirect(string $url, string $message = null) {

### if (headers\_sent()) {

### // Use JavaScript to redirect if headers have already been sent.

### // User will be told to click a link if JavaScript is disabled.

### $script = "<script>location.href='$url';</script>";

### exit("$script <p>" . ($message ?? "Failed redirecting to $url because headers have already been sent.") . "</p><p><a href='$url'> Click here to continue</a></p>");

### }

### 

### header("Location: $url");

### exit($message ?? "Redirecting... to $url");

### }

### 

### /\*\*

### \* Gets a param/field from $\_GET or $\_POST, if not found, exits with an error message.

### \*

### \* @param string $name - The name of the parameter to retrieve.

### \* @param boolean $required - Whether the parameter is required or not.

### \* @return mixed

### \*/

### public function param(string $name, $required = false) {

### $param = $\_GET[$name] ?? $\_POST[$name] ?? null;

### 

### if ($param === null && $required)

### // If a page that requires a param eg. 'id' is accessed without it, it probably means the user manually typed the URL.

### exit("Parameter '$name' not specified, you probably weren't meant to be here.");

### 

### return $param;

### }

### 

### public function appendQuery(string $query, string $url = ''): string {

### [$param, $value] = explode("=", $query);

### $url = $url ?: $\_SERVER['REQUEST\_URI'];

### // Remove param with the same name from the URL if it exists

### $url = preg\_replace("/[&?]?$param=[^&]+/", '', $url);

### $hasParams = parse\_url($url, PHP\_URL\_QUERY);

### // Return the URL with the new query appended

### return $url . ($hasParams ? '&' : '?') . $query;

### }

### 

### /\*\* Called on pages that require a user to be logged in as a staff member. \*/

### public function staffOnly(): *stdClass* {

### if (*$this*->testing)

### return *$this*->db->account->select(['id' => 1]);

### 

### if (!*$this*->loggedIn())

### *$this*->redirect('/admin', "You must log in to access this page.");

### 

### // Check that the user is part of staff using id from session

### $user = *$this*->db->account->select(['id' => $\_SESSION['loggedIn']]);

### 

### if (!$user)

### // User is not found in the database. Maybe account was deleted while user was logged in.

### *$this*->logout();

### 

### if ($user->isAdmin == 0)

### exit("<p>You must be a staff member to access this page.</p>");

### 

### return $user;

### }

### 

### /\*\* Creates doctype, html, head and title tags, etc. \*/

### public function createHead(string $title) {

### // Every page that needs a head and header also needs the categories for the navigation menu.

### // I assign the categories here so every pages can always have updated categories.

### *$this*->categories = *$this*->db->category->selectAll();

### 

### // Render the head with '$title' as well as the header and navigation menu.

### *$this*->renderTemplate('topsection', compact('title'));

### }

### 

### /\*\* Checks whether current user owns a job or is staff. \*/

### public function isOwnerOrAdmin(int $jobId): bool {

### if (!*$this*->loggedIn()) return false;

### 

### // Confirm that the jobs exists

### $job = *$this*->db->job->select(['id' => $jobId]);

### if (!$job) return false;

### 

### // Check that the current user is the creator of the job or is staff

### if ($job->accountId != $\_SESSION['loggedIn'] && !*$this*->isStaff())

### return false;

### 

### return true;

### }

### 

### 

### public function renderTemplate(string $template, array $data = []) {

### // Template will have access to variables in the scope eg. $this, $db, $data, etc.

### // It will not have access to variables in the scope of the function calling this function.

### // So, we use $data to pass the required variables into the scope which the template will have access to.

### extract($data);

### require "../templates/$template.html.php";

### }

### 

### public function renderPage(string $page, string $title = null, array $data = []) {

### // 'admin/login' becomes 'Admin Login'

### *$this*->createHead(ucwords(str\_replace('/', ' ', $title ?? $page)));

### 

### extract($data);

### require "../pages/$page.php";

### }

### }

### 

### 

### <?php

### /\*\*

### \* Used to access a table in a database. It should only be used by the Database class,

### \* unless you create your own PDO instace and pass it along with a table name to the constructor.

### \*/

### 

### namespace *Classes*;

### 

### class Table {

### private $pdo;

### private $table;

### 

### public function \_\_construct($pdoInstance, $table) {

### *$this*->pdo = $pdoInstance;

### *$this*->table = $table;

### }

### 

### /\*\*

### \* Wrapper function for the SELECT query.

### \*

### \* @param $binds any An array of binds and strings to be used in the condition eg. ['id' => 1] or ['id' => 1, 'AND', 'archived' => 0]. $bind keys must match the column names in the table.

### \* @param string|array $items Items to select. Can be an array of values eg. ['title', 'description'], a single string eg. 'title', or empty default for default '\*'

### \*

### \* Better explained with examples:

### \*

### \* $db->job->select([], 'id') result in:

### \* SELECT id FROM job

### \*

### \* $db->job->select(['id' => $jobId, 'AND', 'categoryId' => 1]) results in:

### \* SELECT \* FROM job WHERE id = :id AND categoryId = :categoryId

### \* Values are then bound to the query and executed.

### \*

### \* $db->job->select(['id' => $jobId, 'ORDER BY id']) or:

### \* job->selector(['id' => $jobId, 'ORDER', 'BY id']) result in:

### \* SELECT \* FROM job WHERE id = :id ORDER BY id

### \*

### \* $db->job->select(['id' => $jobId, 'archived' => 1]) results in:

### \* SELECT \* FROM job WHERE id = :id, archived = 1

### \* Which is invlid SQL because AND, OR, etc. are not used.

### \*

### \* $db->job->select(['id' => $jobId, 'OR', true, 'AND', '1 = 1'], ['id', 'title']) results in:

### \* SELECT id, title FROM job WHERE id = :id OR 1 AND 1 = 1

### \* true is automatically translated to 1 in the example above because true is just 1 and false is 0.

### \*/

### public function select($binds = [], $items = '\*') {

### return *$this*->selector($binds, $items)->fetch();

### }

### 

### /\*\* Same as select() but fetches all rows instead of just one. \*/

### public function selectAll($binds = [], $items = '\*') {

### return *$this*->selector($binds, $items)->fetchAll();

### }

### 

### /\*\* Inserts a row into the table. Example of $keysAndValues: ['title' => 'Job title'] \*/

### public function insert($keysAndValues = []) {

### $keys = array\_keys($keysAndValues);

### $fields = implode(', ', $keys);

### $values = implode(', :', $keys);

### 

### $statement = *$this*->pdo->prepare("INSERT INTO $this->table ($fields) VALUES (:$values)");

### $statement->execute($keysAndValues);

### return *$this*->pdo->lastInsertId();

### }

### 

### /\*\*

### \* Method for updating a row in the table.

### \* Unlike select() and selectAll(), $binds don't go first because 'WHERE'

### \* and other statements might not always be required but $values are always required.

### \*

### \* Usage examples:

### \* $db->job->update(['archived' => true], ['id' => $jobId]); results in:

### \* UPDATE job SET archived = :archived WHERE id = :id; which is then executed as:

### \* UPDATE job SET archived = 1 WHERE id = 1;

### \*/

### public function update(array $values, array $binds = []) {

### $fields = [];

### foreach ($values as $key => $value)

### $fields[] = "$key = :$key";

### 

### $fields = implode(', ', $fields);

### 

### $query = "UPDATE $this->table SET $fields";

### $statement = *$this*->bindAndExecute($binds, $query, $values);

### return $statement->rowCount();

### }

### 

### /\*\* Deletes a row from the table. $binds is the same as in select() and selectAll(). \*/

### public function delete(array $binds) {

### $query = "DELETE FROM $this->table";

### $statement = *$this*->bindAndExecute($binds, $query);

### return $statement->rowCount();

### }

### 

### /\*\*

### \* Searches the table for a string in the specified fields. This method allows using the LIKE operator unlike select() and selectAll().

### \*

### \* @param array $binds An array of fields and strings to search for eg. ['location' => %ampton%, 'title' => '%']

### \* @param array $additionalBinds An array of additional binds to be used in the query eg. ['name' => 'John', 'ORDER BY id']

### \*

### \* @return array An array of rows that match the search with additional binds applied if any.

### \*/

### public function search(array $binds, array $additionalBinds = []) {

### $query = "SELECT \* FROM $this->table WHERE ";

### foreach ($binds as $key => $value)

### $query .= "$key LIKE :$key AND ";

### 

### $query = substr($query, 0, -5); // Remove the last ' AND '

### // Combine the additional binds if any

### $query = *$this*->combine($additionalBinds, $query, 'AND');

### $statement = *$this*->bind($binds, $query, $additionalBinds);

### 

### $statement->execute();

### return $statement->fetchAll();

### }

### 

### /\*\*

### \* Function used by other fucntions to combine (not bind) keys and placeholder names in a string.

### \* Example: ['location' => 'Northampton'] becomes 'location = :location' after being passed to this function.

### \* It is then safe to use in a query and bind the values using the $this->bind() function or PDOStatement::execute() or PDOStatement::bindParam().

### \*/

### private function combine($binds, $query, $keyword, $debugMode = false): string {

### if (is\_string($binds))

### // $binds is a string eg. 'ORDER BY id', we just need to apped it to the query

### $query .= " $binds ";

### else if (!empty($binds)) {

### // $binds is an array of binds or strings eg. ['id' => 1, 'AND', 'archived' => 0]

### // In this case we need to use the WHERE keyword and bind the values

### $query .= " $keyword ";

### foreach ($binds as $key => $value)

### // If $key is an index then it means that the value is a condition (eg. AND) without a binding

### $query .= is\_int($key) ? " $value " : " $key = :$key ";

### }

### 

### // Use debugMode to see the final query

### $debugMode && var\_dump($query);

### 

### return $query;

### }

### 

### /\*\*

### \* @param $binds A string or array of SQL conditions and binds eg. ['id' => 1, 'AND', 'archived' => 0].

### \* @param string $query An incomplete query eg. 'SELECT \* FROM job'. The WHERE and others statements are added in and completed from this function.

### \* @param string $statement A statment which has possibly previously been prepared.

### \*/

### private function bindAndExecute($binds, $query, $values = [], $keyword = 'WHERE', $debugMode = false) {

### // Bind the WHERE and others statements

### $query = *$this*->combine($binds, $query, $keyword, $debugMode);

### 

### // Prepare query and bind all keys that are not integers to their values

### $statement = *$this*->pdo->prepare($query);

### 

### // Required by update() not select() and selectAll()

### foreach ($values as $key => $value)

### $statement->bindValue($key, $value);

### 

### if (is\_array($binds))

### foreach ($binds as $key => $value)

### if (!is\_int($key))

### $statement->bindValue($key, $value);

### 

### // If $debugMode is enabled, output query string with binds appended + $binds array.

### $debugMode && var\_dump($statement->queryString);

### $debugMode && var\_dump($binds);

### 

### // Execute the query and return the statement

### $statement->execute();

### return $statement;

### }

### 

### /\*\* Private function used by select() and selectAll() refer to them for documentation \*/

### private function selector($binds, $items) {

### $query = "SELECT ";

### 

### if (is\_array($items))

### $items = implode(', ', $items);

### 

### $query .= "$items FROM $this->table";

### $statement = *$this*->bindAndExecute($binds, $query);

### return $statement;

### }

### 

### private function bind($binds, $query, $additionalBinds = []){

### // Prepare query and bind $binds

### $statement = *$this*->pdo->prepare($query);

### foreach ($binds as $key => $value)

### $statement->bindValue($key, $value);

### 

### // Also bind the additional binds if any, ignoring keys that are integers

### foreach ($additionalBinds as $key => $value)

### if (!is\_int($key))

### $statement->bindValue($key, $value);

### 

### return $statement;

### }

### }

### 

### 

### <?php

### 

### namespace *Controllers*;

### 

### use *\Classes\Database*;

### use *\Classes\Page*;

### 

### class About extends *Page* {

### public function \_\_construct(*Database* $db) {

### *parent*::\_\_construct($db);

### }

### 

### public function run() {

### *$this*->renderPage("about");

### }

### }

### 

### 

### <?php

### 

### /\*\*

### \* Controller for the accounts page.

### \* It handles the logic for the accounts page and its subpages.

### \* Some methods are not pages, only method names that end with "Page" are pages.

### \*

### \* Manage accounts path

### \* =====================

### \* /admin/accounts

### \*

### \* Add account path

### \* =================

### \* /admin/accounts/modify

### \* /admin/accounts/modify?action=add

### \*

### \* Edit account path

### \* ==================

### \* /admin/accounts/modify?id=X

### \* /admin/accounts/modify?id=X&action=edit

### \*

### \* Delete account path

### \* ====================

### \* /admin/accounts/modify?id=X&action=delete

### \*/

### 

### namespace *Controllers*;

### 

### use *\Classes\Database*;

### use *\Classes\Page*;

### 

### class Accounts extends *Page* {

### protected $uriSegments;

### protected $subpage;

### protected $user;

### 

### public function \_\_construct(*Database* $db, array $uriSegment, bool $testing = false) {

### *parent*::\_\_construct($db, $testing);

### *$this*->uriSegments = $uriSegment;

### *$this*->subpage = *$this*->uriSegments[3] ?? '';

### *$this*->user = *$this*->staffOnly();

### }

### 

### public function run() {

### $page = "{*$this*->subpage}Page";

### $accountId = *$this*->param('id');

### $account = $accountId ? *$this*->db->account->select(['id' => $accountId]) : 0;

### 

### if ($accountId && !$account)

### *$this*->redirect('/admin/accounts', 'Account not found.');

### 

### if (method\_exists(*$this*, $page))

### return *$this*->{$page}($account);

### 

### *$this*->accountsPage($account);

### }

### 

### public function add($account, $form) {

### if ($errors = *$this*->validateAddForm($form))

### return (new *Error*(*$this*->db, $errors, 'Account Creation Error'))->run();

### 

### *$this*->db->account->insert([

### 'username' => $form['username'],

### 'password' => password\_hash($form['password'], PASSWORD\_DEFAULT),

### 'isAdmin' => $form['type'] === 'staff' ? 1 : 0

### ]);

### 

### *$this*->redirect('/admin/accounts', 'Account created.');

### }

### 

### public function edit($account, $form) {

### // Password field should be set, even if it's empty.

### $form['password'] = $form['password'] ?? '';

### $form['account'] = $account;

### 

### if ($errors = *$this*->validateEditForm($form))

### return (new *Error*(*$this*->db, $errors, 'Account Modification Error'))->run();

### 

### *$this*->db->account->update([

### 'username' => $form['username'],

### 'password' => $form['password'] ? password\_hash($form['password'], PASSWORD\_DEFAULT) : $account->password,

### 'isAdmin' => $form['type'] === 'staff' ? 1 : 0

### ], ['id' => $account->id]);

### 

### *$this*->redirect('/admin/accounts', 'Account updated.');

### }

### 

### public function validateAddForm($form) {

### $errors = [];

### 

### if (empty($form['password']))

### $errors[] = 'Password is required.';

### if (empty($form['type']))

### $errors[] = 'Account type is required.';

### if (empty($form['username']))

### $errors[] = 'Username is required.';

### else if (*$this*->db->account->select(['username' => $form['username']]))

### $errors[] = 'Username already exists, try another.';

### 

### return $errors;

### }

### 

### public function validateEditForm($form) {

### $errors = [];

### 

### if (empty($form['type']))

### $errors[] = 'Account type is required.';

### if (empty($form['username']))

### $errors[] = 'Username is required.';

### else if ($account2 = ($form['account2'] ?? *$this*->db->account->select(['username' => $form['username']])))

### if ($account2->id != $form['account']->id)

### $errors[] = 'Cannot change username to one that is already used by another account.';

### 

### return $errors;

### }

### 

### public function action(string $action, $account) {

### switch ($action) {

### case 'delete':

### *$this*->db->account->delete(['id' => $account->id]);

### *$this*->redirect('/admin/accounts', 'Account Deleted');

### break;

### case 'edit':

### *$this*->redirect('/admin/accounts/modify?id=' . $account->id);

### break;

### case 'add':

### *$this*->redirect('/admin/accounts/modify');

### break;

### default:

### *$this*->redirect('/admin/accounts', 'Invalid action.');

### }

### }

### 

### public function accountsPage() {

### $accountType = *$this*->param('type');

### $constraint = 'order by username asc limit 10';

### 

### *$this*->renderPage('admin/accounts', 'Accounts', [

### 'user' => *$this*->user,

### 'accountType' => $accountType,

### 'accounts' => *$this*->db->account->selectAll($accountType ? ['isAdmin' => $accountType === 'staff', $constraint] : $constraint)

### ]);

### }

### 

### /\*\* Page to add or edit an account depending on whether an id is specified. \*/

### public function modifyPage($account) {

### $action = *$this*->param('action');

### 

### if ($action)

### return *$this*->action($action, $account);

### 

### if (!*$this*->param('submit'))

### return *$this*->renderPage('admin/accountmodify', 'Account Management', [

### 'account' => $account,

### 'pageType' => $account ? 'Edit' : 'Add'

### ]);

### 

### // Form is submitted as POST but we'll merge it with $\_GET just in case.

### $form = array\_merge($\_POST, $\_GET);

### 

### *$this*->{$account ? 'edit' : 'add'}($account, $form);

### }

### 

### public function leftSection($account, $accountType) {

### *$this*->renderTemplate('leftsection-staff-accounts', [

### 'account' => $account,

### 'subpage' => *$this*->subpage,

### 'accountType' => $accountType,

### ]);

### }

### }

### 

### 

### <?php

### 

### namespace *Controllers*;

### 

### use *\Classes\Database*;

### use *\Classes\Page*;

### 

### class Admin extends *Page* {

### protected $uriSegments;

### protected $subpage;

### 

### public function \_\_construct(*Database* $db) {

### *parent*::\_\_construct($db);

### *$this*->uriSegments = explode('/', explode('?', $\_SERVER['REQUEST\_URI'])[0]);

### *$this*->subpage = *$this*->uriSegments[2] ?? '';

### }

### 

### public function run() {

### if (*$this*->subpage == 'logout')

### return *$this*->logout();

### 

### if (*$this*->subpage == 'login' || !*$this*->loggedIn())

### // All admin pages require a login. The Login controller will handle that.

### return (new *Login*(*$this*->db, *$this*->uriSegments))->run();

### 

### // Load controller with the same name as the subpage if it exist

### $class = '\\Controllers\\' . ucfirst(*$this*->subpage);

### if (class\_exists($class))

### return (new $class(*$this*->db, *$this*->uriSegments))->run();

### 

### // Call appropriate method for subpage if it exist or fallback to home

### *$this*->{method\_exists(*$this*, *$this*->subpage) ? *$this*->subpage : 'home'}();

### }

### 

### public function home() {

### *$this*->renderPage('admin/index', 'Admin Home');

### }

### }

### 

### 

### <?php

### 

### namespace *Controllers*;

### 

### use *\Classes\Database*;

### use *\Classes\Page*;

### 

### class Categories extends *Page* {

### protected array $uriSegments;

### protected $subpage;

### protected $user;

### 

### public function \_\_construct(*Database* $db, array $uriSegment) {

### *parent*::\_\_construct($db);

### *$this*->uriSegments = $uriSegment;

### *$this*->subpage = *$this*->uriSegments[3] ?? '';

### *$this*->user = *$this*->staffOnly();

### }

### 

### public function run() {

### $page = "{*$this*->subpage}Page";

### $categoryId = *$this*->param('id');

### $category = $categoryId ? *$this*->db->category->select(['id' => $categoryId]) : 0;

### 

### if ($categoryId && !$category)

### *$this*->redirect('/admin/categories', 'Category not found.');

### 

### if (method\_exists(*$this*, $page))

### return *$this*->{$page}($category);

### 

### *$this*->categoriesPage($category);

### }

### 

### public function edit($category) {

### $newName = *$this*->param('name', 1);

### 

### *$this*->db->category->update(['name' => $newName], ['id' => $category->id]);

### *$this*->redirect('/admin/categories', 'Category updated.');

### }

### 

### public function add() {

### $name = *$this*->param('name', 1);

### $category = *$this*->db->category->select(['name' => $name], 'id');

### 

### if ($category)

### exit("Category with the same name already exists. Want to

### <a href='/admin/categories/modify?id={$category->id}'>manage it</a>?");

### 

### *$this*->db->category->insert(['name' => $name]);

### *$this*->redirect('/admin/categories', 'Category added.');

### }

### 

### public function action(string $action, $category) {

### switch ($action) {

### case 'delete':

### *$this*->db->category->delete(['id' => $category->id]);

### *$this*->redirect('/admin/categories', 'Category Deleted');

### break;

### default:

### *$this*->redirect('/admin/categories', 'Invalid action.');

### }

### }

### 

### public function categoriesPage() {

### *$this*->renderPage('admin/categories', 'Categories', [

### 'categories' => *$this*->db->category->selectAll()

### ]);

### }

### 

### public function modifyPage($category) {

### $action = *$this*->param('action');

### 

### if ($action)

### return *$this*->action($action, $category);

### 

### if (!*$this*->param('submit'))

### return *$this*->renderPage('admin/categorymodify', 'Category Management', [

### 'category' => $category,

### 'pageType' => $category ? 'Edit' : 'Add'

### ]);

### 

### *$this*->{$category ? 'edit' : 'add'}($category);

### }

### 

### public function leftSection($category) {

### *$this*->renderTemplate('leftsection-staff-categories', [

### 'category' => $category,

### 'subpage' => *$this*->subpage

### ]);

### }

### }

### 

### 

### <?php

### 

### namespace *Controllers*;

### 

### use *\Classes\Database*;

### use *\Classes\Page*;

### 

### class Contact extends *Page* {

### public function \_\_construct(*Database* $db) {

### *parent*::\_\_construct($db);

### }

### 

### public function run() {

### *$this*->renderPage("contact");

### }

### 

### public function handleSubmit() {

### if (!*$this*->param('submit'))

### return 'For any enquiries, please contact us using the form below.';

### 

### $name = *$this*->param('name');

### $email = *$this*->param('email');

### $telephone = *$this*->param('telephone');

### $enquiry = *$this*->param('enquiry');

### 

### if (!($name && $email && $telephone && $enquiry))

### return 'Enqury not sent! Please fill in all fields.';

### 

### // Using regular expressions to check if number is valid.

### if (!preg\_match('/^(\+\d|0)7\d{9}$/', $telephone))

### return 'Enqury not sent! Phone number should start with 0 or country code (eg. +44) followed by 10 numbers.';

### 

### if (!filter\_var($email, FILTER\_VALIDATE\_EMAIL))

### return 'Enqury not sent! Please enter a valid email address.';

### 

### *$this*->db->enquiry->insert([

### 'name' => $name,

### 'email' => $email,

### 'telephone' => $telephone,

### 'enquiry' => $enquiry,

### ]);

### 

### return 'Enquiry sent. Got something else to say? Fill in the form below.';

### }

### }

### 

### 

### <?php

### 

### namespace *Controllers*;

### 

### use *\Classes\Database*;

### use *\Classes\Page*;

### use *\Classes\EntryPoint*;

### 

### class Enquiries extends *Page* {

### protected array $uriSegments;

### protected $subpage;

### protected $user;

### 

### public function \_\_construct(*Database* $db, array $uriSegment) {

### *parent*::\_\_construct($db);

### *$this*->uriSegments = $uriSegment;

### *$this*->subpage = *$this*->uriSegments[3] ?? '';

### *$this*->user = *$this*->staffOnly();

### }

### 

### public function run() {

### $page = "{*$this*->subpage}Page";

### $action = *$this*->param('action');

### $enquiryId = *$this*->param('id');

### $enquiry = $enquiryId ? *$this*->db->enquiry->select(['id' => $enquiryId]) : 0;

### 

### if ($enquiryId && !$enquiry)

### *$this*->redirect('/admin/enquiries', 'Enquiry not found.');

### 

### if (method\_exists(*$this*, $page))

### return *$this*->{$page}($enquiry);

### 

### if ($action)

### return *$this*->action($action, $enquiry);

### 

### *$this*->enquiriesPage($enquiry);

### }

### 

### public function edit($enquiry) {

### $type = *$this*->param('type', 1);

### $username = *$this*->param('username', 1);

### $password = *$this*->param('password');

### $enquiry2 = *$this*->db->enquiry->select(['username' => $username]);

### 

### if ($enquiry2 && $enquiry2->id != $enquiry->id)

### exit('Username already exists');

### 

### *$this*->db->enquiry->update([

### 'username' => $username,

### 'password' => $password ? password\_hash($password, PASSWORD\_DEFAULT) : $enquiry->password,

### 'isAdmin' => $type === 'staff' ? 1 : 0

### ], ['id' => $enquiry->id]);

### 

### *$this*->redirect('/admin/enquiries', 'Enquiry updated.');

### }

### 

### public function add() {

### $type = *$this*->param('type', 1);

### $username = *$this*->param('username', 1);

### $password = *$this*->param('password', 1);

### 

### if (*$this*->db->enquiry->select(['username' => $username]))

### exit('Username exists, try another.');

### 

### *$this*->db->enquiry->insert([

### 'username' => $username,

### 'password' => password\_hash($password, PASSWORD\_DEFAULT),

### 'isAdmin' => $type === 'staff' ? 1 : 0

### ]);

### 

### *$this*->redirect('/admin/enquiries', 'Enquiry created.');

### }

### 

### public function action(string $action, $enquiry) {

### switch ($action) {

### case 'delete':

### *$this*->db->enquiry->delete(['id' => $enquiry->id]);

### *$this*->redirect('/admin/enquiries', 'Enquiry Deleted');

### break;

### case 'complete':

### *$this*->db->enquiry->update(['completedBy' => *$this*->user->id], ['id' => $enquiry->id]);

### *$this*->redirect('/admin/enquiries', 'Enquiry marked as completed.');

### break;

### case 'incomplete':

### *$this*->db->enquiry->update(['completedBy' => null], ['id' => $enquiry->id]);

### *$this*->redirect('/admin/enquiries', 'Enquiry marked as incomplete.');

### break;

### default:

### *$this*->redirect('/admin/enquiries', 'Invalid action.');

### }

### }

### 

### public function enquiriesPage($enquiry) {

### $completedFilter = *$this*->param('completed') !== null;

### $incompleteFilter = *$this*->param('incomplete') !== null;

### 

### if (!$enquiry)

### $enquiries = *$this*->db->enquiry->selectAll();

### else {

### $enquiry = *$this*->db->enquiry->select(['id' => $enquiry->id]);

### if ($enquiry->completedBy)

### $enquiry->completedBy = *$this*->db->account->select(['id' => $enquiry->completedBy])->username ?? 'Unknown';

### }

### 

### *$this*->renderPage('admin/enquiries', 'Enquiries', [

### 'enquiry' => $enquiry ?? 0,

### 'enquiries' => $enquiries ?? [],

### 'completedFilter' => $completedFilter,

### 'incompleteFilter' => $incompleteFilter,

### // 'sub' => EntryPoint::sub // Undefined class constant 'sub' (PHP sucks)

### 'sub' => fn (...$args) => *EntryPoint*::sub(...$args)

### ]);

### }

### 

### /\*\* Page to add or edit an enquiry depending on whether an id is specified. \*/

### public function modifyPage($enquiry) {

### if (!*$this*->param('submit'))

### return *$this*->renderPage('admin/enquirymodify', 'Enquiry Management', [

### 'enquiry' => $enquiry,

### 'pageType' => $enquiry ? 'Edit' : 'Add'

### ]);

### 

### *$this*->{$enquiry ? 'edit' : 'add'}($enquiry);

### }

### 

### public function leftSection($data) {

### *$this*->renderTemplate('leftsection-enquiries', $data);

### }

### }

### 

### 

### <?php

### 

### namespace *Controllers*;

### 

### use *\Classes\Page*;

### use *\Classes\Database*;

### 

### class Error extends *Page* {

### protected array $errors;

### protected string $title;

### 

### public function \_\_construct(*Database* $db, array $errors, $title = "Error") {

### *parent*::\_\_construct($db);

### *$this*->errors = $errors;

### *$this*->title = $title;

### }

### 

### public function run() {

### *$this*->renderPage('error', *$this*->title, ['errors' => *$this*->errors]);

### *$this*->renderTemplate('footer'); // Footer wouldn't be auto-rendered since we exit before it.

### exit;

### }

### }

### 

### 

### <?php

### 

### namespace *Controllers*;

### 

### use *\Classes\Database*;

### use *\Classes\Page*;

### 

### class Faqs extends *Page* {

### public function \_\_construct(*Database* $db) {

### *parent*::\_\_construct($db);

### }

### 

### public function run() {

### *$this*->renderPage("faqs", "FAQs");

### }

### }

### 

### 

### <?php

### 

### namespace *Controllers*;

### 

### use *\Classes\Database*;

### use *\Classes\Page*;

### 

### class Home extends *Page* {

### public function \_\_construct(*Database* $db) {

### *parent*::\_\_construct($db);

### }

### 

### public function run() {

### // Array to select unarchived and unexpired jobs.

### $binds = ['archived' => 0, 'and', 'closingDate > NOW()'];

### $categoryId = *$this*->param('categoryId');

### $location = *$this*->param('location');

### 

### // Include category filter if set.

### if ($categoryId) {

### $binds[] = 'and';

### $binds['categoryId'] = $categoryId;

### }

### 

### // Order by closing date in ascending to show the jobs that are closing soonest first and limit to 10.

### $binds[] = 'order by closingDate asc limit 10';

### 

### // Also filter by location or any locations if location is not set and search.

### $jobs = *$this*->db->job->search(['location' => $location ? "%$location%" : "%"], $binds);

### 

### *$this*->renderPage("index", "Home", compact('jobs'));

### }

### }

### 

### 

### <?php

### 

### /\*\*

### \* Manage jobs path

### \* =====================

### \* /admin/jobs

### \*

### \* Add job paths

### \* =================

### \* /admin/jobs/modify

### \* /admin/jobs/modify?action=add

### \*

### \* Edit job paths

### \* ==================

### \* /admin/jobs/modify?id=X

### \* /admin/jobs/modify?id=X&action=edit

### \*

### \* Delete job path

### \* ====================

### \* /admin/jobs/modify?id=X&action=delete

### \*

### \* Archive/Unarchive job path

### \* ====================

### \* /admin/jobs/modify?id=X&action=archive

### \* /admin/jobs/modify?id=X&action=unarchive

### \*

### \* Job applintants path

### \* ====================

### \* /admin/jobs/applicants?id=X

### \*/

### 

### namespace *Controllers*;

### 

### use *\Classes\Database*;

### use *\Classes\Page*;

### 

### class Jobs extends *Page* {

### protected $uriSegments;

### protected $subpage;

### protected $adminPage; // /admin/jobs/\* or /jobs/\*

### 

### public function \_\_construct(*Database* $db, array $uriSegment, bool $testing = false) {

### *parent*::\_\_construct($db, $testing);

### *$this*->uriSegments = $uriSegment;

### *$this*->adminPage = *$this*->uriSegments[1] == 'admin';

### *$this*->subpage = *$this*->uriSegments[*$this*->adminPage ? 3 : 2] ?? '';

### }

### 

### public function run() {

### $page = "{*$this*->subpage}Page";

### $jobId = *$this*->param('id');

### 

### if ($jobId && *$this*->adminPage && !*$this*->isOwnerOrAdmin($jobId))

### *$this*->redirect('/admin/jobs', 'Job not found or you lack permission to manage it.');

### 

### if (method\_exists(*$this*, $page))

### return *$this*->{$page}($jobId);

### 

### *$this*->jobsPage($jobId);

### }

### 

### public function add(array $fields) {

### *$this*->db->job->insert($fields);

### *$this*->redirect('/admin/jobs', 'Job Added');

### }

### 

### public function edit(array $fields, $jobId) {

### *$this*->db->job->update($fields, ['id' => $jobId]);

### *$this*->redirect('/admin/jobs', 'Job Updated');

### }

### 

### /\*\* Action to delete, archive or unarchive a job. \*/

### public function action(string $action, $jobId) {

### switch ($action) {

### case 'delete':

### *$this*->db->job->delete(['id' => $jobId]);

### *$this*->redirect('/admin/jobs', 'Job Deleted');

### break;

### case 'archive':

### *$this*->db->job->update(['archived' => 1], ['id' => $jobId]);

### *$this*->redirect('/admin/jobs', 'Job Archived');

### break;

### case 'unarchive':

### *$this*->db->job->update(['archived' => 0], ['id' => $jobId]);

### *$this*->redirect('/admin/jobs', 'Job Unarchived');

### break;

### case 'edit':

### *$this*->redirect('/admin/jobs/modify?id=' . $jobId);

### break;

### case 'add':

### *$this*->redirect('/admin/jobs/modify');

### break;

### default:

### *$this*->redirect('/admin/jobs', 'Invalid action.');

### }

### }

### 

### public function jobsPage($jobId) {

### $job = null;

### $category = null;

### $location = *$this*->param('location');

### $categoryId = *$this*->param('categoryId');

### 

### // Path is /admin/jobs

### if (*$this*->adminPage) {

### $user = *$this*->userInfo();

### 

### $binds = [];

### if ($categoryId)

### // Jobs in the category created by the current user or all if user is staff.

### $binds = $user->isAdmin ? ['categoryId' => $categoryId] : ['categoryId' => $categoryId, 'AND', 'accountId' => $user->id];

### else

### // Above but without category filter.

### $binds = $user->isAdmin ? [] : ['accountId' => $user->id];

### 

### // Also filter by location or any locations if location is not set and search.

### $jobs = *$this*->db->job->search(['location' => $location ? "%$location%" : "%"], $binds);

### 

### return *$this*->renderPage('admin/jobs', 'Jobs', [

### 'jobs' => $jobs,

### 'jobId' => $jobId,

### 'subpage' => *$this*->subpage

### ]);

### }

### 

### // Path is /jobs

### if ($jobId) {

### // If jobId param is set, select the job and the category it belongs to.

### $job = *$this*->db->job->select(['id' => $jobId]);

### $categoryId = $job->categoryId ?? null;

### $category = *$this*->db->category->select(['id' => $categoryId]);

### } else if ($categoryId) {

### // Otherwise, if a categoryId param is set, select the category

### $category = *$this*->db->category->select(['id' => $categoryId]);

### if (!$category)

### $categoryId = null;

### }

### 

### // Select the category if no job or category is selected

### if (!$job && !$category) {

### $category = *$this*->db->category->select();

### $categoryId = $category->id ?? null;

### }

### 

### $categoryName = $category->name ?? null;

### $categoryId = $category->id ?? null;

### 

### // Fetch all jobs in the category that are not archived and have a closing date in the future with a location that matches the search term if one is set

### $binds = ['categoryId' => $categoryId, 'AND', 'archived' => 0, 'AND', 'closingDate > NOW()'];

### $jobs = *$this*->db->job->search(['location' => $location ? "%$location%" : "%"], $binds);

### 

### *$this*->renderPage('jobs', 'Jobs', compact(

### 'job',

### 'jobs',

### 'jobId',

### 'location',

### 'category',

### 'categoryId',

### 'categoryName'

### ));

### }

### 

### /\*\* Page to add or edit a job depending on whether an id is specified. \*/

### public function modifyPage($jobId) {

### $action = *$this*->param('action');

### 

### if ($action)

### return *$this*->action($action, $jobId);

### 

### if (!*$this*->param('submit'))

### return *$this*->renderPage('admin/jobmodify', 'Job Management', compact('jobId'));

### 

### $fields = *$this*->validateJobForm();

### *$this*->{$jobId ? 'edit' : 'add'}($fields, $jobId);

### }

### 

### public function applicantsPage($jobId) {

### $jobId || *$this*->redirect('/admin/jobs');

### 

### *$this*->renderPage('admin/jobapplicants', 'Applicants', compact('jobId'));

### }

### 

### public function applyPage($jobId) {

### $jobId || *$this*->redirect('/jobs', 'Job ID not specified.');

### $output = null;

### 

### if (*$this*->param('submit')) {

### $required = ['name', 'email', 'jobId', 'details'];

### foreach ($required as $field)

### if (empty(*$this*->param($field, 0)))

### $output = 'Please fill in all required fields';

### 

### // Error codes: https://www.php.net/manual/en/features.file-upload.errors.php#115746)

### if ($\_FILES['cv']['error'] == 4)

### $output = 'CV file not uploaded';

### else if ($\_FILES['cv']['error'] == 1 || $\_FILES['cv']['error'] == 2)

### $output = 'CV file is too large';

### else if ($\_FILES['cv']['error'] != 0)

### $output = 'There was an error uploading your CV';

### else {

### $parts = explode('.', $\_FILES['cv']['name']);

### $extension = end($parts);

### // $fileName = uniqid() . '.' . $extension;

### $fileName = "job$jobId-" . uniqid() . ".$extension";

### 

### move\_uploaded\_file($\_FILES['cv']['tmp\_name'], 'cvs/' . $fileName);

### 

### *$this*->db->applicant->insert([

### 'name' => $\_POST['name'],

### 'email' => $\_POST['email'],

### 'details' => $\_POST['details'],

### 'jobId' => $\_POST['jobId'],

### 'cv' => $fileName

### ]);

### 

### $output = 'Your application is complete. We will contact you after the closing date.';

### }

### }

### 

### *$this*->renderPage('apply', 'Apply', [

### 'job' => *$this*->db->job->select(['id' => $jobId, 'AND', 'archived' => 0]),

### 'output' => $output,

### ]);

### }

### 

### public function leftSection($jobId) {

### *$this*->renderTemplate('leftsection-staff-jobs', [

### 'jobId' => $jobId,

### 'subpage' => *$this*->subpage,

### ]);

### }

### 

### /\*\*

### \* Validates a job form and exits with an error message if invalid.

### \* A job form is submited after editing an existing job or adding a new one.

### \* @param array $form An array of fields to use instead of the form data.

### \* @return array An array of fields with values from the form if valid.

### \*/

### public function validateJobForm($form = []) {

### $errors = [];

### $form || $form = $\_POST;

### $fields = [

### 'title' => $form['title'] ?? '',

### 'description' => $form['description'] ?? '',

### 'salary' => $form['salary'] ?? '',

### 'location' => $form['location'] ?? '',

### 'categoryId' => $form['categoryId'] ?? '',

### 'closingDate' => $form['closingDate'] ?? '',

### 'accountId' => (*$this*->testing ? $form['accountId'] : $\_SESSION['loggedIn']) ?? '',

### ];

### 

### foreach ($fields as $field => $value)

### // == is used instead of === to allow empty strings

### if ($value == null)

### $errors[] = "Field '$field' is required.";

### 

### // Salary does not need to be a number, it can have a range eg. 20,000 - 30,000, currency symbols,

### // and other stuff the job poster might want to add, eg. a comment or information about the salary eg. "negotiable".

### // Feature to sort jobs by salary is not asked by the client, so we don't need a number anyway.

### 

### // Verify that the closing date is in the future

### if ($fields['closingDate'] && strtotime($fields['closingDate']) < time())

### $errors[] = "Closing date must be in the future.";

### 

### // Verify that category exists the in database

### if ($fields['categoryId'] && !*$this*->db->category->select(['id' => $fields['categoryId']]))

### $errors[] = "Category does not exist.";

### 

### if ($errors)

### if (*$this*->testing)

### // Return errors for testing instead of exiting

### return $errors;

### else

### // Exit to the error page

### exit((new *Error*(*$this*->db, $errors, 'Form Validation Error'))->run());

### 

### return $fields;

### }

### }

### 

### 

### <?php

### 

### namespace *Controllers*;

### 

### use *\Classes\Database*;

### use *\Classes\Page*;

### 

### class Login extends *Page* {

### public function \_\_construct($db) {

### *parent*::\_\_construct($db);

### }

### 

### public function run() {

### if (!*$this*->param('submit'))

### return *$this*->renderPage("admin/login");

### 

### // Login form is sent as POST but we'll merge it with $\_GET just in case.

### $form = array\_merge($\_POST, $\_GET);

### 

### if ($errors = *$this*->validateLoginForm($form))

### return (new *Error*(*$this*->db, $errors, 'Login Error'))->run();

### 

### if ($user = *$this*->db->account->select(['username' => $form['username']]))

### if (password\_verify($form['password'], $user->password))

### $\_SESSION['loggedIn'] = $user->id;

### 

### *$this*->redirect('/admin');

### }

### 

### public function validateLoginForm($form) {

### $username = $form['username'];

### $password = $form['password'];

### $errors = [];

### 

### !$username && $errors[] = 'Username is required.';

### !$password && $errors[] = 'Password is required.';

### 

### return $errors;

### }

### }

### 

### 

### <main *class*="sidebar">

### <?php *$this*->leftSection($account ?? null, $accountType ?? ''); ?>

### 

### <section *class*="right">

### <h2><?= $pageType ?> Account</h2>

### 

### <form *method*="post">

### <label *for*="username">Username</label>

### <input *name*="username" *value*="<?= $account->username ?? ''; ?>" *required* />

### <label *for*="password">Password</label>

### <input *name*="password" *type*="password" />

### <label *for*="type">Account type</label>

### <select *name*="type">

### <option *value*="staff" <?= $account->isAdmin ?? '' ? 'selected' : ''; ?>>Staff</option>

### <option *value*="client" <?= $account->isAdmin ?? '' ? '' : 'selected'; ?>>Client</option>

### </select>

### 

### <input *name*="submit" *type*="submit" *value*="<?= $pageType ?>" />

### </form>

### </section>

### </main>

### <main *class*="sidebar">

### <?php *$this*->leftSection($account ?? null, $accountType ?? ''); ?>

### 

### <section *class*="right">

### <h2>Accounts</h2>

### <div *class*="tablemenu">

### <a *class*="new" *href*="/admin/accounts/modify">Create</a>

### </div>

### 

### <table>

### <thead>

### <tr>

### <th *style*="width: 15%">Username</th>

### <th *style*="width: 15%">Account type</th>

### <th *style*="width: 50%">&nbsp;</th>

### <th *style*="width: 5%">&nbsp;</th>

### </tr>

### </thead>

### 

### <tbody>

### <?php foreach ($accounts as $account) { ?>

### <tr *class*="<?= $account->id === $user->id ? 'currentUser' : '' ?>">

### <td><?= $account->username ?></td>

### <td><?= $account->isAdmin ? 'Staff' : 'Client' ?></td>

### <td><a *href*="/admin/accounts/modify?id=<?= $account->id ?>" *class*="link" *style*="float: right;">Edit</a></td>

### <td><a *href*="/admin/accounts/modify?id=<?= $account->id ?>&action=delete" *class*="link delete" *style*="float: right;" *data-confirm*="Are you sure you want to delete <?= $account->id === $user->id ? 'your own account? You will be logged out.' : "the account '{$account->username}'" ?>?">Delete</a></td>

### </tr>

### <?php } ?>

### </tbody>

### </table>

### </section>

### </main>

### <main *class*="sidebar">

### <?php *$this*->leftSection($categories); ?>

### 

### <section *class*="right">

### <h2>Categories</h2>

### <a *class*="new" *href*="/admin/categories/modify">Add category</a>

### 

### <table>

### <thead>

### <tr>

### <th>Name</th>

### <th *style*="width: 5%">&nbsp;</th>

### <th *style*="width: 5%">&nbsp;</th>

### </tr>

### 

### <?php

### if (!($categories))

### echo '<tr><td>No categories found</td></tr>';

### else

### foreach ($categories as $category) { ?>

### <tr>

### <td><?= $category->name ?></td>

### <td><a *style*="float: right" *href*="/admin/categories/modify?id=<?= $category->id ?>">Edit</a></td>

### <td><a *style*="float: right" *href*="/admin/categories/modify?id=<?= $category->id ?>&action=delete" *class*="link delete" *data-confirm*="Are you sure you want to delete this category? All jobs in this category will also be deleted!">Delete</a></td>

### </tr>

### <?php } ?>

### </thead>

### </table>

### </section>

### </main>

### <main *class*="sidebar">

### <?php *$this*->leftSection($category); ?>

### 

### <section *class*="right">

### <h2><?= $pageType ?> Category</h2>

### <form *method*="post">

### <label *for*="name">Name</label>

### <input *type*="text" *name*="name" *value*="<?= $category->name ?? ''; ?>" *required* />

### <input *type*="submit" *name*="submit" *value*="<?= $pageType ?>" />

### </form>

### </section>

### </main>

### <main *class*="sidebar">

### <?php *$this*->leftSection(compact('enquiry', 'completedFilter', 'incompleteFilter')); ?>

### 

### <section *class*="right">

### <?php if ($enquiry) { ?>

### <h2 *style*="margin-bottom: 0;">Enquiry</h2>

### <div *class*="enquiry">

### <div>By <span *class*="name"><?= $enquiry->name ?></span> on <span *class*="created"><?= $enquiry->created ?></span></div>

### <h3>Contact Details</h3>

### <div>Email: <span *class*="email"><?= $enquiry->email ?? 'Not provided' ?></span></div>

### <div>Telephone: <span *class*="telephone"><?= $enquiry->telephone ?? 'Not provided' ?></span></div>

### <h3>Message</h3>

### <div *class*="message"><?= $enquiry->enquiry ?></div>

### <h3>Status</h3>

### <?php if ($enquiry->completedBy) { ?>

### <div *class*="status">

### <form *class*="status" *method*="post" *action*="/admin/enquiries?action=incomplete&id=<?= $enquiry->id ?>">

### Completed by <?= $enquiry->completedBy ?>.

### <label *for*="status" *class*="link" *style*="text-decoration-color: #444444b2;">Reopen</label>

### <input *type*="submit" *name*="status" *id*="status" *style*="display: none;" />

### </form>

### </div>

### <?php } else { ?>

### <form *class*="status" *method*="post" *action*="/admin/enquiries?action=complete&id=<?= $enquiry->id ?>">

### Incomplete.

### <label *for*="status" *class*="link" *style*="text-decoration-color: #444444b2;">Mark as complete</label>

### <input *type*="submit" *name*="status" *id*="status" *style*="display: none;" />

### </form>

### <?php } ?>

### </div>

### <?php } else { ?>

### <h2>Enquiries</h2>

### <?php if (!$enquiries) { ?>

### <p>No enquiries found</p>

### <?php } else { ?>

### <table>

### <thead>

### <tr>

### <th>Name</th>

### <th>Email</th>

### <th>Telephone</th>

### <th>Enquiry</th>

### <th>Created On</th>

### <th>Completed</th>

### </tr>

### </thead>

### 

### <tbody *class*="trlinks">

### <?php foreach ($enquiries as $enquiry) {

### if ($enquiry->completedBy) $enquiry->completedBy = *$this*->db->account->select(['id' => $enquiry->completedBy])->username ?? 'Unknown';

### if ($completedFilter && !$enquiry->completedBy) continue;

### if ($incompleteFilter && $enquiry->completedBy) continue; ?>

### <tr>

### <td><a *href*="/admin/enquiries?id=<?= $enquiry->id ?>"><?= $enquiry->name ?></a></td>

### <td><a *href*="/admin/enquiries?id=<?= $enquiry->id ?>"><?= $enquiry->email ?></a></td>

### <td><a *href*="/admin/enquiries?id=<?= $enquiry->id ?>"><?= $enquiry->telephone ?></a></td>

### <td><a *href*="/admin/enquiries?id=<?= $enquiry->id ?>"><?= $sub($enquiry->enquiry) ?></a></td>

### <td><a *href*="/admin/enquiries?id=<?= $enquiry->id ?>"><?= $enquiry->created ?></a></td>

### <td><a *href*="/admin/enquiries?id=<?= $enquiry->id ?>"><?= $enquiry->completedBy ? "By {$enquiry->completedBy}" : 'Not yet' ?></a></td>

### </tr>

### <?php } ?>

### </tbody>

### </table>

### <?php } ?>

### <?php } ?>

### </section>

### </main>

### <main *class*="sidebar">

### <?php *$this*->renderTemplate('leftsection-staff'); ?>

### 

### <section *class*="right">

### <?php if (*$this*->isStaff()) { ?>

### <h2>Dashboard</h2>

### <p>Welcome to the admin dashboard. Here you can manage all jobs, categories, and accounts.</p>

### <?php } else { ?>

### <h2>Dashboard</h2>

### <p>Welcome to the client dashboard. Here you can manage all jobs you created and see all applicants.</p>

### <?php } ?>

### <p>Start by selecting an option from the sidebar on the left.</p>

### </section>

### </main>

### <main *class*="sidebar">

### <?php *$this*->leftSection($jobId); ?>

### 

### <section *class*="right">

### <h2>Applicants for <?= $this->db->job->select(['id' => $jobId])->title; ?></h2>

### 

### <table>

### <thead>

### <tr>

### <th *style*="width: 10%">Name</th>

### <th *style*="width: 10%">Email</th>

### <th *style*="width: 65%">Details</th>

### <th *style*="width: 15%">CV</th>

### </tr>

### <?php foreach (*$this*->db->applicant->selectAll(compact('jobId')) as $applicant) { ?>

### <tr>

### <td><?= $applicant->name; ?></td>

### <td><?= $applicant->email; ?></td>

### <td><?= $applicant->details; ?></td>

### <td><a *href*="/cvs/<?= $applicant->cv; ?>">Download CV</a></td>

### </tr>

### <?php } ?>

### </thead>

### </table>

### </section>

### </main>

### <?php

### $pageType = $jobId ? 'Edit' : 'Add';

### $job = $jobId ? *$this*->db->job->select(['id' => $jobId]) : [];

### $jobId && !$job && *$this*->redirect('/admin/jobs');

### ?>

### 

### <main *class*="sidebar">

### <?php *$this*->leftSection($jobId); ?>

### 

### <section *class*="right">

### <h2><?= $pageType; ?> Job</h2>

### <form *method*="post">

### <label *for*="title">Title</label>

### <input *type*="text" *name*="title" *value*="<?= $job->title ?? ''; ?>" />

### 

### <label *for*="description">Description</label>

### <textarea *name*="description"><?= $job->description ?? ''; ?></textarea>

### 

### <label *for*="location">Location</label>

### <input *type*="text" *name*="location" *value*="<?= $job->location ?? ''; ?>" />

### 

### <label *for*="salary">Salary</label>

### <input *type*="text" *name*="salary" *value*="<?= $job->salary ?? ''; ?>" />

### 

### <label *for*="categoryId">Category</label>

### <select *name*="categoryId">

### <?php foreach (*$this*->categories as $row) { ?>

### <option *value*="<?= $row->id; ?>" <?= ($job->categoryId ?? '') == $row->id ? 'selected' : ''; ?>><?= $row->name; ?></option>

### <?php } ?>

### </select>

### 

### <label>Closing Date</label>

### <input *type*="date" *name*="closingDate" *value*="<?= $job->closingDate ?? ''; ?>" />

### 

### <input *type*="submit" *name*="submit" *value*="<?= $pageType; ?>" />

### </form>

### </section>

### </main>

### <main *class*="sidebar">

### <?php *$this*->leftSection($jobId, $subpage); ?>

### 

### <section *class*="right">

### <h2>Jobs</h2>

### <div *class*="tablemenu">

### <a *class*="new" *href*="/admin/jobs/modify">Add new job</a>

### <?php *$this*->renderTemplate('jobfilter'); ?>

### </div>

### 

### <table>

### <thead>

### <tr>

### <th>Category</th>

### <th>Title</th>

### <th>Salary</th>

### <th>Closing Date</th>

### <th *style*="width: 5%">&nbsp;</th>

### <th *style*="width: 5%">&nbsp;</th>

### <th *style*="width: 15%">&nbsp;</th>

### <th *style*="width: 5%">&nbsp;</th>

### </tr>

### </thead>

### 

### <tbody>

### <?php

### foreach ($jobs as $job) {

### $applicantCount = *$this*->db->applicant->select(['jobId' => $job->id], 'count(\*) as count');

### $category = *$this*->db->category->select(['id' => $job->categoryId]);

### ?>

### <tr>

### <td><?= $category->name ?></td>

### <td><?= $job->title ?></td>

### <td><?= is\_numeric(substr($job->salary, 0, 1)) ? '£' . $job->salary : $job->salary ?></td>

### <td><?= date('d/m/Y', strtotime($job->closingDate)) ?></td>

### <td><a *href*="/admin/jobs/modify?id=<?= $job->id ?>">Edit</a></td>

### <?php if ($job->archived) { ?>

### <td><a *href*="/admin/jobs/modify?id=<?= $job->id ?>&action=unarchive">Unarchive</a></td>

### <?php } else { ?>

### <td><a *href*="/admin/jobs/modify?id=<?= $job->id ?>&action=archive">Archive</a></td>

### <?php } ?>

### <td><a *style*="float: right" *href*="/admin/jobs/applicants?id=<?= $job->id ?>">View applicants (<?= $applicantCount->count ?>)</a></td>

### <td><a *style*="float: right" *href*="/admin/jobs/modify?id=<?= $job->id ?>&action=delete" *class*="link delete" *data-confirm*="Are you sure you want to delete this job?">Delete</a></td>

### </tr>

### <?php } ?>

### </tbody>

### </table>

### </section>

### </main>

### <main *class*="sidebar">

### <section *class*="left">

### <ul>

### <li><a *href*="/">Back Home</a></li>

### </ul>

### </section>

### 

### <section *class*="right">

### <h2 *style*="margin-left: 50px;">Log in</h2>

### 

### <form *method*="post" *style*="padding: 0 40px">

### <label *for*="username">Username</label>

### <input *name*="username" />

### <label *for*="password">Password</label>

### <input *name*="password" *type*="password" *required* />

### <input *name*="submit" *type*="submit" *value*="Log In" />

### </form>

### </section>

### </main>

### <main *class*="padded">

### <h2>About Us</h2>

### <p> We're a recruitment agency based in Northampton. We offer a range of different office jobs. <a *href*="/contact" *class*="link"><br>

### Get in touch</a> if you'd like to list a job with us.

### </p>

### </main>

### <main *class*="sidebar">

### <section *class*="left">

### <ul>

### <li><a *href*="/jobs">Other Jobs</a></li>

### <hr>

### <li><a *href*="/">Back Home</a></li>

### </ul>

### </section>

### 

### <section *class*="right">

### <?php if ($output) { ?>

### <p *class*="message"><?= $output ?></p>

### <?php } else { ?>

### <h2>Apply for '<?= $job->title; ?>'</h2>

### <form *method*="post" *enctype*="multipart/form-data">

### <label>Your name</label>

### <input *type*="text" *name*="name" *required* />

### 

### <label>E-mail address</label>

### <input *type*="text" *name*="email" *required* />

### 

### <label>Cover letter</label>

### <textarea *name*="details" *required*></textarea>

### 

### <label>CV</label>

### <input *type*="file" *name*="cv" *required* />

### 

### <input *type*="hidden" *name*="jobId" *value*="<?= $job->id; ?>" *required* />

### <input *type*="hidden" *name*="applying" *value*="true" />

### 

### <input *type*="submit" *name*="submit" *value*="Apply" />

### </form>

### <?php } ?>

### </section>

### </main>

### <main *class*="padded">

### <h2>Contact Us</h2>

### <p><?= $this->handleSubmit(); ?></p>

### <form *method*="post">

### <label *for*="name">Name</label>

### <input *type*="text" *name*="name" *id*="name" *required*>

### <label *for*="email">Email</label>

### <input *type*="email" *name*="email" *id*="email" *required*>

### <label *for*="telephone">Telephone</label>

### <input *type*="tel" *name*="telephone" *id*="telephone" *required*>

### <label *for*="enquiry">Enquiry</label>

### <textarea *name*="enquiry" *id*="enquiry" *required*></textarea>

### <input *type*="submit" *name*="submit" *value*="submit">

### </form>

### </main>

### <main *class*="home padded error">

### <h2><?= $title ?? 'Error' ?></h2>

### <ul>

### <?php foreach ($errors as $error) { ?>

### <li><?= $error ?></li>

### <?php } ?>

### </ul>

### <p><a *class*="link" *href*="<?= $\_SERVER['REQUEST\_URI'] ?>">Go back</a></p>

### </main>

### <main *class*="padded">

### <p>FAQs coming soon</p>

### </main>

### <main *class*="home padded">

### <p>Welcome to Jo's Jobs. <a *href*="/about" *class*="link">Find out about us</a></p>

### <h2 *style*="margin-top: 40px;">Jobs expiring soonest (10 max):</h2>

### 

### <div *class*="tablemenu">

### <?php *$this*->renderTemplate('jobfilter'); ?>

### </div>

### 

### <table>

### <thead>

### <tr>

### <th>Category</th>

### <th>Title</th>

### <th>Salary</th>

### <th>Location</th>

### <th>Closing Date</th>

### </tr>

### </thead>

### 

### <tbody *class*="trlinks">

### <?php

### foreach ($jobs as $job) {

### $applicantCount = *$this*->db->applicant->select(['jobId' => $job->id], 'count(\*) as count');

### $category = *$this*->db->category->select(['id' => $job->categoryId]);

### ?>

### <tr>

### <td><a *href*="/jobs?id=<?= $job->id ?>"><?= $category->name ?></td></a>

### <td><a *href*="/jobs?id=<?= $job->id ?>"><?= $job->title ?></td></a>

### <td><a *href*="/jobs?id=<?= $job->id ?>"><?= is\_numeric(substr($job->salary, 0, 1)) ? '£' . $job->salary : $job->salary ?></td></a>

### <td><a *href*="/jobs?id=<?= $job->id ?>"><?= $job->location ?></td></a>

### <td><a *href*="/jobs?id=<?= $job->id ?>"><?= date('d/m/Y', strtotime($job->closingDate)) ?></td></a>

### </tr>

### <?php } ?>

### </tbody>

### </table>

### </main>

### <main *class*="sidebar">

### <section *class*="left categories">

### <ul>

### <?php foreach (*$this*->categories as $cat) { ?>

### <li <?= $category && $cat->id == $categoryId ? 'class="current"' : '' ?>>

### <a *href*="<?= *$this*->appendQuery("categoryId={$cat->id}") ?>"><?= $cat->name ?></a>

### </li>

### <?php } ?>

### </ul>

### </section>

### 

### <section *class*="right">

### <h1><?= $categoryName ?></h1>

### <ul *class*="listing">

### <?php

### if (!$jobs){

### $string = '';

### if ($categoryId) $string .= ' in this category';

### if ($location) $string .= " with a location containing '$location'";

### if (!$string) $string .= ' yet';

### 

### echo "<p>No unexpired and unarchived jobs$string. <a href='/jobs'> Clear filters?</a></p>";

### } else {

### // If a jobId param is set, display the selected job first

### if ($job) { ?>

### <li>

### <div *class*="details">

### <h2><?= $job->title ?></h2>

### <h3><?= $job->salary ?></h3>

### <p><?= nl2br($job->description) ?></p>

### <a *class*="more" *href*="/jobs/apply?id=<?= $job->id ?>">Apply for this job</a>

### </div>

### </li>

### <?php }

### 

### // Display jobs that are not the selected one (or all jobs if no job is selected)

### foreach ($jobs as $job) { ?>

### <?php if ($job->id != $jobId) { ?>

### <li>

### <div *class*="details">

### <h2><?= $job->title ?></h2>

### <h3><?= $job->salary ?></h3>

### <p><?= nl2br($job->description) ?></p>

### <a *class*="more" *href*="/jobs/apply?id=<?= $job->id ?>">Apply for this job</a>

### </div>

### </li>

### <?php } ?>

### <?php } ?>

### <?php } ?>

### </ul>

### </section>

### </main>

### <?php

### 

### session\_start();

### 

### // Register a function to handle autoloading of classes

### spl\_autoload\_register(function ($className) {

### $classPath = str\_replace('\\', '/', '../' . lcfirst($className)) . '.php';

### 

### if (file\_exists($classPath))

### require\_once $classPath;

### });

### 

### $entryPoint = new *Classes\EntryPoint*();

### $entryPoint->run();

### 

### 

### <footer>

### &copy; Jo's Jobs <?= date('Y'); ?>

### </footer>

### <?php

### $categoryId = *$this*->param('categoryId');

### $location = *$this*->param('location');

### ?>

### 

### <details *class*="filter">

### <summary>Filter Jobs</summary>

### <form>

### <label *for*="category">Category</label>

### <select *name*="categoryId">

### <option *value*>All</option>

### <?php foreach (*$this*->categories as $category) { ?>

### <option *value*="<?= $category->id ?>" <?= $categoryId == $category->id ? 'selected' : '' ?>>

### <?= $category->name ?>

### </option>

### <?php } ?>

### </select>

### <label *for*="location">Location</label>

### <input *placeholder*="Enter location" *name*="location" *class*="nice" *value*="<?= $location ?>" />

### 

### <input *type*="submit" *value*="Filter" />

### </form>

### </details>

### <section *class*="left">

### <h3><?= $enquiry ? 'Options' : 'Filter' ?>:</h3>

### <?php if ($enquiry) { ?>

### <ul>

### <li>

### <?php if ($enquiry->completedBy) { ?>

### <a *href*="/admin/enquiries?id=<?= $enquiry->id ?>&action=incomplete">Reopen</a>

### <?php } else { ?>

### <a *href*="/admin/enquiries?id=<?= $enquiry->id ?>&action=complete">Close</a>

### <?php } ?>

### </li>

### <li><a *href*="/admin/enquiries?id=<?= $enquiry->id ?>&action=delete" *data-confirm*="Are you sure you want to delete this enquiry?">Delete</a></li>

### <hr>

### <li><a *href*="/admin/enquiries">All Enquiries</a></li>

### <?php } else { ?>

### <ul>

### <li><a *href*="?completed" *class*="<?= $completedFilter ? 'current' : '' ?>">Completed</a></li>

### <li><a *href*="?incomplete" *class*="<?= $incompleteFilter ? 'current' : '' ?>">Incomplete</a></li>

### <li><a *href*="/admin/enquiries" *class*="<?= !$completedFilter && !$incompleteFilter ? 'current' : '' ?>">All Enquiries</a></li>

### <?php } ?>

### <hr>

### <li><a *href*="/admin">Dashboard</a></li>

### </ul>

### </section>

### <section *class*="left">

### <?php if ($subpage == 'modify') { ?>

### <h3>Options:</h3>

### <ul>

### <?php if (isset($account->id)) { ?>

### <li><a *href*="/admin/accounts/modify?id=<?= $account->id ?>" *class*="<?= $account->id ? 'current' : '' ?>">Edit</a></li>

### <li *data-confirm*="Are you sure you want to delete '<?= $account->username ?>'?"><a *href*="/admin/accounts/modify?action=delete&id=<?= $account->id ?>" *class*="delete">Delete</a></li>

### <hr>

### <?php } ?>

### <li><a *href*="/admin/accounts/modify" *class*="<?= isset($account->id) ? '' : 'current' ?>">Add Account</a></li>

### <li><a *href*="/admin/accounts">All Accounts</a></li>

### <?php } else { ?>

### <h3>Type:</h3>

### <ul>

### <li><a *href*="?type=client" *class*="<?= $accountType == 'client' ? 'current' : '' ?>">Clients</a></li>

### <li><a *href*="?type=staff" *class*="<?= $accountType == 'staff' ? 'current' : '' ?>">Staff</a></li>

### <li><a *href*="/admin/accounts" *class*="<?= !$accountType ? 'current' : '' ?>">All</a></li>

### <?php } ?>

### <hr>

### <li><a *href*="/admin">Dashboard</a></li>

### </ul>

### </section>

### <section *class*="left">

### <h3>Options:</h3>

### <ul>

### <?php if (isset($category->id)) { ?>

### <li><a *href*="/admin/categories/modify?id=<?= $category->id ?>" *class*="<?= $category->id ? 'current' : '' ?>">Edit</a></li>

### <li *data-confirm*="Are you sure you want to delete the category '<?= $category->name ?>'?"><a *href*="/admin/categories/?action=delete&id=<?= $category->id ?>" *class*="delete">Delete</a></li>

### <hr>

### <?php } ?>

### <li><a *href*="/admin/categories/modify" *class*="<?= $subpage && !isset($category->id) ? 'current' : '' ?>">Add</a></li>

### <li><a *href*="/admin/categoieys" *class*="<?= $subpage ? '' : 'current' ?>">Categories</a></li>

### <hr>

### <li><a *href*="/admin">Dashboard</a></li>

### </ul>

### </section>

### <section *class*="left">

### <h3>Options:</h3>

### <ul>

### <?php if (isset($jobId)) { ?>

### <li><a *href*="/admin/jobs/modify?id=<?= $jobId ?>" *class*="<?= $subpage == 'modify' ? 'current' : '' ?>">Edit Job</a></li>

### <li><a *href*="/admin/jobs/applicants?id=<?= $jobId ?>" *class*="<?= $subpage == 'applicants' ? 'current' : '' ?>">Applicants</a></li>

### <hr>

### <?php } ?>

### <li><a *href*="/admin/jobs/modify" *class*="<?= $subpage == 'modify' && !$jobId ? 'current' : '' ?>">Add Job</a></li>

### <li><a *href*="/admin/jobs" *class*="<?= !$subpage ? 'current' : '' ?>">Jobs</a></li>

### <hr>

### <li><a *href*="/admin">Dashboard</a></li>

### </ul>

### </section>

### <?php /\*\* Left sidebar for the admin area. \*/ ?>

### 

### <section *class*="left">

### <h3>Options</h3>

### <ul>

### <li><a *href*="/admin/jobs">Manage Jobs</a></li>

### <?php if (*$this*->isStaff()) { ?>

### <li><a *href*="/admin/categories">Categories</a></li>

### <li><a *href*="/admin/accounts">Accounts</a></li>

### <li><a *href*="/admin/enquiries">Enquiries</a></li>

### <?php } ?>

### <hr>

### <li><a *href*="/admin/logout">Logout</a></li>

### </ul>

### </section>

### <?php

### 

### /\*\*

### \* This file should be required from the createHead method in classes/Pages.php.

### \* This script will create the head, header, and nav using varibles from the method.

### \*

### \* The body tag will be added and closed by the browser.

### \*/

### ?>

### 

### <!DOCTYPE *html*>

### <html>

### 

### <head>

### <title>Jo's Jobs - <?= $title ?></title>

### <link *rel*="stylesheet" *href*="/static/styles.css" />

### <script *src*="/static/script.js"></script>

### </head>

### 

### <header>

### <section>

### <aside>

### <h3>Office Hours:</h3>

### <p>Mon-Fri: 09:00-17:30</p>

### <p>Sat: 09:00-17:00</p>

### <p>Sun: Closed</p>

### </aside>

### <h1>Jo's Jobs</h1>

### </section>

### </header>

### 

### <nav>

### <ul>

### <li><a *href*="/">Home</a></li>

### <li>

### <a *href*="/jobs">Jobs</a>

### <div *class*="menudropdown">

### <section>

### <h3>Category</h3>

### <ul>

### <?php foreach (*$this*->categories as $category) { ?>

### <li><a *href*="/jobs?categoryId=<?= $category->id ?>"><?= $category->name ?></a></li>

### <?php } ?>

### </ul>

### </section>

### <!-- <section class="seperator">OR</section> -->

### <section>

### <h3>Location</h3>

### <form *action*="/jobs">

### <?php if (strpos($\_SERVER['REQUEST\_URI'], '/jobs') === 0) {

### if (isset($\_GET['categoryId']))

### echo "<input type='hidden' name='categoryId' value='{$\_GET['categoryId']}' />";

### if (isset($\_GET['location']))

### echo "<input type='hidden' name='location' value='{$\_GET['location']}' />";

### if (isset($\_GET['id']))

### echo "<input type='hidden' name='id' value='{$\_GET['id']}' />";

### } ?>

### 

### <input *placeholder*="Enter a location" *name*="location" *class*="nice" />

### <input *type*="submit" *value*="Apply" />

### </form>

### </section>

### </div>

### </li>

### <li><a *href*="/about">About Us</a></li>

### <li><a *href*="/faqs">FAQs</a></li>

### <li>

### <a>More</a>

### <div *class*="menudropdown" *style*="margin-left: 2em;">

### <section *class*="single">

### <a *href*="/contact">Contact</a>

### <a *href*="/admin">Dashboard</a>

### <?php if (*$this*->loggedIn()) { ?>

### <a *href*="/admin/logout">Logout</a>

### <?php } ?>

### </section>

### </div>

### </li>

### </ul>

### </nav>

### 

### <img *src*="/images/randombanner.php" />

### <?php

### 

### namespace *Tests*;

### 

### require \_\_DIR\_\_ . '/../classes/Table.php';

### require \_\_DIR\_\_ . '/../classes/Database.php';

### require \_\_DIR\_\_ . '/../classes/Page.php';

### require \_\_DIR\_\_ . '/../controllers/Accounts.php';

### 

### 

### use *PHPUnit\Framework\TestCase*;

### use *Classes\Database*;

### use *Controllers\Accounts*;

### 

### class AccountsTest extends *TestCase* {

### protected $db;

### protected $accountController;

### protected $uriSegments;

### 

### public function setUp(): void {

### $REQUEST\_URI = '/admin/accounts';

### 

### *$this*->db = new *Database*('student', 'student', 'job');

### *$this*->uriSegments = explode('/', explode('?', $REQUEST\_URI)[0]);

### *$this*->accountController = new *Accounts*(*$this*->db, *$this*->uriSegments, true);

### }

### 

### public function testAccountAddValidationForm() {

### $validForm = ['username' => 'test', 'password' => 'test', 'type' => 'test'];

### $noUsername = ['username' => '', 'password' => 'test', 'type' => 'test'];

### $noPassword = ['username' => 'test', 'password' => '', 'type' => 'test'];

### $noType = ['username' => 'test', 'password' => 'test', 'type' => ''];

### $usernameExists = ['username' => 'admin', 'password' => 'test', 'type' => 'test'];

### $noUsernamePasswordType = ['username' => '', 'password' => '', 'type' => ''];

### 

### // validateForm() returns an empty array if the form is valid otherwise it returns an array of error messages.

### *$this*->assertEquals([], *$this*->accountController->validateAddForm($validForm));

### 

### $noUsernameMessage = 'Username is required.';

### *$this*->assertContains($noUsernameMessage, *$this*->accountController->validateAddForm($noUsername));

### 

### $noPasswordMessage = 'Password is required.';

### *$this*->assertContains($noPasswordMessage, *$this*->accountController->validateAddForm($noPassword));

### 

### $noTypeMessage = 'Account type is required.';

### *$this*->assertContains($noTypeMessage, *$this*->accountController->validateAddForm($noType));

### 

### $usernameExistsMessage = 'Username already exists, try another.';

### *$this*->assertContains($usernameExistsMessage, *$this*->accountController->validateAddForm($usernameExists));

### 

### $noUsernamePasswordTypeMessage = [

### 'Password is required.',

### 'Account type is required.',

### 'Username is required.',

### ];

### 

### *$this*->assertEquals($noUsernamePasswordTypeMessage, *$this*->accountController->validateAddForm($noUsernamePasswordType));

### }

### 

### public function testAccountEdiValidationtForm() {

### // Mock account that we are editing.

### $account = *$this*->db->account->select(['id' => 1]);

### 

### $validForm = ['username' => 'test', 'type' => 'test', 'account' => $account];

### $noUsername = ['username' => '', 'type' => 'test', 'account' => $account];

### $noType = ['username' => 'test', 'type' => '', 'account' => $account];

### 

### *$this*->assertEquals([], *$this*->accountController->validateEditForm($validForm));

### 

### $noUsernameMessage = 'Username is required.';

### *$this*->assertContains($noUsernameMessage, *$this*->accountController->validateEditForm($noUsername));

### 

### $noTypeMessage = 'Account type is required.';

### *$this*->assertContains($noTypeMessage, *$this*->accountController->validateEditForm($noType));

### 

### $usernameExistsFrom = [

### 'username' => 'admin',

### 'type' => 'test',

### 'account' => $account,

### // In the actual function, 'account2' is an account queried from the database if the username is already taken.

### // The method first checks if 'account2' is set, if not it queries the database for an account that isn't the account we are editing.

### // Select test account that has the same username as the account we are editing.

### 'account2' => *$this*->db->account->select(['id' => 2]) // Notice the id is different.

### ];

### 

### $usernameExistsMessage = 'Cannot change username to one that is already used by another account.';

### *$this*->assertContains($usernameExistsMessage, *$this*->accountController->validateEditForm($usernameExistsFrom));

### 

### $usernameExistsFrom2 = [

### 'username' => 'admin',

### 'type' => 'test',

### 'account' => $account,

### 'account2' => $account // Same id and everything, it is the same account.

### ];

### 

### // Code below should not error because while new username is the same as the old one, it is the same account.

### // Same username is allowed in the form because maybe the user just wants to change the account type and left the username the same.

### *$this*->assertNotContains($usernameExistsMessage, *$this*->accountController->validateEditForm($usernameExistsFrom2));

### 

### // 'account2' is false, it means the username is not taken.

### *$this*->assertNotContains($usernameExistsMessage, *$this*->accountController->validateEditForm(['account2' => false]));

### *$this*->assertEquals([], *$this*->accountController->validateEditForm(array\_merge($validForm, ['account2' => false])));

### }

### }

### 

### 

### <?php

### 

### namespace *Tests*;

### 

### require \_\_DIR\_\_ . '/../controllers/Jobs.php';

### 

### use *PHPUnit\Framework\TestCase*;

### use *Classes\Database*;

### use *Controllers\Jobs*;

### 

### class JobsTest extends *TestCase* {

### protected $db;

### protected $jobsController;

### protected $uriSegments;

### 

### public function setUp(): void {

### $REQUEST\_URI = '/admin/jobs';

### 

### *$this*->db = new *Database*('student', 'student', 'job');

### *$this*->uriSegments = explode('/', explode('?', $REQUEST\_URI)[0]);

### *$this*->jobsController = new *Jobs*(*$this*->db, *$this*->uriSegments, true);

### }

### 

### public function testJobsFormValidation() {

### $validForm = [

### 'title' => 'test',

### 'description' => 'test',

### 'salary' => 'test',

### 'location' => 'test',

### 'categoryId' => '1',

### 'closingDate' => '2025-05-05',

### 'accountId' => '1',

### ];

### 

### foreach ($validForm as $field => $value) {

### // Iterate through each field in the $validForm array and create an invalid form by setting the current field to an empty string.

### $invalidForm = array\_merge($validForm, [$field => '']);

### // Then, assert that the returned errors array contains the message "Field '$field' is required." for that specific field.

### *$this*->assertContains("Field '$field' is required.", *$this*->jobsController->validateJobForm($invalidForm));

### }

### 

### $pastClosingDate = array\_merge($validForm, ['closingDate' => '2021-01-01']);

### *$this*->assertContains('Closing date must be in the future.', *$this*->jobsController->validateJobForm($pastClosingDate));

### 

### $nonexistentCategory = array\_merge($validForm, ['categoryId' => 65768673]);

### *$this*->assertContains('Category does not exist.', *$this*->jobsController->validateJobForm($nonexistentCategory));

### 

### // validateJobForm() returns an array of errors if the form is invalid.

### // If the form is valid, it returns the $validForm array (instead of an empty array) to be used by the calling method.

### // Below we assert that the returned array is the same as the $validForm array.

### *$this*->assertEquals($validForm, *$this*->jobsController->validateJobForm($validForm));

### }

### }

### 

### 

### <?php

### 

### namespace *Tests*;

### 

### require \_\_DIR\_\_ . '/../controllers/Login.php';

### 

### use *PHPUnit\Framework\TestCase*;

### use *Classes\Database*;

### use *Controllers\Login*;

### 

### class LoginTest extends *TestCase* {

### protected $db;

### protected $loginController;

### 

### public function setUp(): void {

### *$this*->db = new *Database*('student', 'student', 'job');

### *$this*->loginController = new *Login*(*$this*->db);

### }

### 

### public function testLoginFormValidation() {

### $validForm = ['username' => 'test', 'password' => 'test'];

### $noUsername = ['username' => '', 'password' => 'test'];

### $noPassword = ['username' => 'test', 'password' => ''];

### $noUsernamePassword = ['username' => '', 'password' => ''];

### 

### // validateLoginForm() returns an empty array if the form is valid otherwise it returns an array of error messages.

### *$this*->assertEquals([], *$this*->loginController->validateLoginForm($validForm));

### 

### $noUsernameMessage = 'Username is required.';

### *$this*->assertContains($noUsernameMessage, *$this*->loginController->validateLoginForm($noUsername));

### 

### $noPasswordMessage = 'Password is required.';

### *$this*->assertContains($noPasswordMessage, *$this*->loginController->validateLoginForm($noPassword));

### 

### $noUsernamePasswordMessage = [

### 'Username is required.',

### 'Password is required.',

### ];

### 

### *$this*->assertEquals($noUsernamePasswordMessage, *$this*->loginController->validateLoginForm($noUsernamePassword));

### }

### }

### 