Advanced Web Programming

JOBSHEET 4

MODEL AND ELOQUENT ORM

Ayu Jovita Widyadhari / 2241720219/2-I

POLITEKNIK NEGERI MALANG |

Link GitHub:

https://github.com/Jow05/PWL_POS

Practicum 1- How **$fillable** works

1. Defining Allowed Fields with $fillable:
   $fillable will insert the new data inputted, for example in the code trying to insert data level_id =2, username = manager_dua, name = manager_2, password using hash::make('12345').
   - ⇒ frameworks like Laravel, $fillable is a property within a model class that specifies which fields can be mass-assigned during data insertion or updates. It acts as a safeguard to prevent unintended alterations to sensitive fields.
2. Handling Password Hashing:
   Same like in previous steps, $fillable will enter the new data entered, for example in the code trying to enter data level_id =2, username = manager_tiga, name = manager_3, password using hash::make('12345').
   - ⇒ Passwords are treated differently due to security concerns. They are not included in $fillable. Instead, a separate mechanism (like hash::make() in your examples) is employed to hash passwords before storing them in the database. This makes it difficult for hackers to steal and use passwords even if they gain access to the data.

Practicum 2.1 – Retrieving Single Models

1. The code utilizes the find method to locate a specific record within the m_user data by directly referencing its primary key. In this instance, it seeks the record possessing a primary key value of 1.
2. The first method is employed to retrieve a single record from the m_user data based on certain criteria. Here, it filters records to those with a level_id value of 1, returning the first matching record it encounters.
3. Similar to the first method, firstWhere also retrieves a single record based on specified conditions. In this case, it fetches the first record from m_user that holds a level_id value of 1, mirroring the behavior of first.
4. The outcome of the retrieval process hinges on the existence of a matching record: - If a record with the sought-after primary key (1 in this scenario) is found, the $user variable stores this record, and subsequently, the "user" page is rendered, showcasing the retrieved data. - If no such record exists, a 404 error page is displayed, indicating the absence of the requested data.
5. The specific result of "page not found 404" arises due to a failed retrieval attempt—the system was unable to locate any record possessing a primary key of 20 within the m_user data.

Practicum 2.2 – Not Found Exceptions

1. In step 1, the FindOrFail method is used to search for records based on a specific primary key, if the record is not found, it throws a ModelNotFoundException exception. In the code above the record is found, generating user data based on the primary key searched.
2. The database data does not have 'manager9', so there is no record with the data 'manager9' the result is page not found

Practicum 2.3 – Retreiving Aggregrates

When running the above step, the web page '2 // app\Http\Controllers\UserController.php:50' will appear, where this code will count the number of users who have a level_id equal to 2 and print the value for debugging, and then render the 'user' view with the number of users.

Practicum 2.4 – Retreiving od Creating Models

1. The first step in the example above is the code used to create new user data in the database. The method used is firstOrCreate.
2. In this practicum we create new data in the database, namely username 'manager22', name 'Manager Dua Dua', level_id '2'.
3. The firstOrNew method will retrieve the data you want to find. If the searched data is in the database, then the searched result will be displayed.
4. Step 8, the database does not display the new data results because the inputted data does not exist / has not been saved.
5. Step 10, the database can display the new data result because the inputted data exists/has been saved by the save() method.

Practicum 2.5 – Attribute Changes

1. After the save()method is called, the $user->isDirty()code checks if there are any changes to the $user model. A false result indicates that no changes have been detected, so the data in the $user model remains the same.
2. On the other hand, the $user->wasChanged()method indicates that there were changes to the $user model after the save operation. This is useful for performing validation or other operations based on the changes.

Practicum 2.6 – Create, Read, Update, Delete (CRUD)

1. The webpage loads user data from the database.
   There are two links:
   "Add User": Leads to the /users/add route to add a new user.
   "Delete": Leads to the /users/delete route to delete the user.
2. The webpage displays an error message because the route /user/add_save is not defined.
3. When the "+add user" button is clicked, the webpage will display a form to add a new user.
   Users can fill in the form and click the "save" button to save the new user data.
4. When the "Change" button is clicked, the webpage will display a form to change the user data.
5. When the user data is changed and the "Change" button is clicked, the user data will be updated in the database.
6. When the "Hapus" button is clicked, the selected user data will be deleted from the database.

Practicum 2.7 – Relationships

1. The code above all contains data from the UserModel model and its level relations. The output generated from dd($user) will print the data for inspection.
2. This code all contains user data and adds new columns, namely level_code and level_name.