Gateway roundtable

Background

In legacy, pages are very idiosyncratic, and it can be difficult to write code that can reused across pages. For example in the Preapprovals rewrite, we use a query like the following to get a user's (Joe Shimel's) pending approvals.

```
SELECT
      report_date,
      pa_number,
      section,
      district,
      center_code,
      center_description,
      pay_center,
      user_id,
      system_entry_date,
      purchase_description,
      ncas_account,
      account_description,
      unit_quantity,
      unit_cost,
      sum(unit_quantity*unit_cost) as 'requested_amount',
      purchase_type,
      purchase_date,
      justification,
      center_approved,
      district_approved,
      disu_comments,
      section_approved,
      section_comments,
      division_approved,
      bo_comments,
      osbm_approved,
      re_number
FROM purchase_request_3
WHERE
    section_approved='u' AND
    division_approved='u' AND
    (center_code IN ('ADMI', 'ADMN', 'ASRO', 'BOCR', 'CABE', 'CACR', 'CEDI', 'CENT',
'CHRO', 'CLNE', 'COMM', 'CONC', 'CRMO', 'DERI', 'DIRE', 'DISW', 'EDGS', 'ELKN',
'ENRI', 'FALA', 'FITE', 'FOFI', 'FOMA', 'FULA', 'GISP', 'GLSG', 'GOCR', 'GORG',
'GRAN', 'GRMO', 'GRST', 'HABE', 'HARI', 'HARO', 'INTE', 'ITOP', 'ITPR', 'JONE',
'JORD', 'JORI', 'KELA', 'LAHY', 'LAJA', 'LANO', 'LAPP', 'LAWA', 'LOHA', 'LURI',
'MARI', 'MEMI', 'MEMO', 'MOJE', 'MOMI', 'MOMO', 'MOTS', 'NARA', 'NDTO', 'NEDI',
'NERI', 'NWDI', 'OPRA', 'PACR', 'PAPL', 'PETT', 'PIMO', 'PIVI', 'PMET', 'PUIS',
'RARO', 'REEV', 'RELE', 'REMA', 'REMO', 'RERE', 'SACR', 'SAFE', 'SEDI', 'SILA',
'SOMO', 'SPHR', 'STMO', 'STPA', 'STWD', 'SUMT', 'SWDI', 'TRGR', 'VADE', 'VOLS',
'WAMI', 'WARE', 'WEBS', 'WEWO', 'WILD', 'WIUM', 'YEMO', 'dede', 'none', 'stwd',
'usfw') OR user_id = 'Shimel63')
GROUP BY pa_number
```

```
ORDER BY system_entry_date DESC
LIMIT 2500
```

It's generated with the following function:

```
// arguments omitted & query shortened for brevity
function whereClause($temp_id, $params) {
    // example of using one of the optional params
    if (isset($params['entered_before'])) {
        $where .= " AND system_entry_date <= '" . $params['entered_before'] . "'";
    }

    // ... more optional params ....

    // temp_id is required
    $highest = getHighestApprovalLevel($temp_id) . "_approved";
    $where = $where . " AND $highest = 'u'";

    return $where;
}


function selectPreapprovals($connection, $params, $temp_id) {
    $where = whereClause($temp_id, $params);
    $sql = "SELECT
                ...
            WHERE $where"
    $result = mysqli_query($connection, $sql);

    $arr = [];
    while ($row = mysqli_fetch_assoc($result)) {
        $arr[] = $row;
    }
    return $arr;
}
```

For Preapprovals, Zelda wrote a component that displays the amount of pending approvals a user has:

Looks good Zelda!

To go the value into her component she could have just copied & pasted my query into her component. For a more mature page, this isn't such a big deal, we do this all the time. But since I was still actively working on my page, she needed a way to replicate the behavior of my query without having to manually update her copy each time I tweaked it.

To do this, we refactored my query into a *Gateway Class.*

What is a Gateway Class?

A Gateway class put simply, is a container for static functions. It doesn't have a constructor or any internal state. Here's an example:

```
// WebServer/Documents/_globals/Budget/PreApprovals.php

namespace Budget;

class PreApprovals {

    public static function helloWorld() {
        return "Hello, world!"
    }

}

// WebServer/Documents/budget/aDiv/some_page.php
$hello_world = \Budget\PreApprovals::helloWorld();
/*              ^      ^             ^
                |      |             |
                |      class name    function name
                |
                namespace
*/

// you might prefer this syntax, they do the same thing
$PreApprovals = new \Budget\PreApprovals();
$hello_world_alt = $PreApprovals->helloWorld();
```

Instead of manually including them, Gateway classes are loaded by our autoloader in `globalFunctions.php`. So if your page includes `globalFunctions`, you can reference any class written in the `_global` folder.

Let's move the whereClause function into the Gateway class for Preapprovals, and write a function for Zelda to use:

```
// WebServer/Documents/_globals/Budget/PreApprovals.php

namespace Budget;

class PreApprovals {

    public static function getHighestApprovalLevel($temp_id) {
        // returns 'cashier', 'manager', 'district', or 'section'
    }

    public static function whereClause($temp_id, $params = null) {
        $where = "";
        if ($params == null) {
            // default behaviour if no page params are set
        } else {
            // ... do something with the params ...
        }
        $highest = getMyHighestApprovalLevel($temp_id) . "_approved";
        $where = $where . " AND $highest = 'u'";
```

```
        return $where;
    }

    public static function getPendingApprovalCount($connection, $temp_id) {
        $connection = //...
        $where = whereClause($temp_id); // omit params since this function is called
on multiple pages
        $sql = "SELECT pa_number FROM purchase_request_3 WHERE $where";
        $result = mysqli_query($connection, $sql);
        return mysqli_num_rows($result);
    }

}
```

Awesome! Now Zelda can use `getPendingApprovalCount` without knowing anything about my page, and I can change the behavior of my query and the number in her component will change as well.

But Zelda is still using this component across multiple pages. If she changes how it looks on one, she'll have to manually change how it looks on all of them. She tried using `include` to link her file, but she was running into issues with variable names she used in her file overwriting the variables set on the pages she used. Let's use a Gateway class to solve both of these problems:

```
<?php
// WebServer/Documents/_globals/Budget/PreApprovals.php
namespace Budget;

class PreApprovals {

    public static function getHighestApprovalLevel($temp_id) {
        // returns 'cashier', 'manager', 'district', or 'section'
    }

    public static function whereClause($temp_id, $params = null) {
        $where = "";
        if ($params == null) {
            // default behaviour if no page params are set
        } else {
            // ... do something with the params ...
        }
        $highest = getMyHighestApprovalLevel($temp_id) . "_approved";
        $where = $where . " AND $highest = 'u'";
        return $where;
    }

    public static function getPendingApprovalCount($temp_id) {
        $connection = //...
        $where = whereClause($temp_id); // omit params since this function is called
on multiple pages
        $sql = "SELECT pa_number FROM purchase_request_3 WHERE $where";
        $result = mysqli_query($connection, $sql);
        return mysqli_num_rows($result);
    }

    public static function renderMyPendingApprovalsButton($temp_id) {
        $targetUrl = '/budget/aDiv/park_purchase_request_view.php';
```

```
        $pending = self::getPendingApprovalCount($temp_id);
        $label = $pending > 0 ? "Pending Approvals ($pending)" : "No Pending
Approvals";
        // exit PHP tag and render HTML
        ?>
            <a href="<?= htmlspecialchars($targetUrl); ?>">
                <button type="submit"
                    style="background-color:#007bff;
                           color:#fff;
                           border:none;
                           padding:8px 16px;
                           border-radius:4px;
                           cursor:pointer;
                           font-size:14px;">
                    <?= htmlspecialchars($label); ?>
                </button>
            </a>
        <?php
    }
```

Now when Zelda wants to use her button she just writes:

```
<?php
// some_other_page.php
$temp_id = $_SESSION['budget']['tempID'];
// ...
<?php \Budget\PreApprovals::renderMyPendingApprovalsButton($temp_id); ?>
```

TLDR

In summary, we can use Gateway classes to:

- avoid manually updating copied & pasted code
- replicate a page's functionality without having to understand all it's edge cases
- avoid overwriting variables set in other files
- split business logic into digestible steps without side effects