```php
<?php
declare(strict_types=1);

namespace Tithely\Giving\Infrastructure\Query\Deposit;

use DateTime;
use GuzzleHttp\Client;
use Exception;
use DateTimeImmutable;
use GuzzleHttp\Exception\GuzzleException;
use Cache\Adapter\Common\Exception\CacheException;
use Tithely\Common\Infrastructure\Query\Query;
use Tithely\Giving\Infrastructure\Auth\UserContext;
use Tithely\Giving\Infrastructure\Query\DepositAccount\DepositAccount;
use Tithely\Giving\Infrastructure\Query\DepositAccount\DepositAccountFinder;
use Tithely\Giving\Infrastructure\Query\ExternalFinderAbstract;
use Tithely\Giving\Infrastructure\Query\FinderInterface;
use Tithely\Giving\Infrastructure\Query\Organization\Location;
use Tithely\Giving\Infrastructure\Query\Organization\OrganizationFinder;
use Tithely\Giving\Infrastructure\Query\Payer\Payer;
use Tithely\Giving\Infrastructure\Query\ReconciledDeposit\ReconciledDeposit;
use Tithely\Giving\Infrastructure\Query\ReconciledDeposit\ReconciledDepositFinder;
use Tithely\Giving\Infrastructure\Query\Transfer\Transfer;
use function Functional\first;
use function Functional\map;

class DepositFinder extends ExternalFinderAbstract implements FinderInterface
{
    private DepositAccountFinder     $accountFinder;
    protected Client                 $client;
    private UserContext              $context;
    private OrganizationFinder       $orgFinder;
    private ReconciledDepositFinder  $reconciledFinder;

    /**
     * @param UserContext             $context
     * @param OrganizationFinder      $orgFinder
     * @param DepositAccountFinder    $accountFinder
     * @param ReconciledDepositFinder $reconciledFinder
     * @param Client                  $client
     */
    public function __construct(
        UserContext $context,
        OrganizationFinder $orgFinder,
        DepositAccountFinder $accountFinder,
        ReconciledDepositFinder $reconciledFinder,
        Client $client
    ) {
        $this->context = $context;
        $this->orgFinder = $orgFinder;
        $this->accountFinder = $accountFinder;
        $this->reconciledFinder = $reconciledFinder;
        $this->client = $client;
    }

    /**
     * Finds Deposit matching the criteria.
     *
     * @param Query $query
     * @return DepositList
     * @throws Exception
     * @throws GuzzleException
     * @todo refactor the reconciled flag usage to honor the flag on the TS response item
     */
    public function find(Query $query): DepositList
    {
        $qs = $this->buildQueryString($query);
```

```
        $url = sprintf('/organization/%s/deposits', $this->context->getOrganizationId());

        $depositData = [];

        while (true) {
            $response = $this->findManyExternally($url, $qs);
            $depositData = array_merge($depositData, $response['data']);
            $total = $response['navigation']['count'];

            if ($query->getLimit() || !$response['navigation']['next']) {
                break;
            }

            $qs['page'] ??= 1;
            $qs['page']++;
        }

        return $this->deserialize($depositData, $total);
    }

    /**
     * Get a single Deposit record
     *
     * @param Query $query
     * @return Deposit|null
     * @throws Exception
     * @throws GuzzleException
     */
    public function findOne(Query $query): ?Deposit
    {

        $qs = $this->buildQueryString($query);
//          $qs['transfers_page_size'] = 2;
        $qs['transfers_page'] = 1;

        $depositTransfersData = [];

        while (true) {
            $url = sprintf(
                '/organization/%s/deposits/%s',
                $this->context->getOrganizationId(),
                $query->getParameter('id')
            );
            $response = $this->findManyExternally($url, $qs);
            $depositData = array_merge($depositTransfersData, $response['transfers']['data']);

            if (!$response['transfers']['navigation']['next']) {
                break;
            }

            $qs['page'] ??= 1;
            $qs['page']++;
        }

        if (empty($depositData)) {
            return null;
        }
var_dump($depositTransfersData);exit();
        return current(
            $this->deserialize([$depositData], 1)->items()
        );
    }

    /**
     * Render raw deposit data to a Deposit instance
     *
```

```
     * @param array $depositData
     * @param int   $totalItems
     * @return DepositList
     * @throws GuzzleException
     * @throws CacheException
     * @throws Exception
     */
    private function deserialize(
        array $depositData,
        int $totalItems
    ): DepositList {
        $depositAccounts = $this->accountFinder->find(
            Query::build()->with('organization_id', $this->context->getOrganizationId())
        );

        $reconciliations = $this->reconciledFinder->find(
            Query::build()
        );

        $organization = $this->orgFinder->findOne(
            Query::build()->with('id', $this->context->getOrganizationId())
        );

        $deposits = map(
            $depositData,
            function (array $deposit) use ($depositAccounts, $organization, $reconciliations):
Deposit {
                $depositAccount =
                    $depositAccounts->getById($deposit['deposit_account_id']) ??
                    new DepositAccount(
                        $deposit['deposit_account_id'],
                        null,
                        null,
                        null,
                        null,
                        null,
                        null,
                        null,
                        null
                    );

                $location = first(
                    $organization->locations(),
                    fn (Location $location) => $location->legacyId() == $depositAccount-
>drupalLocId()
                );

                /** @var ReconciledDeposit $reconciled */
                $reconciled = $reconciliations->filter(
                    fn (ReconciledDeposit $reconciledDeposit) => $reconciledDeposit-
>depositId() === $deposit['id']
                )->first();

                // @todo supply looped data for transfers to account for deposits with more
than 100 transfers.
                return new Deposit(
                    $deposit['id'],
                    $depositAccount,
                    is_null($deposit['available_on']) ? null : new DateTime('@' .
$deposit['available_on']),
                    is_null($deposit['created']) ? null : new DateTime('@' .
$deposit['created']),
                    $deposit['currency'],
                    $deposit['description'],
                    $deposit['gross_amount'] ?? 0,
                    $deposit['net_amount'],
                    map(
```

```
                        $deposit['transfers']['data'],
                        fn (array $transfer) => new Transfer(
                            $transfer['id'],
                            new DateTimeImmutable('@' . $transfer['created']),
//                              $transfer['currency'],
                            $transfer['description'] ?: null,
                            $transfer['fee'],
                            $transfer['gross'],
                            $transfer['net'],
                            !empty($transfer['payer'])
                                ? new Payer(
                                    $transfer['payer']['id'],
                                    null,
                                    $transfer['payer']['drupal_id'],
                                    $transfer['payer']['email'],
                                    $transfer['payer']['first_name'],
                                    $transfer['payer']['last_name'],
                                    $transfer['payer']['name'],
                                    $transfer['payer']['people_ids'],
                                    $transfer['payer']['phone'],
                                )
                                : null,
                            $transfer['payment_category']['id'] ?? null,
                            $transfer['payment_category']['product'] ?? null,
                            $transfer['payment_category']['drupal_id'] ?? null,
                            $transfer['status'],
                            $transfer['transfer_type'],
                            $transfer['payment_id'],
                            $transfer['refund_id'] ?: null,
                        )
                    ),
                    $deposit['transaction_ids'] ?? null,
                    strtolower($deposit['status']),
                    $location,
                    $reconciled,
                    $deposit['refund_count'] ?? null,
                    $deposit['refund_ids'] ?? null
                );
            }
        );

        return new DepositList($deposits, $totalItems);
    }

    /**
     * Build an array of query params to be sent in the call
     *
     * @param Query $query
     * @return array
     */
    public function buildQueryString(Query $query): array
    {
        $qs = [];

        if ($query->getCursor()) {
            $qs['page'] = $query->getCursor();
        }

        if ($query->getLimit()) {
            $qs['page_size'] = $query->getLimit();
        }

        if ($query->hasParameter('available_on_before')) {
            $qs['available_on_before'] = $query->getParameter('available_on_before')-
>getTimestamp();
        }
```

```
        if ($query->hasParameter('available_on_after')) {
            $qs['available_on_after'] = $query->getParameter('available_on_after')->getTimestamp();
        }

        if ($query->hasParameter('created_before')) {
            $qs['created_before'] = $query->getParameter('created_before')->getTimestamp();
        }

        if ($query->hasParameter('created_after')) {
            $qs['created_after'] = $query->getParameter('created_after')->getTimestamp();
        }

        if ($query->hasParameter('currency')) {
            $qs['currency'] = $query->getParameter('currency');
        }

        if ($query->hasParameter('deposit_account')) {
            $qs['deposit_account'] = implode(',', $query->getParameter('deposit_account'));
        }

        if ($query->hasParameter('location_id')) {
            $qs['location_id'] = implode(',', $query->getParameter('location_id'));
        }

        if ($query->hasParameter('net_amount')) {
            $qs['net_amount'] = $query->getParameter('net_amount');
        }

        if ($query->hasParameter('status')) {
            $qs['status'] = $query->getParameter('status');
        }

        $orderBy = $query->getParameter('sort') ?? 'created';
        $order = $query->getParameter('order') ?? 'desc';
        $negation = $order === 'desc' ? '-' : '';
        $qs['order_by'] = "{$negation}{$orderBy}";

        return $qs;
    }
}
```