/********** START OF FILE **********/

<?php

```
namespace App;

use Illuminate\Database\Eloquent\Model;

class Note extends Model
{
    /**
     * The attributes that are mass assignable.
     *
     * @var array
     */
    protected $fillable = [
    ];

    /**
     * The attributes that should be hidden for arrays.
     *
     * @var array
     */
    protected $hidden = [
    ];

    /**
     * Get the Note's Family
     * @return \Illuminate\Database\Eloquent\Relations\BelongsTo
     */
    public function family()
    {
        return $this->belongsTo('App\Family');
    }

    /**
     * Get the Note's author (a CentreUser)
     * @return \Illuminate\Database\Eloquent\Relations\BelongsTo
     */
    public function user()
    {
        return $this->belongsTo('App\CentreUser');
    }

}
```

/********** END OF FILE **********/

```
/********** START OF FILE **********/

<?php

namespace App;

use Illuminate\Database\Eloquent\Relations\belongsToMany;
use Illuminate\Database\Eloquent\Relations\HasMany;
use Illuminate\Notifications\Notifiable;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Support\Collection;
use App\Notifications\StorePasswordResetNotification;

class CentreUser extends Authenticatable
{
    use Notifiable;

    protected $guard = 'store';

    /**
     * The attributes that are mass assignable.
     *
     * @var array
     */
    protected $fillable = [
        'name', 'email', 'password', 'role', 'downloader',
    ];

    /**
     * Calculated attributes
     *
     * @var array
     */
    protected $appends = [
        'homeCentre'
    ];

    /**
     * The attributes that should be hidden for arrays.
     *
     * @var array
     */
    protected $hidden = [
        'password', 'remember_token',
    ];

    /**
     * The attributes that should be cast.
     *
     * @var array
     */
    protected $casts = [
        'downloader' => 'boolean',
    ];
```

```
    /**
     * Get the Notes that belong to this CentreUser
     *
     * @return HasMany
     */
    //Because of merge and refactoring User to CentreUser, FK has to be explicitly
stated here
    public function notes()
    {
        return $this->hasMany('App\Note', 'user_id');
    }

    /**
     * Get the CentreUser's Current Centre
     *
     * @return Centre
     */
    public function getCentreAttribute()
    {
        // Check the session for a variable.
        $currentCentreId = session('CentreUserCurrentCentreId');

        // check it's a number
        if (is_numeric($currentCentreId)) {
            // check the centre is in our set
            /** @var Centre $currentCentre */
            $currentCentre = $this->centres()->where('id', $currentCentreId)-
>first();
            if ($currentCentre) {
                return $currentCentre;
            }
        }

        // return default homeCentre if broken.
        /** @var Centre $currentCentre */
        $currentCentre = $this->homeCentre;
        return $currentCentre;
    }

    /**
     * Get the centres assigned to a user
     *
     * @return belongsToMany
     */
    public function centres()
    {
        return $this->belongsToMany('App\Centre');
    }

    /**
     * Gets the first homeCentre, makes it an attribute.
     *
     * @return Centre
     */
    public function getHomeCentreAttribute()
    {
        return $this->homeCentres()->first();
    }
```

```
/**
 * Get the home centres for this user
 * Alas, we lack a belongsToThrough method to this is a collections.
 *
 * @return belongsToMany
 */
protected function homeCentres()
{
    return $this->belongsToMany('App\Centre')->wherePivot('homeCentre', true);
}

/**
 * Get the relevant centres for this CentreUser, accounting for it's role
 *
 * @return Collection
 */
public function relevantCentres()
{
    // default to empty collection
    $centres = collect([]);
    switch ($this->role) {
        case "foodmatters_user":
            // Just get all centres
            $centres = collect(Centre::get()->all());
            break;
        case "centre_user":
            // If we have one, get our centre's neighbours
            /** @var Centre $centre */
            $centre = $this->centre;
            if (!is_null($centre)) {
                $centres = collect($centre->neighbours()->get()->all());
            }
            break;
    }
    return $centres;
}

/**
 * Is a given centre relevant to this CentreUser?
 *
 * @param Centre $centre
 * @return bool
 */
public function isRelevantCentre(Centre $centre)
{
    return $this->relevantCentres()->contains('id', $centre->id);
}

/**
 * Send the password reset notification.
 *
 * @param  string  $token
 * @return void
 */
public function sendPasswordResetNotification($token)
{
    $this->notify(new StorePasswordResetNotification($token, $this->name));
}}/********** END OF FILE **********/
```

```
/********** START OF FILE **********/

<?php


namespace App;

use App\Services\VoucherEvaluator\AbstractEvaluator;
use App\Services\VoucherEvaluator\IEvaluee;
use App\Traits\Evaluable;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsTo;
use Illuminate\Database\Eloquent\Relations\HasMany;
use Log;

class Family extends Model implements IEvaluee
{
    use Evaluable;
    /**
     * The attributes that are mass assignable.
     *
     * @var array
     */
    protected $fillable = [
        'leaving_on',
        'leaving_reason',
        'centre_sequence',
    ];

    /**
     * The attributes that are cast as dates.
     *
     * @var array
     */
    protected $dates = [
        'leaving_on',
    ];

    /**
     * The attributes that should be hidden for arrays.
     *
     * @var array
     */
    protected $hidden = [
    ];

    /**
     * Attributes to autocalculate and add when we ask.
     *
     * @var array
     */
    protected $appends = [
        'expecting',
        'rvid'
    ];
```

```
    /**
     * Gets the evaluator from up the chain.
     *
     * @return AbstractEvaluator
     */
    public function getEvaluator()
    {
        return $this->registrations()->first()->evaluator;
    }

    /**
     * Gets the due date or Null;
     *
     * @return mixed
     */
    public function getExpectingAttribute()
    {
        $due = null;
        foreach ($this->children as $child) {
            if (!$child->born) {
                $due = $child->dob;
            }
        }
        return $due;
    }

    /**
     * Generates and sets the components required for an RVID.
     *
     * @param Centre $centre
     * @param bool $switch Force the user to switch centre and change RVID.
     */
    public function lockToCentre(Centre $centre, $switch = false)
    {
        // Check we don't have one.
        if (!$this->centre_sequence || $switch) {
            if ($centre) {
                // Get the centre's next sequence.
                $this->centre_sequence = $centre->nextCentreSequence();
                // set the sequence
                $this->initialCentre()->associate($centre);
            } else {
                Log::info('Failed to generate RVID: No Centre given.');
            }
        } else {
            Log::info('Failed to generate RVID: ' . $this->rvid . ' already
exists.');
        }
    }
```

```
    /**
     * Calculate the 'rvid' attribute and return it.
     *
     * @return string
     */
    public function getRvidAttribute()
    {
        $rvid = "UNKNOWN";
        if ($this->initialCentre && $this->centre_sequence) {
            $rvid =  $this->initialCentre->prefix . str_pad((string)$this-
>centre_sequence, 4, "0", STR_PAD_LEFT);
        }
        return $rvid;
    }

    /**
     * Get the Family's designated Carers
     * There should always be ONE of these!
     *
     * @return HasMany
     */
    public function carers()
    {
        return $this->hasMany('App\Carer');
    }

    /**
     * Get the Family's Children
     * @return HasMany
     */
    public function children()
    {
        return $this->hasMany('App\Child');
    }

    /**
     * Get Notes about this Family
     *
     * @return HasMany
     */
    public function notes()
    {
        return $this->hasMany('App\Note');
    }

    /**
     * Get the Registrations with Centres for this Family
     *
     * @return HasMany
     */
    public function registrations()
    {
        return $this->hasMany('App\Registration');
    }
```

```
    /**
     * Get the Family's intial registered Centre.
     * @return BelongsTo
     */
    public function initialCentre()
    {
        return $this->belongsTo('App\Centre', 'initial_centre_id');
    }

    public function scopeWithPrimaryCarer($query)
    {
        $subQuery = \DB::table('carers')
            ->select('name')
            ->whereRaw('family_id = families.id')
            ->orderBy('id', 'asc')
            ->limit(1);

        return $query->select('families.*')->selectSub($subQuery, 'pri_carer');
    }
}

/********** END OF FILE **********/
```

```
/********** START OF FILE **********/

<?php


use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateNotesTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('notes', function (Blueprint $table) {
            $table->increments('id');
            $table->text('content'); // 64k
            $table->integer('family_id')->unsigned(); // FK Families
            $table->integer('user_id')->unsigned(); // FK Centre Users
            $table->timestamps();

            $table->foreign('family_id')
                ->references('id')
                ->on('families');

            $table->foreign('user_id')
                ->references('id')
                ->on('centre_users');
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('notes');
    }
}

/********** END OF FILE **********/
```

```
/********** START OF FILE **********/

<?php

namespace Tests\Unit\Models;

use App\Centre;
use App\CentreUser;
use App\Note;
use Tests\TestCase;
use Illuminate\Foundation\Testing\DatabaseMigrations;

class CentreUserModelTest extends TestCase
{
    use DatabaseMigrations;

    protected $centreUser;
    protected $notes;
    protected function setUp()
    {
        parent::setUp();
        $this->centreUser = factory(CentreUser::class)->create()->fresh();
        $this->notes = factory(Note::class, 2)->create(['user_id' => $this-
>centreUser->id]);
    }

    /** @test */
    public function testCentreUserHasExpectedAttributes()
    {
        $cu = $this->centreUser;
        $this->assertNotNull($cu->name);
        $this->assertNotNull($cu->email);
        $this->assertContains($cu->role, ['centre_user', 'foodmatters_user']);
        // Default false
        $this->assertFalse($cu->downloader);
    }

    /** @test */
    public function testCentreUserCanHaveNotes()
    {
        $this->assertCount(2, $this->centreUser->notes);
    }

    /**@test */
    public function testCentreUserCanHaveDownloadTrue()
    {
        // Standard CU
        $cu = $this->centreUser;
        $this->assertFalse($cu->downloader);

        // Change their settings
        $cu->downloader = true;
        $cu->fresh();
        $this->assertTrue($cu->downloader);
        $cu = factory(CentreUser::class, 'withDownloader')->create()->fresh();
        $this->assertTrue($cu->downloader);
    }
```

```
    /** @test */
    public function testCentreUserCanHaveAHomeCentre()
    {
        $cu = $this->centreUser;
        // Has no centres;
        $this->assertEmpty($cu->centres);
        $this->assertNull($cu->homeCentre);

        // Make one, set it to Home
        $centre = factory(Centre::class)->create();
        $cu->centres()->attach($centre->id, ['homeCentre' => true]);

        // There is one
        $this->assertEquals(1, $cu->centres()->count());
        // It is the homeCentre
        $this->assertEquals($centre->id, $cu->homeCentre->id);
    }

    /** @test */
    public function testCentreUserCanHaveAlternativeCentres()
    {
        $cu = $this->centreUser;
        // Has no centres;
        $this->assertEmpty($cu->centres);

        // Make some
        $centres = factory(Centre::class, 4)->create();
        $cu->centres()->attach($centres->pluck('id')->all());

        // There is 4
        $this->assertEquals(4, $cu->centres()->count());

        // But We have no homeCentre
        $this->assertEmpty($cu->homeCentre);
    }
}

/********** END OF FILE **********/
```