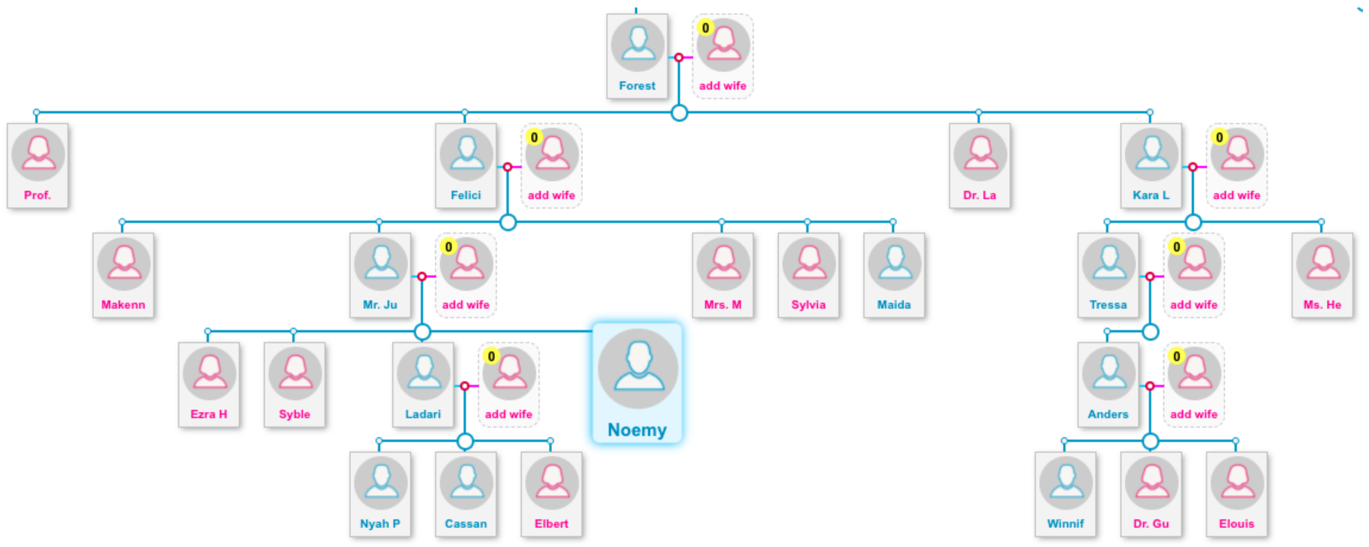

Building Family Tree

packagist v1.0.1 tests repo, branch, or workflow not found code style repo, branch, or workflow not found downloads 2

Content

Introduction

girover/laravel-family-tree is a package that allows you to build family trees. With this package it will be very simple to create trees and add nodes to trees.

Every tree is allowed to have 85 generations. Assuming each generation has 20 years, this means that 1700 years of data can be stored in one tree.
Every node in a tree is allowed to have 676 direct children.

prerequisites

- Laravel 8+
- PHP 8+
- Mysql 5.7+

Installation

You can add the package via **composer**:

```
composer require girover/laravel-family-tree
```

Before installing the package you should configure your database.

Then you can install the package by running Artisan command

```
php artisan tree:install
```

this command will take care of the following tasks:

- Publishing Config file `config\tree.php` to the config folder of your Laravel application.
- Publishing migration files to folder `Database\migrations` in your application.
- Migrate the published migrations.
- Publishing Assets **(css, js, images)** to the public folder in your application and they will be placed in `public\vendor\tree`.
- Publishing JSON Translation files to the `resources\lang\vendor\tree`

Assets

After publishing assets (css, js, images), they will be placed in `public` folder of the project in a folder called `vendor/tree`. You are free to move any of these assets to other directories.

You should add the CSS file `vendor/tree/css/tree.css` to your blade file to get the tree styled.

Images

Every node in tree has an avatar photo. Male and Female Icons by default will be stored in the public folder under
`vendor/tree/images`.
however you are free to choose another folder for the images of nodes, but you must provide it in the `config\tree.php` file.

```
'photos_folder' => 'path/to/your/images',
```

So your images folder should be in the **`public`** folder. Example: if images are stored in folder called `images/avatars` the configs must be:

```
'photos_folder' => 'images/avatars',
```

Note: When saving photos in storage folder, it is important to create symbolic link to the photos folder.

Usage

To start building family trees, you must have two models. The first one represents trees in database and it must use the trait:
`\Girover\Tree\Traits\Treeable`
The second model represents nodes in these trees, and it must use trait:
`\Girover\Tree\Traits\nodeable`
NOTE: The names of the models that represents trees and nodes in database must be provided in `config/tree.php`.

```
// config/tree.php
return [
    /*
    |------------------------------------------------------------
    | Model That uses trait \Girover\Tree\Traits\Treeable
    |------------------------------------------------------------
    |
    | example: App\Models\Family::class
    */
    'treeable_model' => App\Models\Family::class,
    /*
    |------------------------------------------------------------
    | Model That uses trait \Girover\Tree\Traits\Nodeable
    |------------------------------------------------------------
    |
    | example: App\Models\Person::class
    */
    'nodeable_model' => App\Models\Person::class,
    .
```

```
        .
        .
```

Tree

To start building a tree or creating a new tree, it is very simple and thanks to Eloquent models from Laravel. The model that represents trees in database should use **trait**: `\Girover\Tree\Traits\Treeable` For example if your model is called `Tree` so it should looks like this:

```
namespace App\Models;

use Girover\Tree\Traits\Treeable;

class Tree extends Model
{
    use Treeable;
}
```

And now you can use your model to deal with trees.

```
use App\Models\Tree;

$tree = Tree::create(
    [
        'info' => 'info',
        'another_info' => 'another info',
    ]
);
```

After creating the tree, you can start to add as many nodes as you want.
Let's start adding the First node, the **`Root`**, to the tree.

```
    $data = ['name'=>'root', 'birth_date'=>'2000-01-01'];

    $tree->createRoot($data);
```

What if you want to make a new Root?
In this case you can use the method **`newRoot`**, and the previously created Root will become a child of the new created Root.

```
    $new_root_data = ['name'=>'new_root', 'birth_date'=>'2001-01-01'];

    $tree->newRoot($new_root_data);
```

After creating the Root in the tree, let's add first child for the Root.

```
    use Girover\Tree\Models\Tree;

    $tree = Tree::find(1);

    $first_child_data = ['name'=>'first child', 'birth_date'=>'2001-01-
01'];

    $tree->pointerToRoot()->newChild($first_child_data ,'m'); // m = male

    // Or you can do this instead
    $tree->pointerToRoot()->newSon($first_child_data);
```

Now our tree consists of two nodes, the root and the first child of the root.

You can call the following method on an object of Tree

#	function	Description	Params
1	`createRoot($data)`	create root in the tree.	`$data` is array of root info
2	`newRoot($data)`	makes new root for the tree.	`$data` is array of info for the new root
3	`toHtml()`	convert the tree to html to view it	
4	`draw()`	convert the tree to html to view it	
5	`toTree()`	convert the tree to html to view it	
6	`emptyTree()`	return an empty tree to view it	
7	`pointer()`	To get the pointer inside the tree	
8	`pointerToRoot()`	To move the pointer to indicate to the root	
9	`pointerTo($nodeable)`	To move the pointer to the given node	
10	`goTo($nodeable)`	To move the pointer to the given node	
14	`countGenerations()`	To get how many generations this tree has	
15	`nodesOnTop()`	Get the newest generation members in the tree	
16	`mainNode()`	Get the main node in the tree	

Pointer

A tree has a **Pointer** inside it, and this **Pointer** points to one node.
Pointer can move through all nodes in the tree.
Because the Pointer points to a **node** inside the tree, so it can call all methods of model that uses Nodeable Trait .
To get the pointer you can do the following:

```
use App\Models\Tree;

$tree    = Tree::find(1);
$pointer = $tree->pointer();
```

And now you can use the pointer to make a lot of actions inside the tree, for example moving through nodes, deleting and retrieving more information about nodes.
Eg. To move the pointer to specific node:

```
use App\Models\Tree;
use App\Models\Node;

$tree    = Tree::find(1);
$node    = Node::find(10);

$tree->pointer()->to($node);
```

And now you can get the node data by calling the method `node`

```
$node = $pointer->node();
echo $node->attribute_1;
echo $node->attribute_2;
echo $node->attribute_3;
```

Note that we called method `node` after we had called the method `to($node)`.
This is because when a tree object is created, its **Pointer** indicates to `null`.

Node

- What is a node
- Retrieving nodes
- Adding nodes
- Updating nodes
- Deleting nodes
- Checking nodes
- Relationships
- Relocating nodes

What is a node

Node is a **person** in a tree and every node in the tree is connected with another one by using **Location mechanism**.

To retrieve a nodes you can use Eloquent model that uses trait `Girover\Tree\Traits\Nodeable`

```
use App\Models\Node;

$node    = Node::find(1);
```

Retrieving nodes

To get the tree that the node belongs to.

```
$node->getTree();
```

To get the father of the node:

```
return $node->father();
```

To get the grandfather of the node:

```
return $node->grandfather();
```

To get the ancestor that matches the given number as parameter.

```
$node->ancestor(); // returns father
$node->ancestor(2); // returns grandfather
$node->ancestor(3); // returns the father of grandfather
```

To get all ancestors of a node

```
return $node->ancestors();
```

To get all uncles of the node:

```
return $node->uncles();
```

To get all aunts of the node:

```
return $node->aunts();
```

To get all children of the node:

```
return $node->children();
```

To get all sons of the node:

```
return $node->sons();
```

To get all daughters of the node:

```
return $node->daughters();
```

To get all descendants of the node:

```
return $node->descendants();
```

To get all male descendants of the node:

```
return $node->maleDescendants();
```

To get all female descendants of the node:

```
return $node->femaleDescendants();
```

To get the first child of the node:

```
return $node->firstChild();
```

To get the last child of the node:

```
return $node->lastChild();
```

To get all siblings of the node:

```
return $node->siblings();
```

To get all brothers of the node:

```
return $node->brothers();
```

To get all sisters of the node:

```
return $node->sisters();
```

To get the next sibling of the node. gets only one sibling.

```
return $node->nextSibling();
```

To get all the next siblings of the node. siblings who are younger.

```
return $node->nextSiblings();
```

To get the next brother of the node. gets only one brother.

```
return $node->nextBrother();
```

To get all the next brothers of the node. brothers who are younger.

```
return $node->nextBrothers();
```

To get the next sister of the node. gets only one sister.

```
return $node->nextSister();
```

To get all the next sisters of the node. sisters who are younger.

```
return $node->nextSisters();
```

To get the previous sibling of the node. only one sibling.

```
return $node->prevSibling();
```

To get all the previous siblings of the node. siblings who are older.

```
return $node->prevSiblings();
```

To get the previous brother of the node. only one brother.

```
return $node->prevBrother();
```

To get all the previous brothers of the node. brothers who are older.

```
return $node->prevBrothers();
```

To get the previous sister of the node. only one sister.

```
return $node->prevSister();
```

To get all the previous sisters of the node. sisters who are older.

```
return $node->prevSisters();
```

To get the first sibling of the node.

```
return $node->firstSibling();
```

To get the last sibling of the node.

```
return $node->lastSibling();
```

To get the first brother of the node.

```
return $node->firstBrother();
```

To get the last brother of the node.

```
return $node->lastBrother();
```

To get the first sister of the node.

```
return $node->firstSister();
```

To get the last sister of the node.

```
return $node->lastSister();
```

Adding nodes

The next code will create father for a node, only if this node is a Root in the tree.
If the node is not a `root`, `Girover\Tree\Exceptions\TreeException` will be thrown.

```
$data = ['name' => $name, 'birth_date' => $birth_date];
$node->createFather($data);
// OR
$tree = Tree::find(1);
$tree->pointerToRoot()->createFather($data);
```

To create new sibling for the node:

```
$data = ['name'=>$name, 'birth_date'=>$birth_date];

return $node->newSibling($data, 'm'); // m = male
```

To create new brother for the node:

```
    $data = ['name'=>$name, 'birth_date'=>$birth_date];
    $node->newBrother($data);

    // or you can use the newSibling method
    $node->newSibling($data, 'm');
```

To create new sister for the node:

```
    $data = ['name'=>$name, 'birth_date'=>$birth_date];
    $node->newSister($data);

    // or you can use the newSibling method
    $node->newSibling($data, 'f');
```

To create new son for the node:

```
    $data = ['name'=>$name, 'birth_date'=>$birth_date];
    $node->newSon($data);

    // or you can use the newChild method
    $node->newChild($data, 'm');
```

To create new daughter for the node:

```
    $data = ['name'=>$name, 'birth_date'=>$birth_date];
    $node->newDaughter($data);

    // or you can use the newChild method
    $node->newChild($data, 'f');
```

To make the node as the main node in its tree.

```
    $node->makeAsMainNode();
```

Deleting nodes

To delete a node.

```
    use App\Models\Person;

    $node = Person::find(10)->delete();
```

note: The node will be deleted with all its descendants.
But if you want just to delete the node and not its descendants,
then you can move its descendants before deleting the node.

```
use App\Models\Person;

$node = Person::find(10);
$another_node = Person::find(30);

$node->moveChildrenTo($another_node);
$node->delete();
```

Or you can directly move children to the father of deleted node by using `onDeleteMoveChildren` method without passing any node as parameter.

```
use App\Models\Person;

$node = Person::find(10);

$node->onDeleteMoveChildren()->delete();
```

To delete all children of a node:

```
use App\Models\Person;

$node = Person::find(10);

return $node->deleteChildren(); // number of deleted nodes
```

Checking nodes

To determine if the node is root in the tree

```
return $node->isRoot(); // returns true or false
```

To determine if the node is ancestor of another node:

```
use App\Models\Person;

$node = Person::find(1);
$another_node = Person::find(2);
```

```
return $node->isAncestorOf($another_node); // returns true OR false
```

To determine if the node is father of another node:

```
use App\Models\Person;

$node = Person::find(1);
$another_node = Person::find(2);

return $node->isFatherOf($another_node); // returns true OR false
```

Determine if the node has children

```
return $node->hasChildren(); // returns true or false
```

To determine if the node is child of another node:

```
use App\Models\Person;

$node = Person::find(1);
$another_node = Person::find(2);

return $node->isChildOf($another_node); // returns true OR false
```

Determine if the node has siblings

```
return $node->hasSiblings(); // returns true or false
```

To determine if the node is sibling of another node:

```
use App\Models\Person;

$node = Person::find(1);
$another_node = Person::find(2);

return $node->isSiblingOf($another_node); // returns true OR false
```

To count the children of the node:

```
    return $node->countChildren();
```

To count sons of a node:

```
    return $node->countSons();
```

To count daughters of the node:

```
    return $node->countDaughters();
```

To count all siblings of the node:

```
    return $node->countSiblings();
```

To count all brothers of the node:

```
    return $node->countBrothers();
```

To count all sisters of the node:

```
    return $node->countSisters();
```

To count all descendants of the node:

```
    return $node->countDescendants();
```

To count all male descendants of the node:

```
    return $node->countMaleDescendants();
```

To count all female descendants of the node:

```
    return $node->countFemaleDescendants();
```

Relationships

To get all wives of the node:

```
    $node->wives;
    // to add constraints
    $node->wives()->where('name', $name)->get();
```

When trying to get wives of female node a `Girover\Tree\Exceptions\TreeException` will be thrown.
Note that this will get divorced wives too.
To get only wives who are not divorced you can do this:

```
    $node->wives()->ignoreDivorced()->get();
```

To get husbands of the node:

```
    $node->husband;
    // to add constraints
    $node->husband()->where('name', $name)->get();
```

When trying to get husband of male node a `Girover\Tree\Exceptions\TreeException` will be thrown.
Note that this will get divorced husbands too.
To get only husbands who are not divorced you can do this:

```
    $node->husband()->ignoreDivorced()->get();
```

To assign wife to a node:

```
    $wife = Node::find($female_node_id)
    $data = ['date_of_marriage'=>'2000/01/01',
'marriage_desc'=>'description'];

    return $node->getMarriedWith($wife, $data);
```

When trying to do this with a female node (woman) a `Girover\Tree\Exceptions\TreeException` will be thrown. so if `$node` is a woman Exception will be thrown.

To divorce a wife

```
    $husband = Node::find($male_node_id)
    $wife    = Node::find($female_node_id)
```

```
return $husband->divorce($wife);
```

When trying to do this with a female node a `Girover\Tree\Exceptions\TreeException` will be thrown. so if `$husband` is a woman Exception will be thrown.

To set uploaded photo to the node:

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\Node;

class PersonController extends Controller
{
    public function addPhoto(Request $request)
    {
        $person     = Node::find($request->person_id);
        $photo      = $request->file('photo');

        $person->setPhoto($photo, 'new name');

        return view('persons.index')->with('success', 'photo was 
added');
    }
}
```

Relocating nodes

to move an existing node to be child of another node.
Note this will move the node with its children.

```
use \Girover\Tree\Models\Node;

$node = Node::find(10);
$another_node = Node::find(30);

$node->moveTo($another_node);
```

To move children of a node to be children of another node
you can do this:

```
use \Girover\Tree\Models\Node;
```

```
    $node = Node::find(10);
    $another_node = Node::find(30);

    $node->moveChildrenTo($another_node);

    // OR
    $location = 'aa.fd';

    $node->moveChildrenTo($location);
```

Note: When trying to move node1 to node2, if node2 is descendant of node1 `TreeException` will be thrown. The same exception will be thrown if node2 is a female node.

To move a node after another node you can do this:

```
    use \Girover\Tree\Models\Node;

    $node = Node::find(10);
    $another_node = Node::find(30);

    $node->moveAfter($another_node);
```

To move a node before another node you can do this:

```
    use \Girover\Tree\Models\Node;

    $node = Node::find(10);
    $another_node = Node::find(30);

    $node->moveBefore($another_node);
```

Note: When trying to move the Root or when trying to move a node after or before one of its descendants `TreeException` will be thrown.

to get the generation number of a node in a tree:

```
    return $node->generation(); // int or null
```

Note: You can use the **Pointer** of tree to access all methods of class Node.
for example:

```
    use Girover\Tree\Models\Tree;

    $tree = Tree::find(1);
    $tree->pointer()->to('aa.aa')->father();       // move Pointer to
location 'aa.aa' and then get its father.
    $tree->pointer()->grandfather();       // get grandfather of node that
Pointer indicates to
    $tree->pointer()->ancestor(3);            // get father of node that
Pointer indicates to
    $tree->pointer()->children();          // get children of node that
Pointer indicates to
    $tree->pointer()->sons();              // get sons of node that
Pointer indicates to
    $tree->pointer()->newDaughter($data);  // create daughter for the node
that Pointer indicates to
    $tree->pointer()->newSon($data);  // create son of node that Pointer
indicates to
    $tree->pointer()->newSister($data);  // create sister for the node
that Pointer indicates to
    $tree->pointer()->newChild($data, 'm');  // create son for the node
that Pointer indicates to
    $tree->pointer()->firstChild();  // get the first child of node that
Pointer indicates to
    .
    .
    .
    .
    .
    $tree->pointer()->toHtml();
```

Attaching and Detaching

When creating nodeable models by using methods `create` or `save`, the created model will not be attached with any node in any tree.

```
    use App\Models\Person;

    $adam = Person::create(['name'=>'Adam', 'birth_date'=>'0000-00-00']);

    $eva = new Person;
    $eva->name = 'Eva';
    $eva->birth_date = '0000-00-00';
    $eva->save();
```

Both Adam and Eva will be created in database table, but they will not be a node yet in any tree.
To make Adam as a node in a tree you can use all nodeable methods like:

```
    use App\Models\Person;

    $adam = Person::where('name', 'adam')->first();
    $person = Person::find(1);

    $person->newSon($adam);
    //or
    $person->createFather($adam); // only if $person is a root in the tree
    //or
    $person->newBrother($adam); // only if $person is not a root in the
tree
    //or
    .
    .
    .
```

detaching

When deleting nodeable models they will be deleted from database table, but what if you need to delete(detach) them from a tree while keeping them in database.
Let's look at the following code:

```
    use App\Models\Person;

    $person = Person::find(100);
    $person->delete();
```

In the code above, the person will be deleted from database with all its descendants.

But we need only to delete it from a tree not from database. Look at this code:

```
    use App\Models\Person;

    $person = Person::find(100);
    $person->detachFromTree();
```

Now the person still exists in database, but it is not a node in any tree anymore.

Helpers

To get all detached nodeable models, you can use the helper function `detachedNodeables`.
This will return instance of `\Illuminate\Database\Eloquent\collection` with all detached nodeable models.

```
    <?php
    $detached = detachedNodeables();
```

To get the count of detached nodeable models, use the helper function `countDetachedNodeables`.

```php
<?php

$count_detached = countDetachedNodeables();
```

Rendering trees

To show a tree in the browser you can use one of these methods `toHtml`, `toTree` or `draw` on an object of the Treeable model eg. `App\Models\Tree`

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\Tree;

class TreeController extends Controller
{
    public function index()
    {
        $tree     = Tree::find(1);

        $treeHTML = $tree->toHtml()
        // OR
        $treeHTML = $tree->toTree()
        // OR
        $treeHTML = $tree->Draw()

        return view('tree.index')->with('treeHTML', $treeHTML);
    }
}
```

And now inside your blade file you can render the tree by using custom blade directive `@tree($treeHtml)` or by using this syntax `{!! toHtml !!}`.

```
<!-- views/tree/index.blade.php -->
<div>
    @tree($treeHtml)
</div>
<!-- OR -->
<div>
```

```
    {!! $treeHTML !!}
</div>
```

Note the above example will render the entire tree, but if you want to render a subtree starting from a specific node you can use the **Pinter** to do that:

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\Tree;
use App\Models\Node;

class TreeController extends Controller
{
    public function index()
    {
        $tree     = Tree::find(1);

        $person     = Node::find(11);

        $treeHTML = $tree->pointer()->to($person)->toHtml();
        // OR
        $treeHTML = $tree->pointer()->to($person)->toTree();
        // OR
        $treeHTML = $tree->pointer()->to($person)->draw();

        return view('tree.index')->with('treeHTML', $treeHTML);
    }
}
```

You can also use a node model to do the same by using one of these methods `toHtml`, `toTree` or `draw` on an object of the nodeable model eg. `App\Models\Node`:

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\Node;

class TreeController extends Controller
{
    public function index()
    {
        $person   = Node::find(11);

        $treeHTML = $person->toHtml();
```

```
            // OR
            $treeHTML = $person->toTree();
            // OR
            $treeHTML = $person->draw();

            return view('tree.index')->with('treeHTML', $treeHTML);
        }
    }
```

Customizing node style

If you want to add some css classes to the nodes that have specific role, then you can bind a custom function called `nodeCssClass` to the **Service Container**. This function must return a callable function which accept one argument represents nodeable model. And this callable function must return css classes names as string. Let's take an example.
If your nodeable models have an attribute called `is_died` which has values `true` or `false`, then you want to add a css class called `is-died` to the nodes that have this value as `true` to give them more style.

```
    // AppServiceProvider
    public function boot()
    {
        $this->app->singleton('nodeCssClasses',function($app){
            return function($nodeable){
                return ($nodeable->is_died) ? 'is-died' :'';
            };
        });
    }
```

So every nodeable model that has **true** as **is_died** will get css class called **is-died**.
Now you can write your won css to style classes **is-died** as you like.

Testing

```
./vendor/bin/PHPUnit
```

Changelog

Please see CHANGELOG for more information on what has changed recently.

Contributing

Please see CONTRIBUTING for details.

Security Vulnerabilities

Please review our security policy on how to report security vulnerabilities.

Credits

- Majed Girover
- All Contributors

License