# Вовед

## Креирање на нов проект

## Структура на Laravel

Laravel
├── app
│   ├── Http
│   │   ├── **Controllers**
│   │   ├── **Middleware**
│   ├── Providers
│   ├── **Models**
│   ├── ...
├── bootstrap
├── config
├── database
│   ├── **migrations**
│   ├── seeders
│   ├── factories
│   ├── ...
├── **public**
├── resources
│   ├── views
│   ├── lang
│   ├── ...
├── routes
│   ├── **web.php**
│   ├── api.php
│   ├── ...
├── **storage**
│   ├── app
│   ├── framework
│   ├── logs
│   ├── ...
├── tests
├── vendor
├── **.env**
├── **.env.example**
├── .gitignore
├── composer.json
├── ...

# Model

# Migrations (Миграции)

## Создавање на нова миграција

За да креирате нова миграцијата ја користите следнава команда:

```
php artisan make:migration create_name_table
```

Фајлот ќе биде зачуван во `/app/database/migrations/`.

## Структура на миграцијата

Миграциите се состојат од две главни методи `up` и `down`.

Доколку сакаме да додадеме нов атрибут во табелата тоа го правиме во `public function up` така што го користиме параметарот $table->тип('име на атрибутот').

```
return new class extends Migration
{
    public function up(): void
    {
        Schema::create('post_votes', function (Blueprint $table) {
            $table->id();
            $table->timestamps();
        });
    }
    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('post_votes');
    }
};
```

Постојат многу податочни типови кои може да ги користиме при декларирањето на атрибути, затоа ќе ги наброиме само најбитните:

- **Примарен клуч**: под defult е `$table->id()` но доколку сакаме да го променимнуваме во друго име, тогаш користиме `$table->primary('name')`.

    - Доколку сакаме да направиме композитен примарен клуч, тогаш користиме: `$table->primary(['primary1','primary2'],..)`
- **Надворешен клуч**: `$table->foreign('name')->references('attribute')->on('tableName');`
    - **Cascade** и **restrict**:
        - `->cascadeOnUpdate();` - Направи cascade при update.
        - `->restrictOnUpdate();` - Не дозволувај update.
        - `->cascadeOnDelete();` - Направи cascade при delete.
        - `->restrictOnDelete();` - Не дозволувај бришење.
        - `->nullOnDelete();` - При бришење, потолни ги подаотците со `null`.
- **Unique**: За да создадеме атрибут со единствени вредности користиме `->unique()`, на пример секој студент има единствен матичен број или индекс: `$table->string('index')->unique();`.
- **Defult**: Доколку сакаме да иницијалираме некоја вредност при креирање на нов запис користиме: `default(вредност)`. На пример `$table->boolean('online')->default(true)`.
- **Null**: Доколку сакаме некоја вредност да не ја дефинираме при создавањето на нов запис, тогаш користиме само `->nullable()`. На пример `$table->string('email')->nullable();`
- **Цел број**: `$table->integer('name');`
- **Булеан**: `$table->boolean('name');`
- **Текст**: `$table->string('name');` или `$table->text('name');`
- **Децимален број**: `$table->float('name', 8, 2);`

За да ја зачуваме миграцијата во базата користиме само `php artisan migrate` оваа команда ќе ја создаде новата табела со дефинираните атрибути. Кога ќе креираме нова миграција таа веднаш преоѓа во состобја `pending` па затоа оваа команда ќе ја зачува табелата. Но итсто така постојат и други команди кои треба да ги знаете:

| Команда | Опис |
| --- | --- |
| `php artisan migrate:status` | Ќе ги прикаже состојбите на сите миграции. |
| `php artisan migrate` | Ќе ја изврши `up` методата на секоја миграција која е во состојба `pending`. |
| `php artisan migrate:refresh` | Доколку направиме некаква промена на миграцијата и сакаме да се вратиме на иницијалната состојба, тогаш го користиме ова. |
| `php artisan migrate:fresh` | Ќе ги избрише сите табели и повторно ќе ги создаде. |
| `php artisan migrate:rollback` | Ќе ги негира последните миграцции кои сме ги направиле. |

Laravel користи obejct realtional mapper (ORM) кој ни дозволува работа со базата но преку php.

За таа цел креираме модели кои ги поврзуавме со соодветните шеми кои ги дефиниравме во претходниот чекор.

Во следните чекори ќе објасниме како се поврзуваат миграциите и моделите и како може да работиме со нив.

# Создавање на нов модел

За да создадеме нов модел ја користиме командата: `php artisian make:model име`, на пример :

```
php artisan make:model Comment
php artisan make:model Post
```

Сите модели се сместени во `/app/Models`.

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Post extends Model
{
    // protected $table = 'posts';
}
```

---

**Како Laravel знае како да ги поврзе моделите и табелите во базата?**

Laravel пробува да ги спои табелите според имињата (доколку имаат слично име или заеднички зборови), но понекогаш имињата на моделите и табелите не ни се исти, па затоа најдобро е да ги поврземе сами со помош на командата: `protected $table = 'име на таблеата';`.

На пример доколку имаме некоја миграција со: `Schema::create('poinakvo_ime'` тогаш во моделот ќе може да ја споиме
споиме со помош на: `protected $table = 'poinakvo_ime';`.

---

# Eloquent: Relationships

Во миграциите ние дефиниравме надворешни клучеви од една табела кон друга.

Laravel ни нуди поедноставен и поефикасен начин за работа со тие врски.

Битно е да знеате дека:

`belongsTo` и `belongsToMany` се користи кога овој модел ги рефренцира другите модели. `belongsTo` се користи кога имаме `one-to` додека пак `belongsTo` кога имаме `many-to`

`hasOne` или `hasMany` се користи од се користи кога овој модел е референциран од страна на друг модел. Кога имаме `one-to` се користи `hasOne` , кога имаме `many-to` се користи `hasMany`

## one-to-one

На пример еден корисник има еден профил. Profile има надворешен клуч кон Student.

```
class Student extends Model
{
    // името може да биде произволно
    public function profile()
    {
        // студент моделот е рефренциран од страна на Profile па затоа имаме
hasOne
        return $this->hasOne(Profile::class);
    }
}

class Profile extends Model
{
    // името може да биде произволно
    public function stduent()
    {
        // има надворешен клуч кон Student
        return $this->belongsTo(Student::class);
    }
}
```

## one-to-many

На пример една објава има многу коментари.

```
class Post extends Model
```

```
{
    public function comments()
    {
        return $this->hasMany(Comment::class);
    }
}

class Comment extends Model
{
    public function post()
    {
        return $this->belongsTo(Post::class);
    }
}
```

## many-to-many

Еден студент може да има многу предмети и еден предмет може да има многу студенти.

```
class Course extends Model
{
    public function enroll()
    {
        return $this->hasMany(Enroll::class);
    }
}

class Enroll extends Model
{
    public function courses()
    {
        return $this->belongsToMany(Course::class);
    }

    public function students()
    {
        return $this->belongsToMany(Student::class);
    }


}

class Student extends Model
{
    public function enroll()
    {
        return $this->hasMany(Enroll::class);
    }
```

```
}
```

Овој начин ни дава поголема контрола врз кодот, бидејќи имаме модел `Enroll` со кој полесно ќе работиме.

Вториот начин е да ги поврземе `Student` и `Course` директно, овај начин ќе ни генерира нова **пивот** табела во базата со која може да работиме.

```
class Student extends Model
{
    public function courses()
    {
        return $this->belongsToMany(Course::class);
    }
}

class Course extends Model
{
    public function students()
    {
        return $this->belongsToMany(Student::class);
    }
}
```

За да ги споиме студентите и курсевите, ние ќе мора да користиме `attach()` , `detach()` .

## Attach/Detach

Со помош на `detach()` ја остануваме врската за одреден објект, додека пак `attach()` ќе креира нова врска.

Пример:

Да го земеме претходниот пример со студенти и курсеви:

Студенти:

| ID | Име | Презиме |
|---|---|---|
| 1 | Петар | Петровски |
| 2 | Ана | Андоновска |
| 3 | Марко | Марковски |
| 4 | Елена | Георгиева |
| 5 | Иван | Ивановски |

Курсеви:

| ID | Име |
|---|---|
| 1 | Дискретна Математика |
| 2 | Структурно Програмирање |

Доколку сакаме да ги поврземе курсеви и студентите, тоа мора да го направиме преку `attach()`, моментално пивот табелата е празна.

```php
// најди го студентот со ID = 2 (Ана)
$student = Student::find(2);

// најди го курсот со ID = 1 (Структурно Програмирање)
$course = Course::find(1);
// во пивот табелата додади ја оваа врска
$student->courses()->attach($course);

// истито може да се направи и обратно:
// $course->students()->attach($student);
```

Доколку сакаме да додеме низа од курсеви или студенти, го правиме тоа преку нивните ID's

```php
// најди го студентот со ID = 5 (Иван)
$student = Student::find(5);
// низа со IDs на курсевите (Дискретна Математика со ID = 1 и Структурно Програмирање со ID = 2)
$courseId = [1, 2];

// во пивот табелата додади ја оваа врска
$student->courses()->attach($courseIds);
```

По извршување на наредбите, пивот табелата ќе ги има следниве податоци:

| student_id | course_id |
|---|---|
| 2 | 1 |
| 5 | 1 |
| 5 | 2 |

За да избришеме некоја врска користиме `detach()`, работи на истиот начин како и `attach()`.

```php
//најди го курсот со ID = 1 (Структурно Програмирање)
$course = Course::find(1);
// најди го студентот со ID = 5 (Иван)
$student = Student::find(5);

// избриши ја оваа врска од пивот табелата
$course->students()->detach($student); .

// $course->students()->detach([1,5]);
```

По извршување на наредбата, пивот табелата ќе ја избрише дефинираната врска од табелата.

| student_id | course_id |
|---|---|
| 2 | 1 |
| 5 | 2 |

# Работа со модели

Да речеме дека го имаме модел `Student` со атрибути `id`, `име`, `презиме`:

| ID | Име | Презиме |
|---|---|---|

## Create

Во Laravel, можете да креирате нов запис во моделот Student на неколку начини:

Користејќи го `new` клучниот збор и методот `save`:

```php
$student = Student();

$student->име = 'Петар';
$student->презиме = 'Петровски';

$student->save();
```

Користејќи го `create`:

```php
Student::create([
    'име' => 'Ана',
    'презиме' => 'Андоновска'
]);
```

За да може да работи `create` мора во моделот да додадеме `protected $fillable` и да ги наброите сите атрибути кои сакаме да ги користиме при креирање. во случајот користиме само `име` и `презиме`, па затоа:

```php
class Student extends Model
{
    // доколку имате другит атрибути мора да ги додадете tuka за да работи create методот.
    protected $fillable = ['име', 'презиме'];
}
```

Ова не важи за `save`.

| ID | Име | Презиме |
|---|---|---|
| 1 | Петар | Петровски |
| 2 | Ана | Андоновска |

# Update

Во Laravel, можете да ажурирате некој запис на неколку начини:

Со помош на `save`, прво го наоѓаме студентот со помош на методот `find` па потоа правиме некаква промена и истата ја зачувуваме.

```php
// најди го студентот со ID = 1 (Петар)
$student = Student::find(1);
//смени го името и презимето на овој студент
$student->име = 'Марко';
$student->презиме = 'Маркоски';
// зачувај ја промената
$student->save();
```

| ID | Име | Презиме |
|---|---|---|
| 1 | Марко | Маркоски |

| ID | Име | Презиме |
|---|---|---|
| 2 | Ана | Андоновска |

Вториот начин е со користење на методот `update`.

```
// најди го студнетот со ID = 1 (Марко)
$student = Student::find(1);

$student::update([
    'име' => 'Петар'
    'презиме' => 'Петровски'
]);
```

За да може да работи `update` мора да ги имаме сите атрибути дефинирано во `fillable`, исто како во `create`.

## Delete

Бришење на запис од базата може да се направи на неколку начини, со помош на `delete` или `destroy`.

`delete` го брише записот за одреден елемент, додека пак `destory` ги брише записите според примарниот клуч

```
$student = Student::find(1);
$student.delete();

// Student::destory(1);
// Student::destory([1,2]);
```

## one-to-one, one-to-many

Како што може да забележите за да се креира `one-to-one` или `one-to-many` врска доволно е само да го дефинираме надворешниот клуч и да му зададеме вредност.

```
$profile = new StudentProfile();
$profile->student_id = 1;
$profile->save();
```

Но Laravel ни нуди полесен начин на поврузавње на ваквите типови на врски.

На пример:

Ја имаме табелата `Student`:

| ID | Име | Презиме |
|---|---|---|
| 1 | Марко | Маркоски |
| 2 | Ана | Андоновска |

И табелата `Profile`:

| ID | Пол | student_id |
|---|---|---|
| 1 | М | 1 |

И ги имаме следниве модели:

```php
class Student extends Model
{
    public function profile()
    {
        return $this->hasOne(Profile::class);
    }
}

class Profile extends Model
{
    public function student()
    {
        return $this->belongsTo(Student::class);
    }
}
```

Како што може да видите имаме `one-to-one` врска помеѓу `Student` и `Profile`. `Profile` чува надворешен клуч кон `Student`.

Доколку сакаме да создадеме нов `Profile` при креирањето на `Student` може тоа да го направиме на овој начин:

```php
// наместо find, може да креирате и нов студент
$student = Student::find(2);
$profile = new StudentProfile();
// на овој начин се доделуваат атрибути
$profile->пол('Ж');
$student->profile()->save($profile);
```

Со помош на овој код, Laravel ќе ни креира нов запис во табелата `Profile` и во `student_id` ќе го додаде својот `id`.

По извршување на кодот табелата ќе изгледа вака:

| ID | Пол | student_id |
|---|---|---|
| 1 | М | 1 |
| 2 | Ж | 2 |

Овој начин е многу поефикасен и полесен.

# Controller

Задачата на еден контролер во Laravel е да преработи некое HTTP барање и да врати некаков `view` или `податоци`, во овој дел ќе го објасниме тоа.

# Создавање на нов контролер

За да создадеме нов контролер ние ја користиме командата:

```
php artisan make:controller ИмеController
```

Контролерите се зачувани во: `/app/Http/Controllers`.

# Работа со контролер

Доколку отворите некој контролер, може да приметите дека Laravel сам ни генерира неколку методи, ние може да ги користиме тие или да си креираме сопствени методи.

Во овој дел ќе научиме како да работиме со барањата и како да вратиме `view` или податоци од базата.

## Request

`Request` е библиотека која ни овозможува работа со HTTP барања, за да може да ја користиме оваа библиотека, само го додаваме `Request $request` како параметар во методот.

Со помош на `$request` ние може да пристапиме до сите испратени податоци: file, integer, string, ip, header.

Исто така може да направиме и валидација на податоците користејќи `$request->validate`.

На пример сакаме да ги испратиме следниве податоци:

```
{
    "name": "Ema",
    "integer": 12,
    "float": 3.14,
    "boolean": true,
    "custom-header": "CustomHeaderValue",
    "nestedArray": {
        "key1": 32,
        "key2": 33
    },
    "file": "cat.jpg"
}
```

Во контролерот:

```
<?php
namespace App\Http\Controllers;

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

class ExampleController extends Controller
{
    public function create(Request $request)
    {
        // Валидација на податоци
        // доколку не исполниме некој услов од валидацијата, нема да се изврши останатиот код и ќе ни врати само ерор.
        // required значи дека мора да е испратен тој податок
        $request->validate([
            'name' => 'required|max:255', // да нема над 255 карактери
            'integer' => 'required|integer',  // цел број
            'float' => 'required|numeric', // број
            'boolean' => 'required|boolean', // булеан
            'file' => 'file', // датотека
            'custom-header' => 'string', // текст
        ]);


        // со помош на input ние може да ги земеме сите податоци на една форма: текст,
        // цели броеви, децимални броеви, булеан вредности.
        $name = $request->input('name'); // Ema
        $integer = $request->input('integer'); // 12
        $float = $request->input('float'); // 3.14
        $boolean = $request->input('boolean'); // true

        $nestedArray = $request->input('nestedArray'); //nestedArray
```

```
        // Како да пристапиме до вредностите
        $key1 = $nestedArray['key1']; // 32
        $key2 = $nestedArray['key2']; // 33

        // прво проверуваме дали е испратена датотека со помош на hasFile
        if ($request->hasFile('file')) {
             $file = $request->file('file'); // cat.jpg
        }

        // header кој бил испратен
        $header = $request->header('custom-header'); //CustomHeaderValue

        // IP адреса на испраќачот
        $ip = $request->ip();

        Example::create({
            'name': $name,
            'integer': $integer,
            'ip': $ip,
            ...
        });
    }
}
```

## Работа со датотеки

Чување на податоци директно во базата е лоша имплементација, Laravel го решава овој проблем така што ни дозволува зачувување на датотеките на така наречени дискови. Дисковите се подфолдери во `/app/storage` со кои ние може да додаваме, бришеме или праивме измени на податоците.

Со помош на библиотеката `Storage` ние може да креираме диск `Storage::disk('име')`.

Се импортира со:

```
use Illuminate\Support\Facades\Storage;
```

Пример да додадеме и избришеме некоја слика од `public` дискот.

```


$folder = '/image';
$file = $request->file('image');
// во /app/storage/public/image/ (image е всушност под фолдер) ќе го смести
$file
$path = Storage::disk('public')->putFile($folder, $file);

// Втор пример:
```

```
// $path = Storage::disk('public')->putFile('/avatar', $file);
// фајлот ќе биде сместен во /app/storage/public/avatar


// бришење на фајлот
Storage::disk('public')::delete('/image/image.png');
```

# Враќање на запис од база

Во секцијата за Модел објаснивме како се додава, менува и брише запис од база, но не објаснивме како се селектира запис или податок од база, во овој дел ќе го објасниме тоа.

## Raw SQL

Прво ќе објасниме како да пишувате "чист" SQL, во вториот дел ќе го ојасниме препорачаниот начин со помош на Eloquent.

Со помош на `DB::select("")` ние може да напишеме SQL прашалник на пример:

```
$user = DB::select('SELECT * FROM user WHERE user_id = 3');
```

Доколку сакате да додадете параметар при селектирањето тоа се прави со помош на `?` додека пак на крајот од прашалникот треба да ги излистата податоците во []. Кога имаме поголем број на '?' тогаш распоредот е битен, односно тие се земмат секвенцијално првиот прашалник со првиот параметар, вториот прашалник со вториот параметар итн..

На пример

```
$user_id = 5;
$age = 23;

// Битен е распоредот првиот прашалник е првиот параметар во []
// односно првиот прашалник ќе ја има вредноста на $user_id,
// додека вториот прашалник ќе ја има вредноста на $age
$user = DB::select('SELECT * FROM user WHERE user_id = ? AND age = ?',
[$user_id,$age]);

// Ова е еднакво на:
$user = DB::select('SELECT * FROM user WHERE user_id = 5 AND age = 23');

// Доколку го смениме распоредот
$user = DB::select('SELECT * FROM user WHERE user_id = ? AND age = ?',
[$age,$user_id]);

// Ова е еднакво на:
$user = DB::select('SELECT * FROM user WHERE user_id = 23 AND age = 5');
```

# Eloquent Query Builder

Вториот начин е со користење на 'Eloquent Query Builder' (слично е на JPARepository во Spring Boot). Овој начин е стандард е препрачан од страна на Laravel, па затоа ќе го објасниме.

За таа цел имаме следните модели Student и Course. Со следниве податоци во база:

Student:

| ID | Name | Age |
|---|---|---|
| 1 | Петар | 23 |
| 2 | Ана | 22 |
| 3 | Марко | 18 |
| 4 | Елена | 19 |
| 5 | Иван | 21 |

Course:

| ID | Name |
|---|---|
| 1 | Дискретна Математика |
| 2 | Структурно Програмирање |

Student_Course

| student_id | course_id |
|---|---|
| 2 | 1 |
| 5 | 2 |
| 1 | 1 |
| 1 | 2 |

## Основа

### all()

`all()` ќе ни ги врати сите записи од моделот на кој го повикуваме на пример да ги земеме сите записи од моделот Studnet:

```php
// ќе ги врати сите студенти
$students = Student::all();
```

**find($id)**

Ќе ни го врати записот со дефинираниот примарен клуч.

```php
// ќе ни го врати студентот со ID = 3 односно Марко
$student = Student::find(3);
```

Доколку направиме `return $student`, ова ќе ни врати JSON објект:

```
[
    "id": 1,
    "name": "Петар",
    "age": 23
]
```

Ова важи за сите примери.

**where()**

Се користи за филтрирање на податоците според некој услов.

- `=` : еднакво
- `<>` или `!=` : не еднакво
- `>` или `<` : поголемо, помало
- `>=` и `<=` : поголемо или еднакво
-

```php
$student = Student::where('name', 'Ана')->get();
$students = Student::where('age', '>', 20)->get();
$students = Student::where('age', '=', 20)->get();
$students = Student::where('age', '<=', 20)->get();
...
```

**orWhere()**

Се користи доколку сакаме да филтрираме по уште еден параметар на пример по години.

```
// Прво ќе ги земе сите студенти кои имаат над 20 години па потоа
//
$students = Student::where('age', '<', 20)->orWhere('name', 'Ана')->get();
```

За операцијата `and` се користи само where на пример:

```
// ова е and, студент постар од 20 години и студент со името Ана
$students = Student::where('age', '<', 20)->where('name', 'Ана')->get();
```

**whereIn()**

Проверува дали одреден атрибут е дел од некоја низа на вредности може да се 3 може да се 100.

```
// ќе ги врати студентите кои имаат 21 и 23 години во случајот Иван и Петар.
$students = Student::whereIn('age', [21, 23])->get();
```

**whereNull()**

Ќе провери дали одреден атрибут е null.

```
$students = Student::whereNull('age')->get();
//за повеќе атрибути:
$students = Student::whereNull('age')->whereNull('name')->get();
```

**whereBetween()**

Кога сакаме да провериме дали некој атрибут е помеѓу две вредности.

```
// Ќе ги врати сите студенти кои имаат помеѓу 19 и 22 години.
$students = Student::whereBetween('age', [19, 22])->get();
```

**Негација**

За да ги направите негација на операторите кои ги објаснивме само треба да додадете whereNot и името на пример:

```php
$students = Student::whereNot('age', '>', 20)->get();
$students = Student::whereNotNull('age')->get();
$students = Student::whereNotBetween('age', [19, 22])->get();
.
.
.
```

**first()**

Ќе ни го врати првиот запис кој исполнува некој услов

```php
// Ќе ни го врати првиот студент кој има над 20 години во нашиот случај Петар со ID = 1
$student = Student::where('age', '>', 20)->first();
```

**get()**

Ќе ни ги врати сите записи кои исполнуваат одреден услов.

```php
// Ќе ни ги врати сите студенти кои имаат над 20 години (Петар, Ана, Иван)
$students = Student::where('age', '<', 20)->get();
```

**orderBy()**

Ќе ги сортира податоците според некој атрибут, orderBy е `assending`.

```php
// Ќе ги соритра од најмал до најголем
$students = Student::orderBy('age')->get();
// Ќе ги сортира од најголем до најмал.
$students = Student::orderByDesc('age')->get();

// Исто така може да го дефинираме типот на сортирање во самиот orderBy
$students = Student::orderBy('age', 'desc')
                   ->orderBy('name', 'asc')
                   ->get();
```

### skip() и take()

`skip($n)` ќе ги прескокне првите $n (цел број) записи.
`take($n)` ќе ги земе само првите $n (цел број) записи.

```php
// Нема да врати првите 2 студенти (Ана и Петар)
$students = Student::skip(2)->get();
// Ќе ги врати само првите 2 студенти (Ана и Петар)
$students = Student::take(2)->get();
```

## Агрегатни фунцкии

Во овој дел ќе ги објасниме агрегатните функции `max`, `min`, `count` итн.

### count()

Оваа агрегатна фунцкија ни ги брои записите за тој модел.

```php
// Број на студенти во случајот 5
$numberOfStudents = Student::count();

// со where
$students = Student::where('age', '>', 20)->count();
```

### max()

Оваа агрегатна фунцкија ни ја враќа најголемата вредност за одреден атрибут.

```php
// најстар студент
$oldestStudent = Student::max('age'); // 23
```

### min()

Оваа агрегатна фунцкија ни ја враќа најмалата вредност за одреден атрибут.

```php
// најмлад студент
$yongestStudent = Student::min('age'); // 18
```

**avg()**

Оваа агрегатна функција ни го враќа просекот за некој атрибут.

```php
// просек
$ageAverage = Student::min('age'); // 20.6
```

## with()

Во делот на модели ние објаснивме како се прават врски помеѓу моделите `one-to-one`, `one-to-many`, `many-to-many`. Во овој дел ние ќе објасниме како да ги користиме овие функции.

На пример доколку во моделите `Student` и `Course` со следната `many-to-many` врска.

```php
// во Student моделот
class Student extends Model{
    public function courses() {
        return $this->belongsToMany(Course::class);
    }
}

// во Course модело
class Course extends Model{
    public function students() {
        return $this->belongsToMany(Student::class);
    }
}
```

И доколку сакаме да повлечеме информациите од едниот модел вклучувајќи ги и

```php
// најди го студентот со ID = 1 (Петар) и врати ги сите негови курсеви
$student = Student::with('courses')->find(1);
```

Ова ќе ни врати:

```
{
    "id": 1,
    "name": "Петар",
    "age": 23,
    "courses": [
        {
            "id": 1,
```

```
            "name": "Дискретна Математика"
        },
        {
            "id": 2,
            "name": "Структурно Програмирање"
        }
    ]
}
```

Доколку не го користиме `with()` тогаш нема да ни бидат прикажани курсевите, на пример:

```php
// најди го студентот со ID = 1 (Петар)
$student = Student::with('courses')->find(1);
```

Ова ќе ни врати:

```
{
    "id": 1,
    "name": "Петар",
    "age": 23
}
```

Истото може да се направи и обратно, да ги повлечеме информациите на курсот но и неговите студенти:

```php
// најди ги сите студенти кои го слушаат курсот со ID = 2 (Стуктурно програмирање)
$course = Course::with('students')->find(2);
```

Ова ќе ни врати:

```
{
    "id": 2,
    "name": "Структурно Програмирање",
    "students": [
        {
            "id": 1,
            "name": "Петар",
            "age": 23
        },
        {
            "id": 5,
```

```
            "name": "Иван",
            "age": 21
        }
    ]
}
```

Доколку имаме повеќе врски на пример во моделот Course имаме две врски `proffesors` и `lectures` ние може да ги земеме сите нив.

```
$course = Course::with('students','proffesors','lectures')->find(1);
```

ќе ни врати нешто како:

```
{
    "id": 2,
    "name": "Структурно Програмирање",
    "students": [
        {
            "id": 1,
            "name": "Петар",
            "age": 23
        },
        {
            "id": 5,
            "name": "Иван",
            "age": 21
        }
    ],
    "proffesors": [
        ...
    ],
    "image" : [
        ...
    ]

}
```

**$hidden**

Како што може да забележите во претходниот пример кога враќаме некаков JSON ние ги враќаме сите податоци, но некогаш не сакаме да го направиме тоа, па затоа во моделите постои променлива `protected $hidden = ['име_на_атрибут',];` со која може да не дозволиме враќање на тој податок во JSON.

На пример:

```
// врати го студентот со ID = 1 (Петар).
$student = Student::find(1);
```

Ова ќе ни врати

```
[
    "id": 1,
    "name": "Петар",
    "age": 23
]
```

Но доколку сакаме да не го вратиме `age` и `name`, доволно е само да го дефинираме во моделот на пример:

```
class Student extends Model {
    protected $hidden = ['name','age'];
}
```

Сега доколку го пробаме истиот код:

```
$student = Student::find(1);
```

Ова ќе ни врати само:

```
[
    "id": 1
]
```

**join**

Доколку сакаме да споиме две или повеќе табели или модели, ние може да ги користиме `left_join`, `right_join` итн.

Нивната структура изгледа вака:

```
Model::join('име_на_табелата', 'атрибут', 'оператор','атрибут', $type)-
>select('параметри')->get();
```

```
// име на табелата се дефинира во самата миграција
// атрибут од првата или втората табела
// оператор = <> > < итн
// select('') што селектираме, мора да дефинирате од која табела
// ќе ги селектирате параметрите на пример select('student.name')

// inner
$course = Course::join('student_course', 'courses.id', '=',
'student_course.course_id')->select('courses.*')->get();
// left_join
$course = Course::left_join('student_course', 'courses.id', '=',
'student_course.course_id')->select('courses.*')->get();
// right_join
$course = Course::right_join('student_course', 'courses.id', '=',
'student_course.course_id')->select('courses.name')->get();
```

# Рути

Во Lravel, `web.php` е задолжен за хендлање сите HTTP барања, `web.php` се наоѓа во `/app/routes`

Со помош на библиотеката (фасада) `Route` ние ќе може да ги обработиме сите барања.

## Создавање на нова рута

Доколку сакате барањето да ви биде обработено од страна на некој контролер се користи ова:

`Route::тип('url', 'Controller@method');`

`Route::тип('url', [Controller::class, 'method']);`

- `тип` може да биде: `GET`, `POST`, `DELETE`, `PUT`.
- `url` е патеката, на пример `/` или `/home`
- `Controller` некој контролер.
- `method` е метод во контролерот.

Пример:

```
Route::get('/create', [PostController::class, 'create']);
Route::post('/create', [PostController::class, 'store']);
```

Доколку сакате да вратите нешто директно од `web.php` без користење на контролер тогаш може да напишете функција на овој начин:

```
Route::get('/', function () {
    return 'Return something';
});
```

Исто така може и да групираме рути со заеднички префикс, на пример доколку имаме `/post/create` и `/post/delete`, можеме да ги групираме со користење на `Route::prefix('')->group(function({}))` :

```
Route::prefix('post')->group(function () {
   Route::post('/create', [PostController::class, 'create']);
   Route::post('/delete', [PostController::class, 'delete']);
});
```

Во контролерот:

```
class PostController extends Controller
{
    // Пивкан од страна на /create
    public function create(Request $request): RedirectResponse
    {
        return null;
    }

    // Пивкан од страна на /delete
    public function delete(Request $request): RedirectResponse
    {
        return null;
    }

}
```

# Структура на Vue

# Вовед

# Креирање на нов проект

# Структура на Vue

Vue
├── Components
├── Layouts
├── Pages
├── **app.js**
├── bootstrap.js

# Креирање на навигациски бар


Креираме нова компонента во директориумот "Components" со име "Navbar.vue".
На навигацискиот бар воочуваме 3 под-компоненти. Првата е делот за пребарување, втората е логото, и третата е делот за најавен корисник и прилагодувања. Значи, креираме уште три нови компоненти кои ќе бидат деца на "Navbar". Тоа се: "SearchBar.vue", "ApplicationLogo.vue", и "Dropdown.vue". Зошто го правиме ова? Ова се практики кои придонесуваат до реискористлив код (логото може да се најде и во друга компонента, пребарувачот може да бара по различна содржина, дропдаун менито може да содржи и други опции, итн..)

## Креирање на ApplicationLogo.vue

Поради брзина и практичност решивме да работиме со svg (Scalable Vector Graphics) слики и самиот код да го ставиме како vue компонента.

```
<template>
    <svg viewBox="0 0 162 49" fill="none" xmlns="http://www.w3.org/2000/svg">
        <path
```

```
            d="M65.1203 42.0381C64.5732 42.0381 64.1157 41.8504 63.7576
41.4851C63.3995 41.1197 63.2205 40.6654 63.2205 40.1124V27.107C63.2205 26.5639
63.3995 26.1096 63.7576 25.7344C64.1157 25.369 64.5732 25.1814 65.1203
25.1814C65.6673 25.1814 66.1249 25.369 66.483 25.7344C66.841 26.0997 67.0201
26.554 67.0201 27.107V30.0695L66.8013 27.9562C67.03 27.4526 67.3384 27.0082
67.7064 26.6132C68.0744 26.2281 68.4922 25.9022 68.9597 25.6356C69.4272 25.369
69.9245 25.1814 70.4517 25.053C70.9788 24.9246 71.506 24.8654 72.0332
24.8654C72.6697 24.8654 73.1969 25.0431 73.6346 25.3986C74.0722 25.7541 74.2811
26.1787 74.2811 26.6626C74.2811 27.3539 74.102 27.8575 73.744 28.1537C73.3859
28.4599 72.998 28.608 72.5703 28.608C72.1724 28.608 71.8044 28.5389 71.4761
28.3907C71.1479 28.2426 70.7699 28.1735 70.3522 28.1735C69.9742 28.1735 69.5863
28.2624 69.1984 28.4401C68.8105 28.6179 68.4524 28.8944 68.1241 29.2795C67.7959
29.6548 67.5274 30.1288 67.3284 30.6818C67.1295 31.2348 67.03 31.8964 67.03
32.6469V40.1124C67.03 40.6556 66.851 41.1197 66.4929 41.4851C66.1348 41.8504
65.6773 42.0381 65.1203 42.0381Z"
            fill="#F20085"/>
        <path
            d="M84.2873 42.3541C82.487 42.3541 80.9254 41.9788 79.6125
41.2382C78.2995 40.4976 77.275 39.4804 76.5688 38.1967C75.8626 36.9129 75.5045
35.4712 75.5045 33.8517C75.5045 31.9655 75.8924 30.3559 76.6583 29.0129C77.4342
27.6798 78.4388 26.6528 79.692 25.9417C80.9354 25.2307 82.2582 24.8752 83.6607
24.8752C84.7349 24.8752 85.7594 25.0925 86.7243 25.5369C87.6891 25.9812 88.5345
26.5836 89.2805 27.344C90.0166 28.1143 90.6034 28.9931 91.0411 30.0103C91.4787
31.0175 91.6876 32.084 91.6876 33.2197C91.6677 33.7233 91.4688 34.1282 91.0809
34.4442C90.7029 34.7602 90.2553 34.9182 89.748 34.9182H77.6231L76.6683
31.768H88.3058L87.6095 32.4V31.5508C87.5697 30.9385 87.3509 30.3954 86.963
29.9115C86.5751 29.4276 86.0877 29.0425 85.5008 28.766C84.914 28.4895 84.2973
28.3414 83.6408 28.3414C83.0042 28.3414 82.4174 28.4204 81.8604 28.5883C81.3133
28.7561 80.8359 29.0425 80.4281 29.4375C80.0302 29.8325 79.7119 30.3756 79.4732
31.0472C79.2444 31.7187 79.1251 32.5679 79.1251 33.5949C79.1251 34.7305 79.3638
35.6884 79.8412 36.4784C80.3187 37.2684 80.9254 37.8609 81.6813 38.2757C82.4273
38.6806 83.233 38.8879 84.0785 38.8879C84.8642 38.8879 85.4809 38.8287 85.9484
38.7003C86.4159 38.5719 86.7939 38.4238 87.0724 38.2461C87.3509 38.0683 87.6195
37.9202 87.8482 37.7918C88.2262 37.6042 88.5843 37.5054 88.9225 37.5054C89.3899
37.5054 89.7779 37.6634 90.0763 37.9794C90.3846 38.2954 90.5338 38.6608 90.5338
39.0854C90.5338 39.6483 90.2354 40.1618 89.6486 40.6259C89.1015 41.0901 88.3257
41.495 87.331 41.8406C86.3662 42.1763 85.3417 42.3541 84.2873 42.3541Z"
            fill="#F20085"/>
        <path
```

```
            d="M108.895 18.7329C109.442 18.7329 109.9 18.9107 110.258
19.2662C110.616 19.6217 110.795 20.0858 110.795 20.6487V40.1124C110.795 40.6555
110.616 41.1197 110.258 41.485C109.9 41.8504 109.442 42.038 108.895
42.038C108.348 42.038 107.891 41.8504 107.533 41.485C107.174 41.1197 106.995
40.6654 106.995 40.1124V38.5719L107.692 38.8583C107.692 39.1348 107.542 39.4606
107.244 39.8458C106.946 40.2309 106.548 40.616 106.041 40.9913C105.533 41.3665
104.936 41.6924 104.25 41.9492C103.564 42.2158 102.818 42.3442 102.012
42.3442C100.55 42.3442 99.2271 41.9689 98.0435 41.2283C96.8598 40.4876 95.9249
39.4508 95.2385 38.1374C94.5522 36.824 94.2041 35.323 94.2041 33.6146C94.2041
31.8964 94.5522 30.3756 95.2385 29.0622C95.9249 27.7488 96.8499 26.7218 98.0136
25.9812C99.1774 25.2406 100.47 24.8653 101.883 24.8653C102.788 24.8653 103.624
25.0036 104.389 25.2702C105.145 25.5467 105.812 25.8923 106.369 26.3071C106.926
26.7317 107.363 27.1563 107.672 27.581C107.98 28.0155 108.129 28.371 108.129
28.6672L106.985 29.082V20.6388C106.985 20.0957 107.164 19.6315 107.523
19.2662C107.891 18.9205 108.348 18.7329 108.895 18.7329ZM102.49 38.8879C103.425
38.8879 104.23 38.6608 104.936 38.1966C105.633 37.7325 106.18 37.1005 106.568
36.3105C106.956 35.5106 107.155 34.6219 107.155 33.6344C107.155 32.6271 106.956
31.7285 106.568 30.9286C106.18 30.1287 105.633 29.4967 104.936 29.0425C104.24
28.5783 103.425 28.3512 102.49 28.3512C101.584 28.3512 100.779 28.5783 100.083
29.0425C99.3863 29.5066 98.8392 30.1386 98.4513 30.9286C98.0634 31.7285 97.8644
32.6271 97.8644 33.6344C97.8644 34.6219 98.0634 35.5106 98.4513 36.3105C98.8392
37.1104 99.3863 37.7424 100.083 38.1966C100.779 38.6608 101.584 38.8879 102.49
38.8879Z"
            fill="#F20085"/>
        <path
            d="M123.079 25.5665C124.521 25.5665 125.794 25.922 126.899
26.633C128.013 27.344 128.888 28.3315 129.544 29.5758C130.201 30.82 130.529
32.2716 130.529 33.901C130.529 35.5008 130.201 36.9327 129.544 38.2164C128.888
39.5002 128.003 40.5074 126.879 41.2382C125.755 41.9689 124.501 42.3442 123.109
42.3442C122.373 42.3442 121.667 42.2257 121 41.9986C120.334 41.7714 119.727
41.4554 119.18 41.0703C118.633 40.6852 118.146 40.2211 117.718 39.6977C117.29
39.1743 116.952 38.6213 116.703 38.0288L117.27 37.5844V41.1394C117.27 41.4159
117.181 41.6431 117.002 41.8307C116.823 42.0183 116.594 42.1171 116.315
42.1171C116.037 42.1171 115.808 42.0183 115.619 41.8307C115.43 41.6431 115.331
41.4159 115.331 41.1394V19.7106C115.331 19.4538 115.42 19.2366 115.599
19.049C115.778 18.8613 116.017 18.7626 116.315 18.7626C116.594 18.7626 116.813
18.8613 117.002 19.049C117.191 19.2366 117.27 19.4538 117.27
19.7106V30.2571L116.823 30.0103C117.031 29.3585 117.35 28.766 117.758
28.2327C118.165 27.6995 118.653 27.2255 119.2 26.8305C119.747 26.4355 120.354
26.1195 121.01 25.9022C121.667 25.6751 122.353 25.5665 123.079 25.5665ZM122.92
27.3637C121.776 27.3637 120.771 27.6402 119.906 28.1932C119.041 28.7462 118.354
29.5165 117.857 30.504C117.36 31.4915 117.111 32.6271 117.111 33.901C117.111
35.165 117.36 36.3007 117.857 37.3178C118.354 38.3349 119.031 39.1348 119.906
39.7076C120.771 40.2803 121.776 40.5766 122.92 40.5766C124.044 40.5766 125.029
40.2803 125.884 39.6977C126.739 39.1052 127.416 38.3053 127.913 37.2882C128.41
36.271 128.659 35.1453 128.659 33.901C128.659 32.6173 128.41 31.4915 127.913
30.504C127.416 29.5165 126.739 28.7462 125.884 28.1932C125.029 27.6402 124.034
27.3637 122.92 27.3637Z"
            fill="white"/>
        <path
```

```
            d="M149.686 33.9799C149.686 35.5797 149.338 37.0116 148.642
38.2756C147.945 39.5495 146.991 40.5469 145.787 41.2677C144.584 41.9886 143.211
42.354 141.659 42.354C140.157 42.354 138.795 41.9886 137.581 41.2677C136.368
40.5469 135.403 39.5495 134.697 38.2756C133.99 37.0017 133.632 35.5698 133.632
33.9799C133.632 32.3604 133.99 30.9286 134.697 29.6646C135.403 28.4005 136.368
27.4032 137.581 26.6724C138.795 25.9417 140.157 25.5664 141.659 25.5664C143.201
25.5664 144.574 25.9318 145.787 26.6724C146.991 27.4032 147.945 28.4005 148.642
29.6646C149.338 30.9187 149.686 32.3604 149.686 33.9799ZM147.786 33.9799C147.786
32.6962 147.518 31.5606 146.991 30.5632C146.463 29.5658 145.737 28.7857 144.822
28.2129C143.907 27.6402 142.853 27.3637 141.669 27.3637C140.525 27.3637 139.491
27.65 138.566 28.2129C137.641 28.7758 136.905 29.5658 136.358 30.5632C135.821
31.5606 135.552 32.6962 135.552 33.9799C135.552 35.2637 135.821 36.3993 136.358
37.377C136.895 38.3645 137.631 39.1446 138.566 39.7272C139.501 40.3 140.535
40.5962 141.669 40.5962C142.853 40.5962 143.907 40.3099 144.822 39.7272C145.737
39.1545 146.463 38.3645 146.991 37.377C147.518 36.3895 147.786 35.2538 147.786
33.9799Z"
            fill="white"/>
        <path
            d="M152.7 26.0108H160.09C160.349 26.0108 160.558 26.0997 160.727
26.2774C160.896 26.4552 160.985 26.6724 160.985 26.9193C160.985 27.176 160.896
27.3834 160.727 27.5513C160.558 27.7192 160.349 27.808 160.09
27.808H152.7C152.451 27.808 152.232 27.7192 152.053 27.5414C151.874 27.3637
151.785 27.1464 151.785 26.8995C151.785 26.6428 151.874 26.4354 152.053
26.2675C152.232 26.0898 152.451 26.0108 152.7 26.0108ZM155.972 21.567C156.251
21.567 156.47 21.6658 156.659 21.8534C156.848 22.041 156.927 22.2583 156.927
22.515V38.325C156.927 38.9175 157.007 39.352 157.166 39.6482C157.325 39.9445
157.534 40.142 157.783 40.2309C158.041 40.3197 158.29 40.3691 158.549
40.3691C158.718 40.3691 158.867 40.3395 159.006 40.2901C159.145 40.2407 159.295
40.2111 159.464 40.2111C159.653 40.2111 159.822 40.2901 159.951 40.4284C160.08
40.5666 160.16 40.7641 160.16 40.9912C160.16 41.2875 159.991 41.5344 159.653
41.7319C159.314 41.9294 158.917 42.0281 158.449 42.0281C158.26 42.0281 157.972
42.0084 157.594 41.9787C157.216 41.9491 156.818 41.8306 156.42 41.6331C156.022
41.4356 155.684 41.0801 155.405 40.5765C155.127 40.0729 154.998 39.352 154.998
38.404V22.515C154.998 22.2681 155.097 22.041 155.286 21.8534C155.475 21.6658
155.694 21.567 155.972 21.567Z"
            fill="white"/>
        <path
```

d="M47.2861 32.0544C48.4299 31.6989 49.5738 31.3533 50.6977
31.0175C51.3443 34.1479 51.9013 37.2586 52.3588 40.3495C52.6572 42.3343 50.6977
44.2304 48.0321 44.576C33.51 46.393 18.988 47.736 4.45607 48.6149C1.79038
48.7828 -0.228774 46.6794 0.0298367 43.9539C0.417754 39.757 0.934976 35.5897
1.5815 31.4323C2.71541 31.847 3.84933 32.2717 5.00313 32.6765C5.29158 35.3428
5.66955 37.9992 6.09725 40.6358C6.23651 40.6852 6.36581 40.7247 6.50506
40.7741C9.47909 41.7813 12.5327 42.4825 15.6261 42.8873H15.636C16.2527 42.9663
16.8694 43.0355 17.4861 43.0947C18.7294 43.2132 19.9727 43.2823 21.226
43.3021C22.8771 43.3317 24.5382 43.2922 26.2092 43.1638C27.8803 43.0453 29.5414
42.8478 31.2024 42.5911C32.4458 42.3936 33.6891 42.1665 34.9324 41.89C35.5491
41.7517 36.1658 41.6135 36.7825 41.4653H36.7924C39.8759 40.7148 42.9096 39.7471
45.8637 38.6312C45.993 38.5818 46.1323 38.5325 46.2616 38.4831C46.6495 36.3007
46.9976 34.1578 47.2861 32.0544ZM50.4192 29.7239C50.2999 29.7042 50.1805 29.714
50.0612 29.7437C48.6587 30.1485 47.2363 30.5732 45.7941 31.0175C45.5056 33.1505
45.1475 35.3132 44.7397 37.5153C44.7397 37.5153 44.7397 37.5153 44.7397
37.5252C44.2325 37.7029 43.7252 37.8906 43.2179 38.0584C42.7007 38.2362 42.1934
38.4041 41.6762 38.5621C41.4375 38.6411 41.1888 38.7201 40.9501 38.7991C40.592
38.9077 40.224 39.0163 39.846 39.125C39.4879 39.2336 39.1199 39.3323 38.7419
39.441C38.543 39.5002 38.354 39.5496 38.1551 39.599C37.9661 39.6483 37.7672
39.6977 37.5782 39.757C37.3793 39.8063 37.1803 39.8557 36.9814 39.9051C36.7825
39.9545 36.5935 40.0038 36.3946 40.0532C36.1956 40.1026 35.9967 40.1421 35.7978
40.1915C35.6287 40.231 35.4596 40.2705 35.2905 40.3001C34.9423 40.3791 34.6042
40.4482 34.266 40.5173C33.9875 40.5766 33.699 40.6358 33.4106 40.6852C33.2912
40.705 33.1719 40.7346 33.0525 40.7543C32.774 40.8037 32.4955 40.8531 32.217
40.9025C31.8191 40.9716 31.4113 41.0308 31.0134 41.0901C30.8444 41.1098 30.6952
41.1395 30.5261 41.1592C30.4266 41.179 30.3371 41.1888 30.2376 41.1987C29.9392
41.2382 29.6408 41.2777 29.3424 41.3172C29.1037 41.3468 28.855 41.3765 28.6064
41.3962C28.5169 41.4061 28.4174 41.416 28.3179 41.4258C28.0096 41.4555 27.7012
41.4851 27.4028 41.5147H27.3929C27.005 41.5443 26.6171 41.574 26.2291
41.6036C26.2092 41.6036 26.1993 41.6036 26.1794 41.6036C25.7915 41.6233 25.4036
41.653 25.0157 41.6628H25.0057C24.6974 41.6826 24.389 41.6826 24.0906
41.6925C24.0011 41.7023 23.9016 41.7023 23.8022 41.7023C23.5635 41.7122 23.3148
41.7122 23.0661 41.7122C22.7677 41.7122 22.4693 41.7122 22.1709 41.7122C22.0715
41.7122 21.972 41.7122 21.8825 41.7023C21.7134 41.7023 21.5542 41.7023 21.3951
41.6925C20.9972 41.6826 20.5894 41.6727 20.1916 41.6431C19.9131 41.6332 19.6346
41.6135 19.3461 41.5937C19.2267 41.5838 19.1074 41.574 18.988 41.5641C18.6996
41.5443 18.4211 41.5246 18.1326 41.495C17.7845 41.4653 17.4364 41.4258 17.0982
41.3863C16.9291 41.3666 16.76 41.3468 16.5909 41.3271C16.392 41.3073 16.193
41.2777 15.9842 41.2481C15.7852 41.2185 15.5962 41.1987 15.3973 41.1592C15.1984
41.1296 14.9994 41.1 14.8005 41.0703C14.6016 41.0407 14.4126 41.0111 14.2137
40.9716C14.0147 40.942 13.8158 40.9025 13.6268 40.863C13.2588 40.7938 12.8908
40.7247 12.5227 40.6358C12.1547 40.5568 11.7966 40.4778 11.4187 40.389C11.18
40.3297 10.9313 40.2705 10.6826 40.2013C10.1654 40.073 9.64818 39.9248 9.13096
39.7767C8.61374 39.6286 8.10646 39.4607 7.58924 39.2928V39.283C7.15159 36.6463
6.78357 33.9998 6.48517 31.3335C5.03297 30.8102 3.60066 30.2769 2.18825
29.7437C2.07883 29.7042 1.95947 29.6943 1.84012 29.714C2.49659 25.7344 3.27242
21.7745 4.16762 17.8442C4.79425 15.178 7.04218 13.1141 9.11107 13.2523C14.6016
13.5782 20.0822 14.0028 25.5727 14.5065V10.6453C25.5727 10.3886 25.702 10.1516
25.9208 10.0133C26.1396 9.87505 26.4082 9.8553 26.6469 9.95405L29.0142
11.0008V7.40629C27.3133 7.04091 26.0501 5.5399 26.0501 3.75252C26.06 1.67876
27.751 0 29.8398 0C31.9285 0 33.6195 1.67876 33.6195 3.75252C33.6195 5.5794
32.2966 7.10016 30.5559 7.43591V12.1661C30.5559 12.4228 30.4266 12.6598 30.2078
12.7981C29.989 12.9363 29.7204 12.9561 29.4817 12.8573L27.1144
11.8106V14.6447C32.5253 15.1582 37.9462 15.7507 43.3572 16.4321C45.4261 16.7086
47.6242 18.6145 48.2011 20.6487C49.0267 23.6902 49.7727 26.712 50.4192
29.7239ZM21.3454 22.8509C21.3852 21.893 21.0768 21.0141 20.5496 20.3525C20.0225

```
19.7007 19.2765 19.2662 18.4211 19.2168C16.7202 19.128 15.1486 20.6586 14.9398
22.6336C14.5916 25.9615 14.2833 29.2993 14.0346 32.6568C13.8755 34.6516 15.3575
36.2118 17.3071 36.1526C19.2665 36.0933 20.8878 34.4837 20.9475 32.5679C21.0569
29.3289 21.1962 26.0899 21.3454 22.8509ZM38.3043 32.3013C38.0457 29.3783 37.7473
26.4454 37.429 23.5125C37.3295 22.6435 36.8919 21.8337 36.2652 21.2314C35.6486
20.629 34.8329 20.2142 33.9875 20.155C32.2866 20.0266 30.9836 21.4091 31.0532
23.236C31.1925 26.2973 31.3118 29.3585 31.4312 32.4198C31.4909 34.2467 33.1122
35.6588 35.0717 35.5699C37.0112 35.4811 38.4734 34.0196 38.3043 32.3013Z"
            fill="#F20085"/>
    </svg>
</template>
```

Како што приметивте кодот е огромен и содржи шифри кои го цртаат нашето лого и ги означуваат неговите бои. (За секоја буква по една "path" ознака).
Ова е добра практика за ако имате варијанти на лого (темна и светла варијанта) или имате анимации (на покажување, на клик) може лесно да се сменат неговите "fill" својства.

## Креирање на Dropdown.vue

За да направиме наједонставна дропдаун компонента ни треба да размислиме на неколку работи.
Каков тип на податоци може да бидат најуниверзални (да се се реискористат) и што со податокот кој ќе биде селектиран?

```
<template>
    <div class="dropdown p-2 bg-[#141414] text-center text-white rounded inline-
block cursor-pointer">
        <div @click="toggleDropdown">
            <slot class="inline-block"></slot>
            <UserIcon class="mr-2 w-7 inline-block" v-if="this.navbar"/>
            <button type="button" class="dropdown-button">
                {{ selectedOption || placeholder }}
            </button>
            <AngleDownIcon class="ml-3 inline-block"/>
        </div>
        <ul v-if="isOpen" class="dropdown-list">
            <li v-for="{id, name} in options" :key="id"
@click="selectOption({id, name})" class="text-start cursor-pointer py-2">
                {{ name }}
            </li>
        </ul>
    </div>
</template>
```

По додавање на неколку класи од Tailwind и стилизирање на копчето за dropdown, време е да ја објасниме функционалноста.
Прво за да ни се отвораат и затвораат достапните опции треба некако да го тригернеме тоа. Епа со клик на копчето, во vue @click event-handler-от повикуваме рендерирање на додатен html код што ќе ни го отвора останатиот дел со опциите. Значи треба да знаеме дали копчето е притиснато или не, што уствари го означува state-от или состојбата.

```
<script>
import AngleDownIcon from "@/Components/Icons/AngleDownIcon.vue";
import UserIcon from "@/Components/Icons/UserIcon.vue";
export default {
    components: {AngleDownIcon, UserIcon},
    props: {
        options: {
            type: Array,
        },
        navbar: Boolean,
        placeholder: {
            type: String,
            default: 'Select an option',
        },
    },
    data() {
        return {
            isOpen: false,
            selectedOption: null,
        };
    },
    methods: {
        toggleDropdown() {
            this.isOpen = !this.isOpen;
        },
        selectOption({id, name}) {
            this.selectedOption = name;
            this.isOpen = false;
            this.$emit('option-selected', id);
        },

    },
};
</script>
```

Во  делот во Vue се пишува javascript или логиката на самата компонента. Во првиот ред се импортите на компонентите што се користат во самата компонента. Потоа за да биде реискористлива компонентата мора да напишиме "export default" и во тој блок наведуваме неколку работи:

- компонентите кои се користат (сликата на корисникот и стрелка за надолу)
- props: вредностите кои ги прима компонентата или се менуваат (опциите, дали се наоѓа на навбар, и што треба да пишува кога не е ништо селектирано)
- data() - ги дефинираме иницијалните состојби на state-овите, сметаме дека менито по дифолт е затворено и нема предефинирана селектирана опција.
- methods() - ги дефинираме методите кои ги користи самата компонента: toggleDropdown ја менува спротивната состојба на state-от isOpen (дали е отворено менито) и selectOption. Со оваа функција ја поставуваме одбраната опција во state-от на компонентата и го затвораме менито. Најпосле, одбраната опција ја кажуваме на родителот компонента со this.$emit('option-selected', id) и го праќаме id-то.

Најпосле ги вклопуваме потребните компоненти во навигацискиот бар и што потоа?

```
<script>
import ApplicationLogo from "@/Components/Icons/ApplicationLogo.vue";
import SearchIcon from "@/Components/Icons/SearchIcon.vue";
import UserIcon from "@/Components/Icons/UserIcon.vue";
import AngleDownIcon from "@/Components/Icons/AngleDownIcon.vue";
import SettingsIcon from "@/Components/Icons/SettingsIcon.vue";
import Dropdown from "@/Components/Dropdown/Dropdown.vue";
import ApiUtilis from "@/Helpers/ApiUtilis";

export default {
    components: {Dropdown, SettingsIcon, AngleDownIcon, UserIcon, SearchIcon,
ApplicationLogo},
    created() {
        this.fetchUser();
    },
    data() {
        return {
            user: Object
        }
    },
    methods: {
        async fetchUser() {
            try {
                const response = await ApiUtilis.fetchCurrentUser();
                this.user = response.data;
            } catch (e) {
                console.log("Error", e);
            }
        },
        handleRedirect(optionSelected) {
            if (optionSelected === 'profile') {
                window.location.href = '/user/' + this.user.name;
            }
            if (optionSelected === 'logout') {
                window.location.href = '/logout';
            }
        }
    }
}
</script>
```

```
<Dropdown
                                        :options="[{id: 'profile', name:
'Profile'},{id: 'logout', name: 'Logout'}]"
                                        :placeholder="user.name" :navbar="true"
                                        @option-
selected="handleRedirect($event)"
                                    />
```

- created: се извршува откако ќе се креира компонента (завршила со процесирање на се околку state-овите), па повикуваме од самиот back-end да ни врати кој е моменталниот корисник што е најавен. Ако имало некаква грешка при повикот, таа се пречати во catch делот.
- handleRedirect методот ја прима опцијата што е кликната во компоентатата Dropdown, и во зависно од тоа не носи на таа локација.

# Креирање на почетна страница

Со локација во директориумот креираме нова страница "Home.vue".
Според дизајнот во неа воочуваме неколку компоненти. Првата е навигацискиот бар (која ја имплементиравме досега), втората е левиот дел што содржи филтри, третата е десниот дел со опции за креирање, и четвратата и главната е делот каде што треба да се листаат објави.

# Креирање на компонента "Create.vue"

Компонентата е доста едноставна, содржи само две подкомпоненти што се едноставни икони.
Имаме две копчиња што треба да ни отворат форми за креирање на објава или заедница, но тоа ќе го видиме понатаму.

```
<template>
    <div class="bg-[#2D2D2D] inline-block">
        <div class="flex flex-col px-0 md:px-5 z-0">
            <div class="px-5 md:px-1 lg:px-5 py-3 inline-block">
                <a href="/post/create">
                    <div class="text-white inline-block pr-4 text-sm md:text-
base">Make a new post</div>
                    <PlusIcon class="inline-block"/>
                </a>
            </div>
            <div class="px-5 md:px-1 lg:px-5 pb-3 inline-block">
                <a href="/community/create">
```

```
                    <div class="text-white inline-block pr-4 text-sm md:text-
base">Make a new community</div>
                    <PlusIcon class="inline-block"/>
                </a>
            </div>
        </div>
        <WhiteSmallLogo class=" hidden md:block z-10 absolute top-0 right-0 -mt-
5 -mr-3"/>
    </div>
</template>

<script>
import PlusIcon from "@/Components/Icons/PlusIcon.vue";
import WhiteSmallLogo from "@/Components/Icons/WhiteSmallLogo.vue";

export default {
    name: "Create",
    components: {WhiteSmallLogo, PlusIcon}
}
</script>
```

# Креирање на компонента "Sort.vue"

Делот за сортирање треба да има две копчиња: дали да сортира по датум или по популарност. Значи треба да биде само едно копче притиснато во дадено време и секое од нив двете дава различен резултат. Сметаме дека иницијално објавите треба да бидат сортирани по датум.
Од тука воочуваме два state-ови new и top.

```
<script>
import TopIcon from "@/Components/Icons/TopIcon.vue";
import NewIcon from "@/Components/Icons/NewIcon.vue";

export default {
    name: "Filter",
    components: {NewIcon, TopIcon},
    data() {
        return {
            newPosts: Boolean,
            topPosts: Boolean
        }
    },
    emits: ['sort'],
    created() {
        this.newPosts = true;
        this.topPosts = false;
    },
    methods: {
        activateTopPosts() {
```

```
            this.topPosts = true;
            this.newPosts = false;
            this.sortEmmiter();
        },
        activateNewestPosts() {
            this.topPosts = false;
            this.newPosts = true;
            this.sortEmmiter();
        },
        sortEmmiter() {
            if (this.topPosts) {
                this.$emit('sort', 'top');
            }
            if (this.newPosts) {
                this.$emit('sort', 'new');
            }
        }
    }
}
</script>
```

- created - поставуваме newPosts да е true, додека topPost false.
- при кликање на секој од нив, спротивниот го менуваме на false така што никој пат нема да дозволиме да се два кликнати истовремено. И на крај резултатот го емитуваме (праќаме) на родител компонентата.

```
<template>
    <div class="py-2 md:py-5 bg-[#2D2D2D]">
        <div class="flex flex-row flex-wrap ">
            <div class="pl-0 md:pl-5 pr-0 md:pr-7 text-[#898989] text-sm mt-2.5
font-light hidden lg:block">Sort <span
                class="hidden xl:block">posts</span></div>
            <div class="lg:absolute lg:right-0 flex flex-row">
                <a class="cursor-pointer" @click="activateTopPosts()">
                    <div class="px-5">
                        <TopIcon class="h-5" :active="topPosts"/>
                        <div class=" lg:pt-2 text-xs font-light"
                             :class="{'text-white' : topPosts, 'text-gray-500' :
!topPosts}">Top
                        </div>
                    </div>
                </a>
                <a class="cursor-pointer" @click="activateNewestPosts()">
                    <div class="px-5">
                        <NewIcon class="h-5" :active="newPosts"/>
                        <div class="lg:pt-2 text-xs font-light"
                             :class="{'text-white' : newPosts, 'text-gray-500' :
!newPosts}">New
                        </div>
                    </div>
                </a>
            </div>
```

```
        </div>
    </div>
</template>
```

- TopIcon и NewIcon се компоненти кои го примаат state-от и менуваат боја на иконата во зависност дали е кликната опцијата или не.

```
<template>
    <svg width="25" height="32" viewBox="0 0 25 32" fill="none"
xmlns="http://www.w3.org/2000/svg">
        <path
            d="M22.1395 0.0214844H2.16959C1.06593 0.0214844 0.172607 0.914803
0.172607 2.01847C0.172607 3.12214 1.06593 4.01546 2.16959
4.01546H11.8177C11.6273 4.07137 11.4476 4.17522 11.2972 4.32699L4.52471
11.2379C3.76053 12.0114 4.30238 13.3347 5.38208 13.3347H10.1575V29.9763C10.1575
31.0799 11.0509 31.9733 12.1545 31.9733C13.2582 31.9733 14.1515 31.0799 14.1515
29.9763V13.3347H18.927C20.0067 13.3347 20.5485 12.0114 19.7843 11.2379L13.0119
4.32699C12.8615 4.17522 12.6817 4.07137 12.4914 4.01546H22.1395C23.2418 4.01546
24.1364 3.12214 24.1364 2.01847C24.1364 0.914803 23.2418 0.0214844 22.1395
0.0214844Z"
            :fill="active ? 'white' : 'gray'"/>
    </svg>
</template>

<script>
export default {
    name: "TopIcon",
    props: {
        active: Boolean
    }
}
</script>
```

# Креирање на компонента "Content.vue"

Бидејќи оваа компонента рендерира само објави, но само се повикува на различни места (почетна, профил, заедница).
Треба да чува листа од објави, треба да знаеме каде е повикана, и каков тип на сортитање е одбран од сестринската компонента.

```
<template class="relative">
    <div class="w-auto md:w-3/5 mx-5 md:mx-auto mt-48 md:mt-20 rounded-xl
relative bg-[#2d2d2d]">
```

```
        <div v-if="this.type==='home'" class="py-3 float-right" style="margin-
top: -70px;">
            <Dropdown :options="[{id: 'following', name: 'Following'},{id:
'trending', name: 'Trending'}]"
                      :placeholder="this.filter" @option-
selected="handleFilter($event)"
            />
        </div>
        <div class="px-1 md:px-10">
            <CommunityCard v-if="this.type==='community'"
                           :name="this.community.name"
                           :id="community.id"
                           :type="'community'"
                           :active-users="community.activeUsers"
                           :total-users="community.totalUsers"
            />
            <div v-if="posts.length === 0" class="text-white h-screen flex
justify-center items-center"> No posts yet.
            </div>
            <div class="md:py-10 py-10" v-for="post in posts">
                <Post :id="post.id"
                      :description="post.body"
                      :by-user="post.user?.userName"
                      :community-name="post.community?.name"
                      :community-id="post.community?.id"
                      :comments-number="post.comments_number"
                      :vote="post.vote"
                      :karma="post.karma"
                      :date="post.date"
                      :title="post.title"
                      :flair="post.flair"
                      :owner="post.user.id"
                      :user_id="user_id"
                      @deleteEmitter="handleDeletePost($event)"
                />
            </div>
            <div v-if="posts.length !== 0" class="text-white flex justify-center
items-center"> That's all !</div>
        </div>
    </div>
</template>
```

Според дизајнот, на почетна страница имаме копче што претставува филтер (објави од сите заедници, или само од тие заедници на кои корисникот е член). Па ги праќаме тие податоци како листа со клуч и вредност и одбраната опција ја хендламе во handleFilter настанот.
Исто така според дизајнот, во секоја заедница имаме картичка во овој дел каде што ги пишува основните информации за таа заедница. Па ако типот е заедница, тогаш вклучи ја и оваа картичка.
За објавите стандардно чуваме неколку податоци и тоа: идентификатор, опис, од кој корисник е, во која заедница припаѓа објавата, колку коментари има, каква карма има, дали тековно најавениот корисник има гласано на објавата, датум, категорија, и настан за бришење на објавата.

```
mounted() {

        ApiUtilis.fetchActiveUser()
            .then((response) => {
                this.user_id = response.data;
            });


        window.onscroll = () => {
            let bottomOfWindow =
                document.documentElement.scrollTop +
                window.innerHeight >= document.documentElement.offsetHeight -
200;

            if (bottomOfWindow) {
                this.currentPage = this.currentPage + 1;
                this.fetchData();
            }
        };
```

Во состојбата mounted (која ќе се додаде компонентата во DOM) се повикуваат неколку настани. Се фаќа ид-то на активниот корисник, и при скролање се проверув дали сме дошле најдолу на страната. Ако ова е исполнето, се зголемува бројот на страница и се фаќаат наново објави. Ова е груба имплементација на бесконечен скрол со пагинација.

```
watch: {
        sort(newVal) {
            this.currentPage = 1;
            this.posts = [];
            this.fetchData();
        },
        filter(newVal) {
            this.currentPage = 1;
            this.posts = [];
            this.fetchData();
        },
        flair(newVal) {
            this.currentPage = 1;
            if (this.flair.length === 0) {
                this.posts = [];
                return this.fetchData();
            } else {
                this.posts = this.posts.filter(post => {
                    if (post.flair) {
                        return post.flair.id === this.flair
                    } else return false;
                });
            }
        }
    },
```

При промена на состојбите сакаме да рефлектираме промена на некои други. Пример кога ќе притисмене на копчето sort, бидејќи тоа доаѓа од сестринска компонента треба да му кажеме на vue да ја очекува промената. Кога ќе се случи, сакаме да се ресетрира пагинацијата на првата страна, и да се исчистат и наново добијат објавите (рефреш на страната со запомнет сорт).
Истото се случува кога ќе притиснеме на filter dropdown.
Бидејќи имаме и филтрирање по достапни категории во рамки на една заедница, филтрирањето го правиме на фронт. Со што на секоја објава проверуваме дали го има истото флер ид на селектираното и ја менуваме листата со објави.

```
methods: {
        async fetchData() {
            const sortDto = {
                sort: this.sort,
            }
            if (this.type === 'home') {

                if (this.filter.toLowerCase() === 'following') {
                    try {
                        let response = await
ApiUtilis.getPaginatedFollowingPosts(this.currentPage, sortDto);
                        const newData = response.data.data;
                        this.posts = [...this.posts, ...newData];
                    } catch (error) {
                        console.error('Error fetching data:', error);
                    }
                } else {
                    try {
                        let response = await
ApiUtilis.getPaginatedTrendingPosts(this.currentPage, sortDto);
                        const newData = response.data.data;
                        this.posts = [...this.posts, ...newData];
                    } catch (error) {
                        console.error('Error fetching data:', error);
                    }
                }
            } else if (this.type === 'community') {
                const regexPattern = /[^/]+$/;

                const path = window.location.pathname;
                const match = path.match(regexPattern);

                if (match) {
                    const result = match[0];
                    try {
                        let response = await ApiUtilis.fetchCommunity(result);
                        this.community = response.data[0];
                    } catch (error) {
                        console.error('Error fetching data:', error);
                    }
                    try {
                        let response = await
ApiUtilis.fetchCommunityPosts(result, this.currentPage, sortDto)
```

```
                        const newData = response.data.data;
                        this.posts = [...this.posts, ...newData];
                    } catch (error) {
                        console.error('Error fetching data:', error);
                    }
                }
            } else if (this.type === 'user') {
                const regexPattern = /[^/]+$/;

                const path = window.location.pathname;
                const match = path.match(regexPattern);

                if (match) {
                    const userName = match[0];
                    try {
                        let response = await ApiUtilis.fetchUserPosts(userName,
this.currentPage, sortDto);
                        const newData = response.data.data;
                        this.posts = [...this.posts, ...newData];

                    } catch (error) {
                        console.error('Error fetching data:', error);
                    }
                }
            }
        },
    },
```

За да знаеме какви објави ќе прикажуваме, треба да знаеме на која страна се најдуваме или кој е типот наведен во props. Исто така треба да знаеме која е вредноста на сорт и да ја пратиме како DTO (Data Transfer Object). За почетна страница треба type да е home, и зависно кој е одбраниот филтер (дали од сите заедници или само од тие што ги следи корисникот) се обраќаме до два различни end-points кои ни враќаат листа со објави. Ги рефрешираме вредностите на state-от posts.
Ако имаме отворено специфична заедница, треба да ги листаме објавите од таа заедница. Па како го правиме тоа?
Го земаме идентификаторот на заедницата од самото URL (/community/{id}), па со помош на regex го земаме последниот стринг и праќаме повик до бекенд првично да ни ја врати таа заедница (да апдејтираме картичка, најважно инфо), па повторно правиме повик за да ни ги врати објавите од таа заедница.
И конечно, истиот пристап работи и за корисник (/user/{userName}).

# Креирање на форми за заедница и објава

За да провериме дали добро ни се листаат објавите, треба да креираме неколку. А за тоа треба и да припашаат на некоја заедница.

```
<form @submit.prevent="createCommunity" class="p-5">
                <h1 class="text-white text-2xl pb-5 text-center font-
bold">Create Community</h1>
                <div class="pb-5">
                    <div htmlFor="name" class="py-1 text-white">Name:</div>
                    <input type="text" id="name" v-model="form.name"
placeholder="Name"
                           class="rounded bg-[#515151] text-white">
                </div>
                <div class="pb-5">
                    <div htmlFor="description" class="py-1 text-
white">Description:</div>
                    <textarea id="description" v-model="form.description"
placeholder="Description"
                              class="rounded bg-[#515151] w-96 text-white">
</textarea>
                </div>
                <div class="pb-5">
                    <div htmlFor="rules" class="py-1 text-white">Rules:</div>
                    <textarea id="rules" v-model="form.rules" class="rounded bg-
[#515151] w-96 text-white"
                              placeholder="Rules"></textarea>
                </div>
                <div class="pb-5">
                    <div htmlFor="image" class="py-1 text-white">Image:</div>
                    <input type="file" id="image" @change="handleFileUpload"
class="rounded bg-[#515151] text-white">
                </div>
                <div htmlFor="name" class="py-1 pt-5 text-white">Flairs:</div>
                <div class="flex gap-2 py-2">
                    <input type="text" id="flair-name" v-model="flairName"
class="rounded bg-[#515151] text-white"/>
                    <div class="bg-[#CC0974] rounded-2xl inline-block">
                        <button type="button" class="px-4 py-2"
@click="addFlair()">Add</button>
                    </div>
                </div>
                <div class="pb-5">
                    <div v-for="flair in flairs">
                        <div class="py-2">
                            <div class="py-2 px-4 rounded-2xl bg-black text-
white inline-block">
                                {{ flair }}
                                <span class="pl-2 text-sm hover:text-gray-500
cursor-pointer"
                                      @click="removeFlair(flair)">x</span>
                            </div>
                        </div>
                    </div>
                </div>
                <div class="bg-[#CC0974] rounded-2xl inline-block">
                    <button type="submit" class="px-4 py-2">Create</button>
                </div>
                <a href="/">
                    <div class="border border-[#515151] rounded-2xl inline-block
float-right">
```

```
                        <div class="px-4 py-2 text-white">Cancel</div>
                    </div>
                </a>
            </form>
```

Креираме стандардна html форма со полиња за име, опис, слика, категории и правила. Она што е ново во овие полиња е v-model. Тука ја поврзуме внесената вредност на полето со некоја во податоците на самата компонента.

```
export default {
    components: {Navbar, Head},
    data() {
        return {
            showDiv: false,
            flairName: '',
            flairs: []
        };
    },
    setup() {
        const form = useForm({
            name: '',
            description: '',
            rules: '',
            image: '',
            flairs: []
        });

        const editForm = useForm({
            name: '',
            description: '',
            rules: '',
            flairs: []
        });

        const handleFileUpload = (event) => {
            const file = event.target.files[0];
            form.data.image = file;
            editForm.data.image = file;
        };

        const createCommunity = () => {
            form.post(route('communities.store'), {
                onSuccess: () => {
                    // Handle success, e.g., redirect to communities index
                    // You can customize this based on your application's flow
                    console.log("I saved the community!")
                },
                onError: () => {
                    // console.log("The /r/" +
document.getElementById('text').toString().valueOf() + " already exsist!");
                },
            });
```

```
        };

        return {
            form,
            handleFileUpload,
            createCommunity,
            editForm
        };


    },

    methods: {
        deleteCommunity(id) {
            if (confirm('Are you sure you want to delete this community?')) {
                axios.post(`/communities/${id}/delete`)
                    .then(response => {
                        // Handle successful deletion, e.g., update UI or show
success message
                        console.log('Community deleted successfully!');
                    })
                    .catch(error => {
                        // Handle error case, e.g., show error message
                        console.error('Error deleting community:', error);
                    });
            }
        },
        addFlair() {
            if (this.flairName.length !== 0 &&
!this.flairs.includes(this.flairName)) {
                this.flairs.push(this.flairName);
                this.form.flairs = this.flairs;
                this.flairName = ''
            }
        },
        removeFlair(flairName) {
            const index = this.flairs.findIndex(flair => flair === flairName);
            if (index !== -1) {
                this.flairs.splice(index, 1);
                this.form.flairs = this.flairs;
            }
        },
        showEdit() {
            this.showDiv = !this.showDiv;
        },
        editCommunity(id) {
            ApiUtilis.editCommunity(id, this.editForm);
        },
        deleteFlair(id) {
            ApiUtilis.deleteFlair(id);
        },
        createFlair(id) {
            const formData = {
                name: this.flairName,
                community_id: id,
            };
```

```
            ApiUtilis.createFlair(formData);
        },
        handleFileUpload(event) {
            const file = event.target.files[0];
            this.form.image = file;
            this.editForm.image = file;
        },
    }

};
</script>
```

Бидејќи истата форма служи и за едитирање на постоечка заедница, некои својства се сетираат и во editForm и form. Разликата е на edit што формата е автоматски пополнета со постоечките податоци.

- deleteCommunity(id) - можеме да ја избришиме постоечката заедница, со што првично ќе ни се појави прозорец кој ќе праша дали сме сигурни. Ако одговориме да правиме повик до бекенд со кој кажуваме кое е ид-то на заедницата која треба да се избрише.
- addFlair() - додаваме категорија во листата со категории која ја чуваме за секоја заедница. Меѓутоа имаме прво проверка дали пробуваме да додадеме празна категорија или категоријата што сакаме да ја додадеме веќе постои во листата.
- removeFlair(flairName) - бришеме категорија по нејзиното име. Ако некоја категорија од листата со додадени сакаме да ја избришиме само пребаруваме во листата по нејзиното име и ја бришеме.
- createFlair(id) - креираме категорија во самата заедница
- handleFileUpload(event) - при прикачување на слика на самата форма, го земаме file-от од самиот event target и го сетираме мануелно на самите useForm-и.

```
<template>
    <Head><title>Create Post</title></Head>
    <Navbar/>
    <div class="w-auto md:w-3/5 mx-5 md:mx-auto mt-48 md:mt-20 rounded-xl
relative bg-[#2d2d2d]">
        <div class="">
            <h1 class="text-white text-2xl pb-5 text-center font-bold">Create
Post</h1>
            <form class="p-5" @submit.prevent="createPost"
enctype="multipart/form-data">
                <div class="pb-5">
                    <div htmlFor="name" class="py-1 text-white">Community</div>
                    <Dropdown
                        placeholder="Select"
                        :options="communityOptions"
                        @option-selected="handleSelectedCommunity"
                        id="communityId"
                        v-model="form.communityId"></Dropdown>
                </div>
                <div class="pb-5">
                    <div htmlFor="" class="py-1 text-white">Title:</div>
```

```
                    <input type="text" id="title" v-model="form.title"
class="rounded bg-[#515151] text-white w-full">
                </div>
                <div class="pb-5">
                    <div htmlFor="" class="py-1 text-white">Content:</div>
                    <textarea type="text" id="content" v-model="form.body"
                              class="rounded bg-[#515151] w-96 text-white w-
full"></textarea>
                </div>


                <div class="pb-5">
                    <div htmlFor="image" class="py-1 text-white">Image:</div>
                    <input type="file" name="image" id="image"
@change="onFileChange"
                           class="rounded bg-[#515151] text-white">
                </div>

                <div class="pb-5">
                    <div class="py-1 text-white">Mark as spoiler:</div>
                    <input type="checkbox" id="spoiler" @click="form.spoiler =
!form.spoiler" class="bg-black">
                </div>

                <div class="pb-5">
                    <div htmlFor="flair" class="py-1 text-white">Flair</div>
                    <Dropdown
                        placeholder="Select"
                        :options="flairOptions"
                        @option-selected="handleSelectedFlair"
                        id="flairId"
                        v-model="form.flair"></Dropdown>
                </div>

                <div class="bg-[#CC0974] rounded-2xl inline-block">
                    <button type="submit" class="px-4 py-2">Create</button>
                </div>
                <a href="/">
                    <div class="border border-[#515151] rounded-2xl inline-block
float-right">
                        <div class="px-4 py-2 text-white">Cancel</div>
                    </div>
                </a>


            </form>
        </div>
    </div>
</template>

<script>
import Navbar from "@/Components/Navbar/Navbar.vue";
import {useForm, Head} from "@inertiajs/vue3";
import Dropdown from '@/Components/Dropdown/Dropdown.vue';
import ApiUtilis from "@/Helpers/ApiUtilis";
```

```
export default {
    name: "MakePost",
    components: {Dropdown, Navbar, Head},
    data() {
        return {
            communityOptions: [],
            flairOptions: [],
            selectedCommunity: null,
        }
    },
    async created() {
        try {
            const response = await ApiUtilis.fetchUserCommunities();
            this.communityOptions = response.data;
        } catch (error) {
            console.error('Error fetching options:', error);
        }
    },
    setup() {
        const form = useForm({
            communityId: '',
            title: '',
            body: '',
            image: null,
            flair: '',
            spoiler: false
        })
        const createPost = () => {
            form.post(route('posts.create'), {
                onSuccess: () => {
                    // Handle success, e.g., redirect to communities index
                    // You can customize this based on your application's flow
                    console.log("I saved the post!")
                },
                onError: () => {
                    console.log("error");
                },
            });
        }

        return {
            form,
            createPost
        }
    },
    methods: {
        handleSelectedCommunity(option) {
            this.selectedCommunity = option;
            this.form.communityId = option;
            this.fetchFlairs();
        },
        handleSelectedFlair(option) {
            this.form.flair = option;
        },
        async fetchFlairs() {
```

```
            try {
                const response = await
ApiUtilis.fetchCommunityFlairs(this.form.communityId);
                this.flairOptions = response.data;
            } catch (error) {
                console.error('Error fetching options:', error);
            }
        },
        onFileChange(event) {
            this.form.image = event.target.files[0];
        }
    }
}
</script>
```

# Креирање на "Post.vue"

Откако успешно креиравме заедница и додадовме неколку објави во неа, време е да видиме како ќе се рендерираат.

```
<template>
    <div class="w-full mx-auto z-0">
        <div class="pt-2 px-5 md:px-7">
            <div class="grid grid-cols-2 relative">
                <a :href="'/community/'+ communityId">
                    <div class="inline-block relative">
                        <TestIcon class="z-10 inline-block w-4 h-4"/>
                        <div class="ml-2 text-white inline-block text-base
md:text-lg">{{ communityName }}</div>
                        <span class="text-xs ml-0.5 bottom-0 absolute text-
[#898989] mb-1 font-light">/community</span>
                    </div>
                </a>
                <div class="inline-block text-right mr-20">
                    <a :href="`/user/` + byUser">
                        <span
                            class="text-xs bottom-0 text-[#898989] mb-1 font-
light hidden md:inline-block">Posted by</span>
                        <span class="text-xs ml-1 bottom-0 absolute text-white
mb-1.5">{{ byUser }}</span>
                    </a>
                </div>
            </div>
            <div class="py-1">
                <div v-if="flair" class="px-3 rounded-2xl bg-gray-500 text-sm
inline-block">
                    {{ flair && flair.name }}
                </div>
```

```
                <h1 class=" text-2xl font-semibold text-[#F20085]">{{ title }}
</h1>
            </div>


            <div>
                <img v-if="image && spoiler" class="object-cover w-full h-60
object-center rounded"
                     :src="image"/>
                <img v-else-if="image && !spoiler" class="object-cover w-full
blur-sm h-60 object-center rounded"
                     :src="image"/>
            </div>
            <div class="px-2 pt-3">
                <div class="w-full grid grid-cols-2 relative">
                    <div>
                        <div class="inline-block mr-2">
                            <VoteUpIcon :voteUp="this.voteUpState"
                                        class="cursor-pointer w-4 h-4 inline-
block mx-1"
                                        @click="voteUp(id)"/>
                            <div class="inline-block">
                                <span class="text-white text-sm">{{ childKarma
}}</span>
                            </div>
                        </div>
                        <div class="inline-block">
                            <VoteDownIcon
                                :voteDown="this.voteDownState"
                                class="cursor-pointer w-4 h-4 inline-block mx-1"
                                @click="voteDown(id)"/>
                        </div>
                        <div class="inline-block">
                            <CommentIcon class="w-4 h-4 inline-block ml-3
cursor-pointer"
                                         @click="this.toggleComments()"/>
                            <span
                                class="mx-2 inline-block text-[#898989] text-xs
font-light hidden md:inline-block">{{
                                    commentsNumber
                                }} Comments</span>
                        </div>
                        <div class="inline-block">
                            <ShareIcon class="w-4 h-4 inline-block ml-3"/>
                            <span class="mx-2 inline-block text-[#898989] text-
xs font-light hidden md:inline-block">Share</span>
                        </div>


                        <div v-if="user_id === owner" class="inline-block
cursor-pointer" @click="deletePost">
                            <DeleteIcon class="w-4 h-4 inline-block ml-3"/>
                            <span class="mx-2 inline-block text-[#898989] text-
xs font-light hidden md:inline-block">Delete</span>
                        </div>
```

```
                        <div v-if="user_id === owner" class="inline-block
cursor-pointer">
                            <a :href="`/post/` + id + `/edit`">
                                <EditIcon class="w-4 h-4 inline-block ml-3"/>
                                <span
                                    class="mx-2 inline-block text-[#898989]
text-xs font-light hidden md:inline-block">Edit</span>
                            </a>
                        </div>

                    </div>
                    <div class="right-0 absolute">
                        <span class="text-xs text-[#898989]">{{ date }}</span>
                    </div>
                </div>
                <div class="py-2">

                    <p v-if="!spoiler" class="text-[#898989] pt-2 line-clamp-4
w-full text-sm md:text-base">{{
                            description
                        }}</p>
                    <p class="pt-1 text-white"> Expand post</p>
                </div>
            </div>
            <hr class="text-[#505050] border-[#505050] border-2"/>
        </div>
    </div>
    <div v-if="openCommentSection" class=" border-[#505050] border-2 rounded-lg
p-1 md:p-3 m-10">
        <h3 class=" p-1 text-gray-400">Discussion</h3>
        <div class="flex flex-col gap-5 m-3" v-for="comment in comments"
:key="comment.id">
            <Comment :body="comment.body"
                     :id="comment.id"
                     :date="comment.date"
                     :karma="comment.karma"
                     :replies-number="comment.replies"
                     :post_id="comment.post_id"
                     :vote="comment.vote"
                     :user-name="comment.user?.userName"
                     :owner="comment.user.id"
                     :user_id="user_id"
                     @commentDeleteEmitter="handleDelete"
                     @commentEditEmitter="handleEdit"
            />
        </div>
        <WriteComment :postId="this.id" @commentEmitter="handleComment"/>
    </div>
</template>
```

Воочуваме неколку нови компоненти, некои од нив се икони (VoteDownIcon, VoteUpIcon, EditIcon, DeleteIcon, ShareIcon, CommentIcon), некои од нив се целосно нови компоненти со нови функционалности (Comment и WriteComment).
Бидејќи компонентата е многу обемна, ќе одеме функционалност по функционалност:

1. Гласање на објава
   Треба да може да гласаме на сите објави кои ги гледаме, но само по еднаш, и да дадеме позитивен или негативен глас (лајк или дислајк). За таа цел имаме две состојби (voteUpState и voteDownState) од кои само една или ниедна може да биде активна во даден момент (никогаш двете).

```
created() {
        if (this.vote == null) {
            this.voteUpState = false;
            this.voteDownState = false;
        }
        if (this.vote === true) {
            this.voteUpState = true;
            this.voteDownState = false;
        }
        if (this.vote === false) {
            this.voteDownState = true;
            this.voteUpState = false;
        }
    },
```

Ги дефинираме состојбите во компонентата. Vote ни доаѓа од родителот (дали корисникот претходно има гласано на објавата), па ако е null, значи дека нема глас и двете ги дефинираме на false.

```
async voteUp(postId) {
            this.voteUpState = !this.voteUpState;
            this.voteDownState = false;

            const dto = {
                postId: postId,
                vote: this.voteUpState ? 1 : 0
            }

            try {
                if (dto.vote) {
                    const response = await ApiUtilis.votePost(dto);
                    this.childKarma = response.data;
                } else {
                    const response = await ApiUtilis.deleteVotePost(dto);
                    this.childKarma = response.data;
                }
            } catch (error) {
                console.error('Error fetching options:', error);
            }
        },
```

За да можеме позитивно да гласаме на компонентата, треба на end-point-от да пратиме 1 за позитивен глас, 0 за бришење на позитивниот глас. Со ова го покриваме случајот некој два пати да кликне да копчето за лајк.
По успешно гласање, или бришење на гласот, тоа афектира директно на кармата на објавата, па соодветно од одговорот на серверот ја менуваме таа карма.
Истиот случај е и за негативно гласање со обратни вредности.

2. Коментирање на објавата
   Бидејќи боксот за коментар не ни е отворена по default, ја отвораме со клик на иконата за коментар. Во props на самата компонента праќаме кое е ид-то на објавата. HandleComment event-от го додава ново напишаниот коментар во листата со коментари што ја чуваме за објавата.

```
<writeComment :postId="this.id" @commentEmitter="handleComment"/>
```

3. Листање на коментари
   За секоја објава чуваме листа со коментари и број на коментари. За да активираме вчитување на коментарите (да заштедиме време на вчитување на страницата) треба да кликниме на иконата за коментари.

```
<div class="inline-block">
                            <CommentIcon class="w-4 h-4 inline-block ml-3
cursor-pointer"
                                         @click="this.toggleComments()"/>
                            <span
                                class="mx-2 inline-block text-[#898989] text-xs
font-light hidden md:inline-block">{{
                                     commentsNumber
                                }} Comments</span>
                        </div>
```

За секој коментар вчитуваме:

```
<Comment :body="comment.body"
                    :id="comment.id"
                    :date="comment.date"
                    :karma="comment.karma"
                    :replies-number="comment.replies"
                    :post_id="comment.post_id"
                    :vote="comment.vote"
                    :user-name="comment.user?.userName"
                    :owner="comment.user.id"
                    :user_id="user_id"
                    @commentDeleteEmitter="handleDelete"
                    @commentEditEmitter="handleEdit"
            />
```

# Креирање на "Comments.vue"

За секој пишан коментар треба да чуваме неколку информации. Тоа се:

- id: преку кое ќе пишуваме реплика на коментарот, ќе можеме да го едитираме и бришеме
- датум, корисник од кого е напишан
- дали логираниот корисник има гласано и каков е неговиот глас
- карма на коментарот
- колку реплики има коментарот (кои се тие се моментално исклучени, одлучуваме со клик на копче да ги вчитаме - пак со цел да заштедиме перформанси)

```
data() {
        return {
            writeReply: false,
            replies: [],
            voteUpState: false,
            voteDownState: false,
            childKarma: this.karma,
            editMode: false,
            editBody: '',
            childBody: this.body,
        }
    },
```

Ги поставуваме на иницијални вредности дефинираните состојби. WriteReply се однесува на тоа дали треба да се отвори нов html елемент за да се напише реплика, replies ја иницијализираме на празна листа, состојбите за гласови ги поставуваме на false. ChildKarma се однесува на истата карма која ја праќаме од родителот, но може да биде променета. EditMode служи за да знаеме дали треба да е едитабилен коментарот, и childBody исто се

однесува на телото (едитираното) тело на коментарот.
По креирањето на компонентата, исто како и за објавите, првично ги проверуваме вредностите од родителот.

```
created() {
        if (this.vote == null) {
            this.voteUpState = false;
            this.voteDownState = false;
        }
        if (this.vote === true) {
            this.voteUpState = true;
            this.voteDownState = false;
        }
        if (this.vote === false) {
            this.voteDownState = true;
            this.voteUpState = false;
        }
    },
```

Потоа, за гласање користиме две компоненти:

```
<div class="flex flex-col gap-1 pr-3 py-3">
                <div>
                    <VoteUpIcon
                        class="w-4 h-4 cursor-pointer"
                        :vote-up="this.voteUpState"
                        @click="voteUp(this.id)"
                    />
                </div>
                <div class=" text-gray-500 text-sm text-center">{{
this.childKarma }}</div>
                <div>
                    <VoteDownIcon
                        class="w-4 h-4 cursor-pointer"
                        :vote-down="this.voteDownState"
                        @click="voteDown(this.id)"
                    />
                </div>
            </div>
```

Овие компоненти ја примаат состојабата за секој глас. Ако кликнеме на voteUp копчето, се повикува тој настан и се праќа id-то на коментарот како аргумент.

```
async voteUp(commentId) {
            this.voteUpState = !this.voteUpState;
            this.voteDownState = false;
```

```
            const dto = {
                id: commentId,
                vote: this.voteUpState ? 1 : 0
            }

            try {
                if (dto.vote) {
                    const response = await ApiUtilis.voteComment(dto);
                    this.childKarma = response.data;
                } else {
                    const response = await ApiUtilis.deleteVoteComment(dto);
                    this.childKarma = response.data;
                }
            } catch (error) {
                console.error('Error fetching options:', error);
            }
        },
```

Во оваа функција ги покриваме двата настани. И при создавање на позитивен глас, и при отстранување на истиот. Затоа првично voteUpState го поставуваме на спротивната вредност, а voteDownState ги поставуваме на false. Креираме DTO за повикот кон laravel back-end во зависно што сме притиснале и го праќаме DTO-то во самото тело на барањето. Додека пак како одговор, треба да добиеме ажурирана карма (гласот влијае на кармата на коментарот) и ја менуваме во зависност на тоа (различна е со таа на родителот).

```
 <div class="flex gap-3 ">
                    <button class="text-right text-pink-700"
                            @click="toggleWriteReply()">Reply
                    </button>
                    <button v-if="owner === user_id" class="text-right text-
pink-700"
                            @click="toggleEditMode">Edit
                    </button>
                    <button v-if="owner === user_id" class="text-right text-
pink-700"
                            @click="handleDelete()">Delete
                    </button>
                    <button class="text-right text-pink-700"
                            @click="fetchReplies()"> Replies {{ repliesNumber }}
                    </button>
</div>
```

Имаме копчиња со неколку функционалности кои треба да ги има секој коментар.

- При клик на Reply копчето, повикуваме настан toggleWriteReply() кој ќе го хендла тоа.

```
toggleWriteReply() {
            this.writeReply = !this.writeReply;
        },
```

Тој не прави ништо посебно, само ја менува состојбата во нејзината спротивна вредност, која што веднаш потоа ќе одлучи дали ќе ни се покажи оваа компонента.

```
<WriteComment v-if="writeReply" @commentEmitter="handleReply"
                              :parent-id="this.id"/>
```

```
 data() {
        return {
            body: '',
        }
    },
    props: {
        postId : null,
        parentId: null
    },
```

Оваа компонента служи за да пишуваме реплики на коментари и на објави. За таа цел во неа треба да чуваме два идентификатори (кој е родителот на објавата и кој е родителот на коментарот). Некоја од овие вредности не треба да е null, бидејќи така идентификуваме каде е напишан коментарот.
Body е состојба во рамки на самата компонента, додека props доаѓаат од родителот. (Пред да се прикаже оваа комонента треба да има id)

```
<template>
    <div class="w-full px-3 mb-2 mt-6">
        <textarea
            class="bg-neutral-800 rounded border border-gray-400 text-gray-50
            leading-normal resize-none w-full h-20 py-2 px-3 font-medium
            placeholder-gray-400 focus:outline-none focus:bg-neutral-800"

            name="body" v-model="body" placeholder="Comment" required>
</textarea>
    </div>

    <div class="w-full flex justify-end px-3 my-3">
        <input type="button" @click="writeComment()"
               class="bg-blend-darken px-2.5 py-1.5 rounded-md text-white text-
sm bg-pink-700 text-lg"
               value='Post Comment'>
```

```
    </div>
</template>
```

WriteComment компонентата не содржи ништо посебно, освем поле за текст каде што може да внесеме содржина на коментарот и копче за submit.
При клик се активира writeComment() методот.

```
async writeComment() {

            const commentCreationDto = {
                post_id: this.postId,
                parent_comment_id: this.parentId,
                body: this.body
            }

            try {
                const response = await
ApiUtilis.writeComment(commentCreationDto);
                this.emitToParent(response.data);
                this.body = '';
            } catch (error) {
                console.error('Error fetching options:', error);
            }
        },
```

DTO-то кое треба да го пратиме на endpoint-от содржи полиња за двете id и телото на коментарот. Како одговор од Laravel, треба да го добиеме самиот коментар доколку повикот е успешен и да го пратиме(емитнеме) на родителот. Телото на самиот коментар го чистеме.

```
handleReply(data) {
            this.replies.push(data[0]);
        },
```

Во родителот (Comment) само добиениот одговор го додаваме на листата со коментари, за да се ажурира со таа на back-end.

```
async handleDelete() {
            try {
                const response = await ApiUtilis.deleteComment(this.id);
                this.emitDeleteToParent(this.id);
            } catch (error) {
                console.error('Error fetching options:', error);
            }
        },
```

За да избришеме коментар, тоа можеме прво да го пристапиме во самата Comment компонента и се повикува настанот handleDelete(). Се праќа повик за бришење коментар со неговото id, и ако бришењето е успешно, id-то на избришаниот коментар го емитуваме на родителот (Post компонента).

```
handleDelete(id) {
            const index = this.comments.findIndex(comment => comment.id === id);
            if (index !== -1) {
                this.comments.splice(index, 1);
            }
        },
```

Го бараме индексот на избришаниот коментар и го бришеме од листата со коментари.

```
<div v-for="reply in replies" :key="reply.id">
            <div class="text-[#505050] font-bold pl-14">|</div>
            <Comment class="ml-5"
                     :body="reply.body"
                     :id="reply.id"
                     :date="reply.date"
                     :karma="reply.karma"
                     :parent_comment_id="this.id"
                     :replies-number="reply.replies"
                     :vote="reply.vote"
                     :user-name="reply.user?.userName"
                     :owner="owner"
                     :user_id="user_id"
                     @commentDeleteEmitter="handleDeleteReply(reply.id)"
                     @commentEditEmitter="handleEditReply($event)"
            />
```

За секој коментар листаме негови реплики и повикуваме скоро исти функции, но врз id-то на самата реплика.

```
async handleDeleteReply(id) {
            try {
                const response = await ApiUtilis.deleteComment(id);
                const index = this.replies.findIndex(reply => reply.id === id);
                if (index !== -1) {
                    this.replies.splice(index, 1);
                }
            } catch (error) {
                console.error('Error fetching options:', error);
            }
        },
```

Го праќаме id-то на репликата, и правиме повик за бришење до back-end. Ако тоа е успешно, се бара индексот на избришаниот коментар од листата со реплики во Comment, и се брише.

```
                <p class="text-gray-400 mt-2" v-if="!editMode">
                    {{ childBody }}
                </p>
                <textarea v-if="this.editMode"
                          class="bg-neutral-800 rounded border border-gray-400
text-gray-50
            leading-normal resize-none w-full h-20 py-2 px-3 font-medium
            placeholder-gray-400 focus:outline-none focus:bg-neutral-800" v-
model="this.editBody"
                          name="body" placeholder="Comment" required>
{{this.body}}</textarea>
```

За секој коментар чуваме „скриен“ дел за едитирање кој при клик се заменува со постоечкиот и се пополнува автоматски со постоечката вредност. Ако не е во editMode го прикажуваме childBody кој има копирана вредност како и родителот, а при клик на edit, се прикажува body кој ја има директно вредноста на родителот. Зошто е ова вака? Бидејќи при промена на вредноста на телото сакаме да се рефлектира и во детето и во родителот на компонентата.

```
 <div class="w-full flex justify-end px-3 my-3">
                    <input v-if="this.editMode" type="button"
@click="editComment()"
                           class="bg-blend-darken px-2.5 py-1.5 rounded-md text-
white text-sm bg-pink-700 text-lg"
                           value='Edit Comment'>
</div>
```

При клик се повикува editComment() методот.

```
async editComment() {

            const commentUpdateDto = {
                id: this.id,
                body: this.editBody
            }

            try {
                const response = await ApiUtilis.editComment(this.id,
commentUpdateDto);
                this.childBody = response.data.body;
                this.editMode = false;
                this.emitEditToParent(commentUpdateDto);
            } catch (error) {
```

```
                console.error('Error fetching options:', error);
            }
        },
```

Правиме соодветно DTO со идентификаторот на коментарот и неговата нова содржина, и правиме повик до backend. При успешна промена, го променуваме childBody (не е повеќе копија од родителот), го затвораме editMode-от и ја емитуваме промената до родителот. (Post)

```
handleEditReply(replyDto) {
            const index = this.replies.findIndex(reply => reply.id ===
replyDto.id);
            this.replies[index].body = replyDto.body;
        },
```

За да едитираме реплика на некој коментар, го пронаоѓаме тој коментар во листата со реплики на коментарот, и соодветно го менуваме неговото тело со новото.

# Креирање на страница за заедница

За да креираме Community page соодветно имаме неколку компоненти кои ќе ги реискористиме (Navbar, Content и Create).
Меѓутоа треба да направиме нови за Flairs (категории кои ги има секоја заедница) и основни информации (Rules, AboutCard и Moderators)

```
<template>
    <Head title="Community"><title>{{this.community.name}}/c</title></Head>
    <Navbar/>
    <div id="left" class="absolute mt-32 md:mt-20 w-auto md:w-1/6 ml-5">
        <Filter class="rounded-xl" @sort="emitSortType($event)"/>
        <Flairs class="my-3 hidden md:block" :flairs="community.flairs"
                @selectedFlairEmitter="handleFlairFilter($event)"/>
    </div>
    <div id="right" class="absolute mt-10 md:mt-20 w-auto md:w-1/6 md:right-0
mx-5">
        <Create class="rounded-xl"/>
        <AboutCard class="my-3 hidden md:block"
                   :about="community.about"
        />
        <Rules class="hidden md:block"
               :rules="community.rules"
        />
        <Moderators class="my-3 hidden md:block"
                    :community="community"
        />
    </div>
```

```
    <Content :type="'community'" :sort="this.sort" :flair="this.flairFilter"/>
</template>
```

За да дознаеме која заедница моментално е отворена го извршуваме следниот повик:

```
async fetchData() {
            const regexPattern = /[^/]+$/;

            const path = window.location.pathname;
            const match = path.match(regexPattern);

            if (match) {
                const result = match[0];
                try {
                    let response = await ApiUtilis.fetchCommunity(result);
                    this.community = response.data[0];
                } catch (error) {
                    console.error('Error fetching data:', error);
                }
            }
        },
```

Патеката е community/{id}. Моменталниот регекс ја зема вредноста после последното "/" и проверува дали се совпаѓа. Ако е така се праќа повик до back-end за да го земе основниот Community објект и се чува соодветно во самата компонента-страница.

```
<template>
    <div class="py-2 md:py-5 bg-[#2D2D2D] rounded-xl">
        <div class="flex justify-start px-3" v-for="flair in flairs">
            <div class="p-2 text-white font-light text-sm rounded bg-[#505050]
m-1 cursor-pointer"
                 :class="{'bg-[#CC0974]' : selectedFlair === flair.name}"
                 @click="filter(flair)">{{ flair.name }}
            </div>
        </div>
    </div>
</template>
```

Компонентата за категории ги листа постоечките кои се пратени на props и прави филтрирање по истите.

```
filter(flair) {
            if (this.selectedFlair !== '') {
                this.selectedFlair = '';
                this.$emit('selectedFlairEmitter', '');
            } else {
                this.selectedFlair = flair.name;
                this.$emit('selectedFlairEmitter', flair.id);
            }
        }
```

Во сценариото каде што кликаме и селектираме некоја категорија, следно треба да ја ажурираме содржината во Content компонентата што ја соджи таа категорија.
Па така id-то на селектираната категорија го праќаме до родител (Community компонентата), па од таму ќе се прати на Content компонентата. Во спотивно праќаме празна категорија.

```
data() {
        return {
            community: Object,
            sort: 'new',
            flairFilter: ''
        }
    },
```

Во Community чуваме податоци и за која категорија е одбрана (default е никоја или празна) и чуваме default вредност за сорт (по нови). При одбирање на некоја категорија, flairFilter го поставуваме и го праќаме на Content.

```
<Content :type="'community'" :sort="this.sort" :flair="this.flairFilter"/>
```

# Креирање на страница за профил

Повторно реискористуваме компоненти.

```
<template>
    <Head :title="user[0].name"></Head>
    <Navbar :user="user"/>
    <div id="left" class="absolute mt-32 md:mt-20  w-auto md:w-1/6 ml-5">
        <Filter class="rounded-xl" @sort="emitSortType($event)"/>
    </div>
    <div id="right" class="absolute mt-10 md:mt-20  w-auto md:w-1/6 md:right-0
mx-5">
        <ProfileCard :user="user[0]"/>
    </div>
    <Content :type="'user'" :sort="this.sort"/>
</template>
```

Прикажуваме наслов на страната со корисничкото име на корисникот. На навигацискиот бар го праќаме повторно. Опцијата за сортирање ни стои повторно и за содржина исто така. ProfileCard е нова компонента која ни ги прикажува корисникот со неговата профилна и насловна фотографија.

```
<template>
    <div>
        <ProfileBacgkroundPicture :image="user.info.image_id"/>
        <div class="bg-[#2d2d2d] rounded-lg shadow-lg p-6">
            <div class="flex items-center">
                <ProfilePicture :image="user.info.image_id"/>
                <div class="ml-4">
                    <h2 class="text-lg font-semibold" style="color: #F20085">{{
user.name }}</h2>
                    <p class="text-gray-200">{{ user.karma.karma }} Karma</p>
                </div>
            </div>
            <div class="mt-4">
                <p class="text-gray-200">{{ user.info.bio }}</p>
            </div>
            <br>
            <div class="flex justify-between mt-6">
                <div v-if="user.status.active">
                    <p class="text-sm font-semibold text-green-500">Online</p>
                </div>
                <div v-else>
                    <p class="text-sm font-semibold text-red-500">Offline</p>
                </div>
            </div>
        </div>
    </div>
</template>
```

Во оваа компонента прикажуваме и дали корисникот чиј профил го гледаме е активен или не. Меѓутоа на останатите ProfileBacgkroundPicture и ProfilePicture праќаме id. Имаме две различни компоненти бидејќи имаме предифинирана вредност за слика на секоја од нив.

```
export default {
    props: {
        image: Number
    },
    name: 'ProfileBacgkroundPicture',
    data() {
        return {
            imageUrl: 'https://imgv3.fotor.com/images/blog-cover-image/part-
blurry-image.jpg',
        };
    },
    methods: {
        fetchImage(id) {
            ApiUtils.fetchImage(id)
                .then((imagePath) => {
                    this.imageUrl = imagePath;
                })
                .catch((error) => {
                    console.error('Error fetching image:', error);
                });
        },
    },
    mounted() {
        this.fetchImage(this.image);
    },
};
```

```
export default {
    props: {
        image: Number
    },
    name: 'ProfilePicture',
    data() {
        return {
            imageUrl: 'https://imgv3.fotor.com/images/blog-cover-image/part-
blurry-image.jpg',
        };
    },
    methods: {
        fetchImage(id) {
            ApiUtils.fetchImage(id)
                .then((imagePath) => {
                    this.imageUrl = imagePath;
                })
                .catch((error) => {
                    console.error('Error fetching image:', error);
                });
        },
    },
```

```
    mounted() {
        this.fetchImage(this.image);
    },
};
```

За да се добијат сликите се праќа повик за да се добие нивно URL или патека. Потоа таа се праќа во src на image.