Laravel
Lab - 2

PHP WebDevelopment 2019

Milena Tomova
Vratsa Software

https://vratsasoftware.com/

Table of Contents

Learn to Search in Internet

- The course assignments require to search in Internet
  - This is an important part of the learning process
  - Some exercises intentionally have no hints
- Learn to find solutions!
  - Software development includes everyday searching and learning
  - No excuses, just learn to study!
  - Developers learn new technologies, tools, languages every day!

Working with files

Task 1

Allow users to add their photo in their profile page

- the photo can be also changed and deleted
- make sure that after the photo has been replaced or deleted it is removed from the host storage as well

Working with files

Working with files

- Laravel provides a powerful filesystem abstraction thanks to the wonderful Flysystem PHP package by Frank de Jonge.

- The filesystem configuration file is located at **config/filesystems.php**.

By default Laravel is configured to use local driver

'default' => env('FILESYSTEM_DRIVER', 'local')

Working with files

Public Disk

The public disk is intended for files that are going to be **publicly accessible.**

from *config/filesystems.php*

```
    'disks' => [
    ...
    'public' => [
    'driver' => 'local',
    'root' => storage_path('app/public'),
    'url' => env('APP_URL').'/storage',
    'visibility' => 'public',
    ],
...
    ],
```

Working with files

Public Disk

By default,

the **public disk** uses the **local driver** and

stores these files in **storage/app/public**.

To make them accessible from the web,

you should **create a symbolic link** from **public/storage** to **storage/app/public**.

This convention will keep your publicly accessible files in one directory that can be easily shared across deployments.

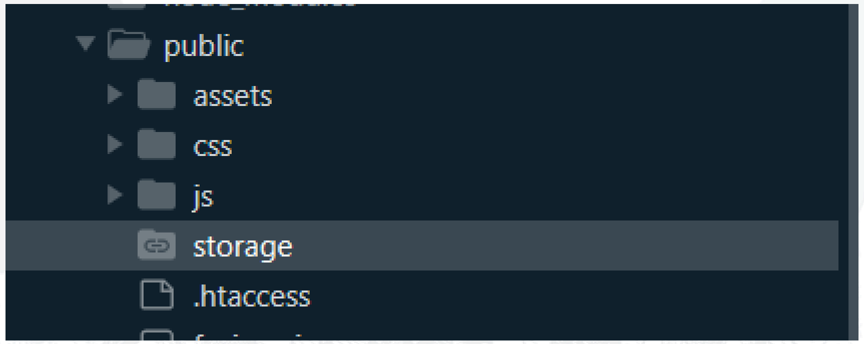
Working with files

To create the symbolic link

```
php artisan storage:link
```

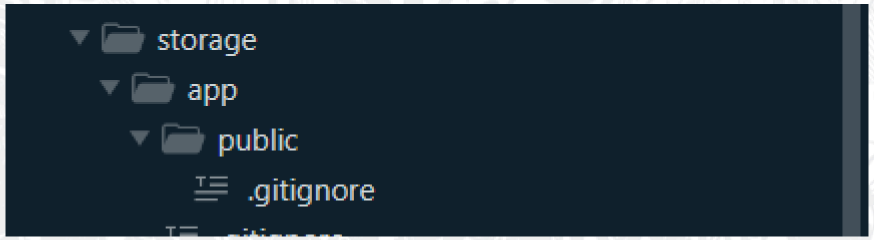
Working with files

the command created new **‘folder’** in public directory



```
▼ public
  ▶ assets
  ▶ css
  ▶ js
    storage
    .htaccess
```

and a **public** folder in **storage/app** path



```
▼ storage
  ▼ app
    ▼ public
        .gitignore
```

Working with files

Once a file has been stored and the symbolic link has been created, you can create a URL to the files using the asset helper function:

```
echo asset('storage/my_file.txt')
```

Working with files

To configure additional symbolic links in your filesystems configuration file - add them in config/filesystem.php

```
 'links' => [
     public_path('storage') => storage_path('app/public'),
     public_path('images') => storage_path('app/images'),
],
```

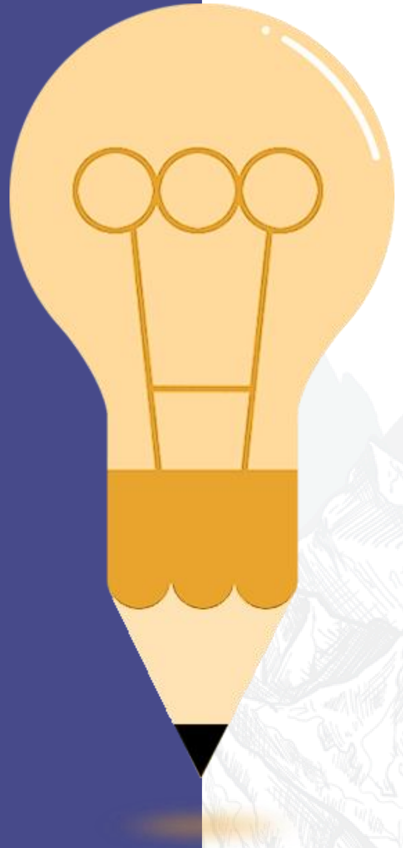
Each of the configured links **will be created** when you run the **storage:link** command:

Upload files

To upload files via forms

Do not forget to -

- set the form **enctype** to **multipart/form-data**
- add **input** type **file**



Upload files

```
{!! Form::open(['action' => 'UsersController@store',
                 'method' => 'POST',
                 'enctype' => 'multipart/form-data'
                 ]) !!}
```

```
{!! Form::file('stock_image') !!}
```

Working with files

Storing files

documentation

The **put** method may be used to store raw file contents on a disk.

Remember, **all file paths** should be specified relative to the **"root" location** configured for the disk:

Working with files

To **create** a file from a **raw content** with a name set at the backend

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


Storage::put('file.jpg', $contents);
```

Working with files

Store a file from the request,

file input name in the form

```
$path = $request->file('image')->store('user_images');
```

the **filename**, the file is stored under, is generated **automatically** and is **unique**

user_images/r5bB9OK7LU42WEbkgZkO8joQQXisj0tHtSN5P1NB.png

store() parameter

folder with this name is created in the storage/app

Working with files

Врацa Софтуер Общество

Store a file from the request, using Storage Facade

```
$path = Storage::putFile('user_images', $request->file('image'));
```

Working with files

Set a filename for the file and store the file with that name.

```
$path = $request->file('image')
      ->storeAs('user_images', $request->user()->id . '.png');
```

The new image will be stored **under same name**, so there be no need to take care for orphan images in the storage. Updated user image **will overwrite** the old one.

Working with files

In case you need to store the file **with its original name** -

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

$name = $request->file('image')->getClientOriginalName();
```

Then use **$name** to save it in the data base or for other logic.

Working with files

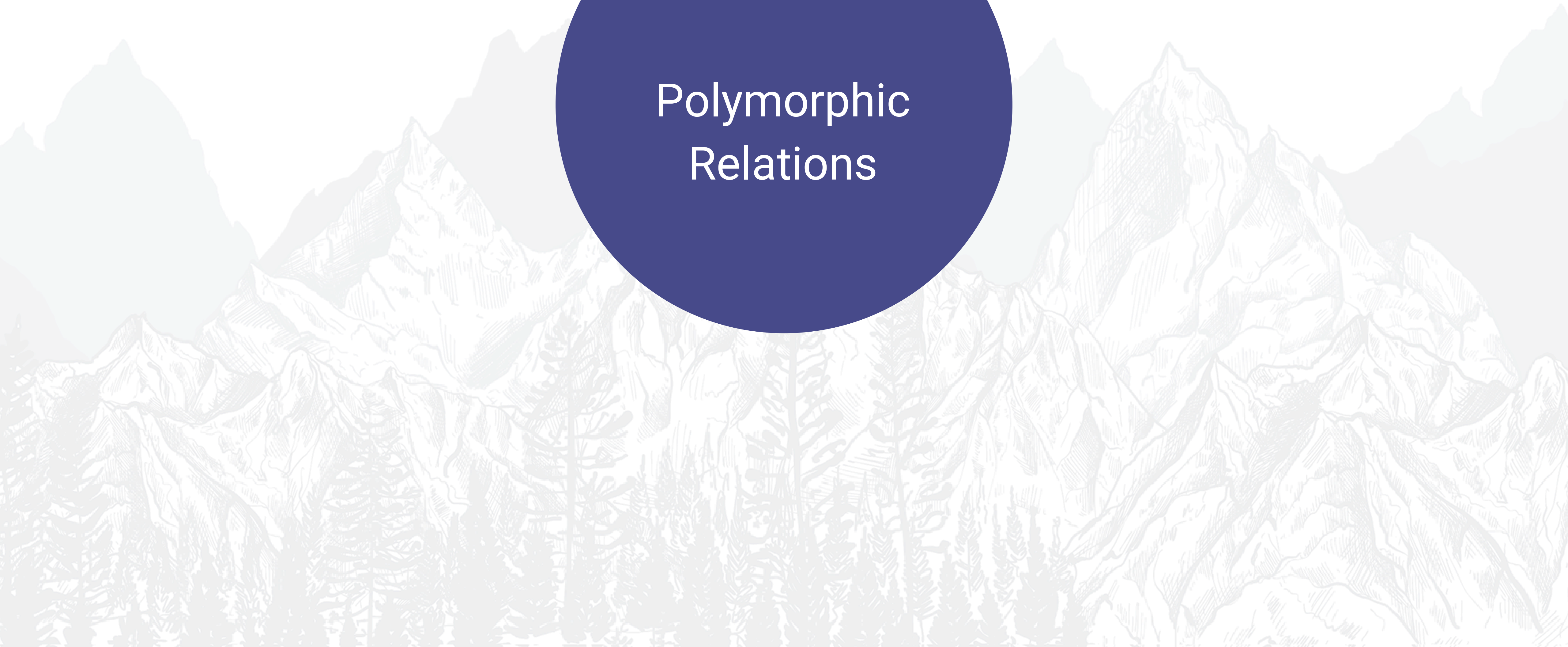
To remove a file from the storage - you need to know its name and the folder it resides in the **storage/app**

```
 use Illuminate\Support\Facades\Storage;
//delete single file from storage
Storage::delete('folder/filename.jpg');
//delete multiple files from storage
Storage::delete(['folder/filename.jpg',
                 'folder/filename2.jpg']);
```

Task 2

Allow admins to add photo for

- **halls**
- **courses**
  - the photo is to be changed and deleted
  - make sure that after the photo is being replaced or deleted it is removed from the host storage as well



Polymorphic Relations

Polymorphic Relations

one-to-one

Polymorphic

Relations

documentation

A polymorphic relationship allows the target model to belong to more than one type of model using a single association.

- **users** can have pictures
- **halls** can have pictures

We can solve this by creating an Image model and write all pictures in an images table in the Data Base.

Polymorphic Relations

Polymorphic Relations

documentation

Why to create a separate Image model?

We allow users to upload images and may be these images will have their behaviour -

- must be cropped
- resized
- etc

One to one polymorphic relations

table structure

```
 halls
     id - integer
     name - string
users
     id - integer
     name - string
```

```
images
     id - integer
     url - string
     imageable_id - integer
     imageable_type - string
```

One to one polymorphic relations

Models structure -

the shared **Image** model

```
 class Image extends Model
{
    /**
     * Get the owning imageable model.
     */
    public function imageable()
    {
        return $this->morphTo();
    }
}
```

One to one polymorphic relations

Models structure -

```
 class User extends Model
{
    /**
     * Get the user's image.
     */
    public function image()
    {
        return $this->morphOne('App\Models\Image', 'imageable');
    }
}
```

One to one polymorphic relations

Models structure -

```
 class Hall extends Model

{

    /**

     * Get the hall's image.

     */

    public function image()

    {

        return $this->morphOne('App\Models\Image', 'imageable');

    }

}
```

One to one polymorphic relations

Access the image model instance

```
$user = App\User::find($id);

$user_photo = $user->image;
```

```
//the parent (owner) of the comments

$image = App\Image::find($id);

$image_owner = $image->imageable;
```

```
$hall = App\Hall::find($id);

$hall_photo = $hall->image;
```

One to one polymorphic relations

Access the image model instance

```
$homework = App\Models\Homework::find(1);

foreach ($homework->comments as $comment)
{

    //access all homework`s comments

}
```

```
//the parent (owner) of the comments

$comment = App\Models\Comment::find(1);

$commentable = $comment->commentable;
```

```
$event = App\Models\Event::find(1);

foreach ($event->comments as $comment) {

    //access all homework`s comments

}
```

Polymorphic Relations

one-to-many Polymorphic Relations

documentation

A one-to-many polymorphic relation is similar to a simple one-to-many relation; however, the target model can belong to more than one type of model on a single association.

Polymorphic Relations

one-to-many

Polymorphic Relations

documentation

For example -

- each homework can have many comments
- each event can have many comments

One to many polymorphic relations

table structure

```
 homeworks

     id - integer

     filename - string

...

events

     id - integer

     title – string

...
```

```
comments

     id - integer

     body - text

     commentable_id - integer

     commentable_type – string

     user_id  - integer
```

One to many polymorphic relations

Models structure - the shared Comments model

```
 class Comment extends Model
{
    /**
     * Get the owning commentable model.
     */

    public function commentable()
    {
        return $this->morphTo();
    }
}
```

One to many polymorphic relations

Врацa Софтуер Общество

Models structure -

```
class Homework extends Model

{

    /**

     * Get all of the homeworks's comments.

     */

    public function comments()

    {

        return $this->morphMany('App\Models\Comment', 'commentable');

    }

}
```

One to many polymorphic relations

Models structure -

```
 class Event extends Model
{
    /**
     * Get all of the event's comments.
     */
    public function comments()
    {
        return $this->morphMany('App\Models\Comment', 'commentable');
    }
}
```

One to one polymorphic relations

Access

the comment

model

instance

```
$homework = App\Homework::find(1);

foreach ($homework->comments as $comment) {

 // access homework comments

}
```

```
$event = App\Event::find(1);

foreach ($event->comments as $comment) {

 // access event comments

}
```

```
//the parent (owner) of the comment

$comment = App\Models\Comment::find($id);
$commentable = $comment->commentable;
```

Insert polymorphic relations data

Insert polymorphic relations data

Враца Софтуер Общество

using

save()

helper

```
$comment = new App\Comment(['body' => $request->body,
                            //other comment data ]);
$event = App\Event::find($event->id);
$comment = $event->comments()->save($comment);
```

```
$homework = App\Homework::find($homework->id);
$homework->comments()->create([ ‘body' => $request->body,
                                  //other comment data
                                  'commentable_id' => $homework->id,
                          'commentable_type' =>get_class($homework)]);
```

using

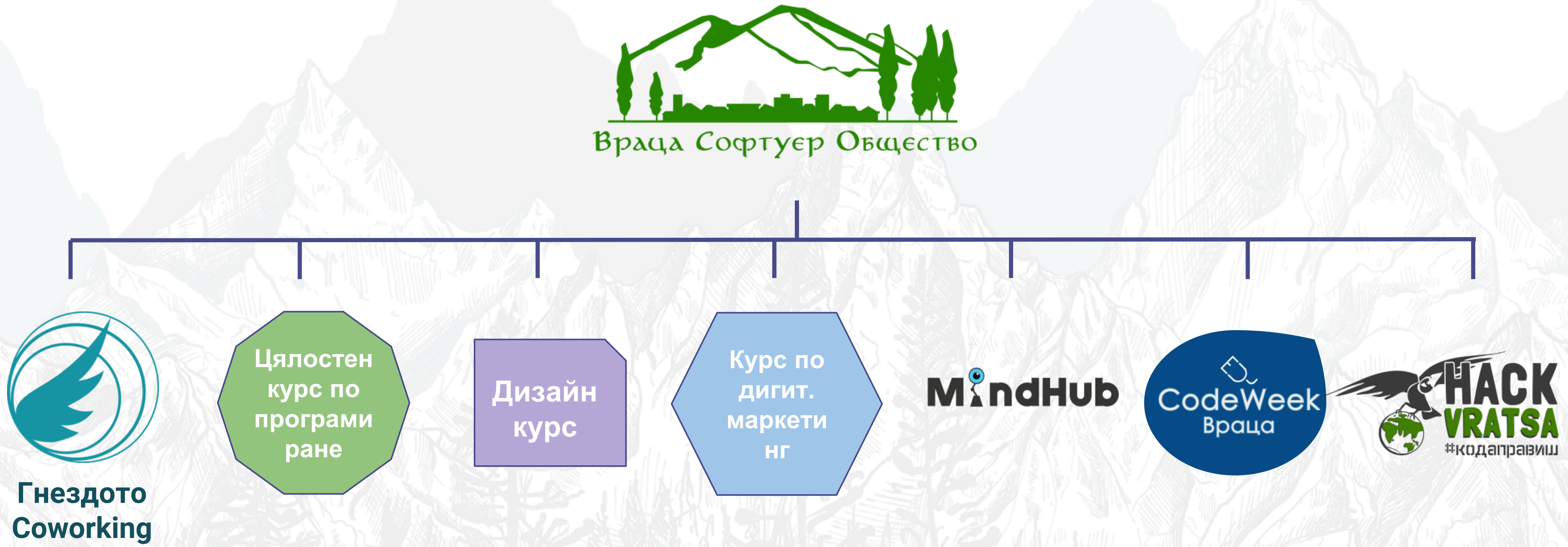
create()

helper

Polymorphic Relations

For more information on polymorphic relations, please see the documentation.

Questions?


Враца Софтуер Общество
Гнездото Coworking
Цялостен курс по програмиране
Дизайн курс
Курс по дигит. маркетинг
MindHub
CodeWeek Враца
HACK VRATSA
#кодаправиш

Partners

AMERICA FOR BULGARIA FOUNDATION

ВРАЦА

Telerik Academy

НАЦИОНАЛНА МРЕЖА ЗА ДЕЦАТА

MindHub

ПРОМЯНАТА NOVA

Trainings @ Vratsa Software

- Vratsa Software – High-Quality Education, Profession and Jobs
  - www.vratsasoftware.com
- The Nest Coworking
  - www.gnezdoto.vratsasoftware.com
- Vratsa Software @ Facebook
  - www.fb.com/VratsaSoftware
- Slack Channel
  - www.vso.slack.com