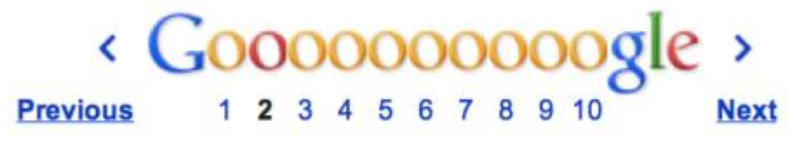
Pagination, Date and Time, and Imaging Handling

http://www.mypocketwatches.org/wp-content/uploads/2012/02/Pocket-Watches-2.jpg

Pagination

- Web applications often store a lot of data.
- Generally we only need to return and display a small portion of the data for each page.
- For example, if a database contained 10,000 items and a search resulted in 1,000 results, generally the user doesn't want all of those results in the one page.
- Returning all results will use extra bandwidth, require more time for the page to load, and require a lot of scrolling by the user.
- The typical solution is to paginate the results.
- Only a smaller number of results are returned per page, possibly no more than will fit on the height of a typical screen (e.g. 30 results).
- A pagination bar is displayed allowing the user to navigate to the:
  - Next page of results
  - Previous page of results
  - Last page of results
  - First page of results
- The pagination bar may also display a list of page numbers which the user can click to go directly to a page.
- The pagination bar should also indicate the current page by highlighting the page number and ensuring it is not linked.

- Google's pagination bar on their results page is a good example:



Limiting Results

- In the Database lecture we covered how to limit the number of results returned from an SQL query and also the starting offset.
- We will use this feature to return a page of results:

```
SELECT *
FROM products
LIMIT 0, 30
```

- Returns the first 30 products.

```
SELECT *
FROM products
LIMIT 30, 30
```

- Returns the next 30 products.
- The number of results per page will either be fixed or could also be set by the user.
- The starting offset will depend on the currently selected page number multiplied by the number of results.
- For example:

```
$page = 0;
$items_per_page = 30;
$offset = $page * $items_per_page;
```

Laravel Offset and Limit

- Laravel also supports offset and limit on queries, for example:

```
$products = Product::query()->offset(30)->limit(30)->get();
```

- Instead of using the functions, **offset()** and **limit()**, you can also use skip() and take() which are equivalent, e.g:

```
$products = Product::query()->skip(30)->take(30)->get();
```

Determining the number of pages

- The number of pages of results is the total number of results divided by the number of results per page, rounded up:

```
$total_pages = ceil($num_items / $items_per_page);
```

- We can use the count() function to determine the total number of results, e.g.:

```
$num_items = Product::query()->count();
```

Displaying the pagination bar

- The pagination bar will be a list of numbered links to the results page indicating the page number or offset as parameter, e.g:

```
@for($page=1; $page <= ceil($num_items/$items_per_page); $page++}
   <a href="item_list.php?page=$page">{{ $page }}</a>
@endfor
```

- item_list.php in this case must know how to utilise $page by converting it to an offset:

```
$offset = ($page - 1) * $items_per_page;
```

- Note that the page numbers displayed are from 1..n however we need to have page numbers from 0..n-1 to calculate the correct offset.
- Note that the $page can also be used to conditionally set whether the next and previous links are active (by checking whether we are on the first or last page).
- Generally the currently viewed page number's link is not active. This can be achieved by testing in the loop if the loop page number is equal to the page number passed to the PHP script.

Laravel Pagination

- Laravel simplifies pagination greatly by providing two functions:
  - **paginate()** - paginates the results
  - **links()** - displays HTML links to each page
- The paginate() function can be called on either:
  - Query (builder) objects, or

```
$products = DB::table('products')->paginate(10);
```

  - Eloquent queries

```
$products = Product::paginate(10);
```

- The above queries will return 10 products.
- How does the paginate() function know what the offset is?
- The paginate() function assumes that a page URL variable has been passed into the request. E.g.: https://.../prod/public/?**page=2**
- We can use the **links()** function in our **view** to generate the paginated links, e.g.:

```
{{ $products->links()}}
```

- That's it!
- Ok, not quite. Pagination does NOT look decent without styling.

Pagination styling

- By default, Laravel uses Tailwind CSS for styling for pagination.
- **Tailwind** ships with Laravel, in **app.css**. To include it, add the following to your HTML/layout:

```
<link href="{{ asset('css/app.css') }}" rel="stylesheet">
```

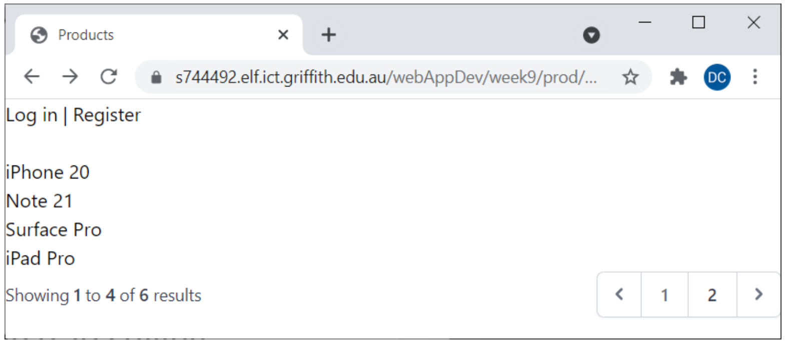
- Laravel also support **Bootstrap** styling for pagination. To use bootstrap styling in addition to include Bootstrap CSS in HTML/layout, the following code needs to be added to boot() of **App\Providers\AppServiceProvider**:

```
use Illuminate\Pagination\Paginator;

public function boot()
{
   Paginator::useBootstrap();
}
```

- Note: you should pick either Tailwind or Bootstrap, not both.

- A screen shot of pagination using Tailwind without any other custom styling:


Products

s744492.elf.ict.griffith.edu.au/webAppDev/week9/prod/...

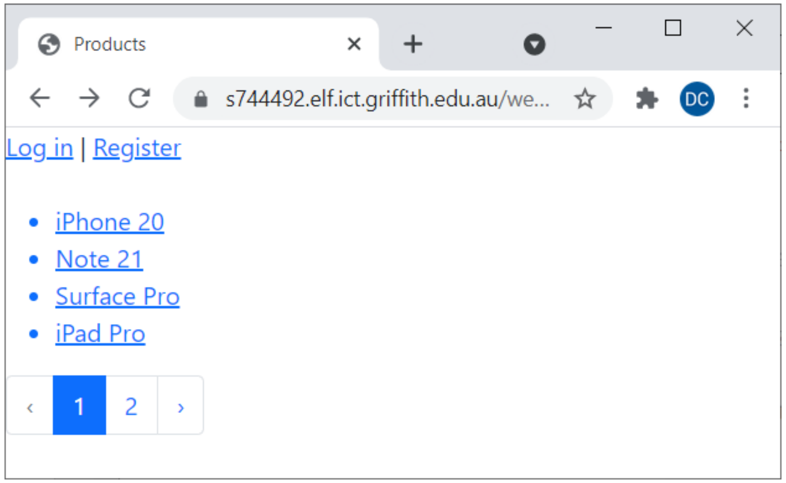
Log in | Register

iPhone 20
Note 21
Surface Pro
iPad Pro

Showing **1** to **4** of **6** results

‹ 1 2 ›

- A screen shot of pagination using Bootstrap without any other custom styling:


Products

s744492.elf.ict.griffith.edu.au/we...

Log in | Register

- iPhone 20
- Note 21
- Surface Pro
- iPad Pro

‹ 1 2 ›

Dates and Times

- Dates and times can be more complicated than they first appear.
- We need to consider that PHP and SQL may have different representations of dates and times.
- We also need to consider the time zone of the server and user.

SQLite Date Representations

- SQLite represents timestamps using the following format:

  YYYY-MM-DD HH:MM:SS

- For example the timestamp 9:30pm on 15 May 2012 will be represented as:

```
2012-05-15 21:30:00
```

PHP Date Representations

- PHP represents dates as Unix timestamps.
- Unix timestamps are represented as the number of seconds since 1 Jan 1970.
- The timestamp above represented as a Unix timestamp would be:

```
1337074200
```

- Useful website for converting Unix timestamps and reference for common functions in various languages:

  http://www.epochconverter.com/

Getting the current time in SQL

- The function **DATETIME()** can be used in SQLite to get the current time in SQL format.
- It can be used when inserting data, comparing dates/times, and also to simply retrieve the current time, e.g:

```
SELECT DATETIME();
```

Getting the current time in PHP

- The time() function can be used to get the current Unix timestamp, e.g:

```
$time = time();
```

Converting between Unix and SQLite timestamp representations

- The conversion from and to Unix timestamps generally happens on the SQLite end using the functions:
  - **datetime(time, 'unixepoch')** - Converts a Unix timestamp to a SQL timestamp.
  - **strftime('%s', time)** - Converts a SQLite timestamp to a Unix timestamp.

Laravel Dates and Times

- Eloquent will automatically convert between SQL datetime format and Unix timestamps by using the **Carbon** class which is a subclass of the PHP **DateTime** class.
- By default the **created_at** and **updated_at** columns are **Carbon** objects.
- You can specify that other datetime columns are also represented using Carbon objects by implementing the **getDates()** method in the model class and returning an array of columns which should be represented using Carbon objects, e.g.:

```
function getDates() {
   return array('created_at',
'updated_at', 'start_date',
'final_date');
}
```

PHP DateTime class

- Creating an instance of the DateTime class returns a DateTime object representing the current time:

```
$now = new DateTime;
```

- You can also pass in a string to the constructor to be parsed.
- The DateTime parser is very flexible, below is an example of some of the strings it will parse:
    - "now"
    - "tomorrow"
    - "+2 days"
    - "1 week" (from now)
    - "20 May" (this year)
    - "20 May 2012"

- "2012-5-20“
- "20 May 2003 8:30pm“
- "tomorrow 22:00"

- A Unix timestamp can be passed in by prepending with the "@"symbol, e.g.:

```
$time = new DateTime('@1337074200');
```

- The Unix timestamp can also be set using **setTimestamp()**:

```
$time->setTimestamp(1337074200);
```

- Use the **getTimestamp()** function to get the Unix timestamp:

```
$timestamp = $time->getTimestamp();
```

The date can be set using setDate(year, month, day):

```
$time->setDate(2014, 5, 20);
```

- The time can be set using setTime(hour, minute, second):

```
$time->setTime(11, 30, 2);
```

Using DateTime in a Database Seeder

- As long as the column is listed in the array returned by the getDates() method you can assign a DateTime object to a column:

```
$job->start_date = new DateTime('6 May 2014');
```

- Additionally you may also assign a string but it must be in the form 'Y-m-d', e.g:

```
$job->start_date = '2014-05-06';
```

DateTime output

- DateTime can be converted to a string using the **format()** function, e.g.:

```
$time = new DateTime;
$string = $time->format("H:i:s");
echo $string;
```

- Will output the current time in the format: HH:MM:SS

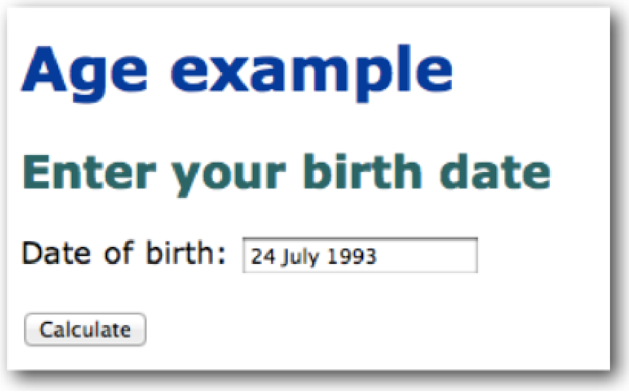
- See the PHP date() function documentation for a detailed list of formatting options:

- http://www.php.net/manual/en/function.date.php

Example format strings:

FORMAT STRING	EXAMPLE
"F j, Y, g:i a"	March 10,2001, 5:16pm
"m.d.y"	03.10.01
"j, n, Y"	10, 3, 2001
"Ymd"	20010310
"D"	Mon
"l" (lowercase 'L')	Monday

Date calculations in PHP


Age example

Enter your birth date

Date of birth: 24 July 1993

- Date time calculations can be tricky.
- Take the following case:
  - Given a birth date and the current date, how old is the person?
- The age example demonstrates the problem:

http://www.ict.griffith.edu.au/teaching/WP/Examples/age-calc/

- First the age is input by the user

- The date is passed as a GET parameter to the same index.php script:

```
index.php?date=24+July+1993
```

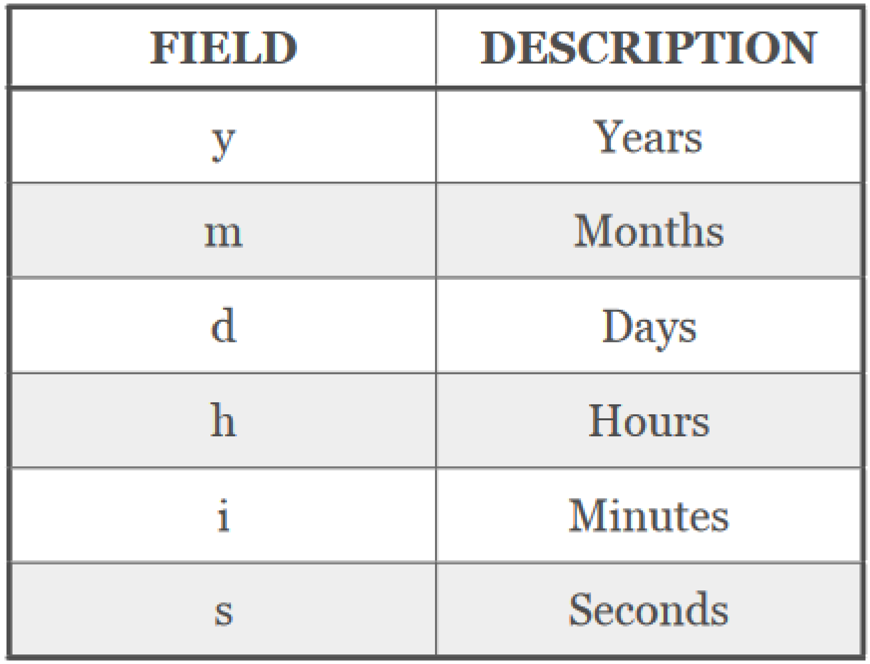
- A DateTime object is created from the date string:

```
$birth_date = new DateTime($date);
```

- Next we need to find the difference between the date entered and the current date.
- The **diff()** function calculates the difference between two date times.

```
$now_date = new DateTime;
$interval = $birth_date->diff($now_date); // $interval is a DateInterval
```

- **diff()** returns a **DateInterval** object which has the following fields:



FIELD	DESCRIPTION
y	Years
m	Months
d	Days
h	Hours
i	Minutes
s	Seconds

- These fields can be accessed to work out the interval between the two DateTime objects:

```
echo "{$interval->y} years, {$interval->m} months, {$interval->d} days.";
```

File/Image Upload

- Many websites need to be able to handle uploaded files.
- Examples include user profile pictures, product photos, and user provided photos.
- There are several steps in uploading and accessing uploaded files:
    - Provide a file uploading form.
    - Retrieve uploaded file details in the controller.
    - Move the file to the storage location.
    - Store the file meta data in the database, i.e. what is the filename, and where it is stored.
    - Allow the file to be accessed.
- We will now cover these mandatory steps in more detail.

How to store uploaded files?

- Files can be stored directly in a database using the binary BLOB data type.
- However files are generally quite large which can make accessing the database inefficient.
- In addition storing files in a database record makes it difficult to access them later through HTTP requests.

To avoid these limitations of databases, files are often stored directly on the file system with references to the file path stored in the database.

Database Migration and Seeding

- The workflow to starts from creating the column in the database to store the meta data.
- In the Products example, we want to store image of a product.
- The column we need is of type **string** to store the path to the file. See an extract from the migration file:

```
Schema::create('products', function(Blueprint $table){
    $table->increments('id');
    $table->string('name');
    $table->float('price');
    $table->integer('manufacturer_id');
    $table->string('image');
    $table->timestamps();
});
```

- The image needs to be displayed in the details page, so it needs to be accessible. So it will need to be placed in the **public** directory.
- For the Products example, we will place images in **public/products_images** directory.
- For seeding purpose, we will place a image with filename **default.png** in the **products_image** directory, and use this to seed products. An extract from **ProductsTableSeeder.php**

```
DB::table('products')->insert([
   'name' => ‘iphone',
   'price' => ‘600',
   'manufacturer_id' => '1',
   'image' => 'products_images/default.png',
   'updated_at' => \DB::raw('CURRENT_TIMESTAMP'),
]);
```

File Upload Form

- Files (and images) are uploaded by using a HTML input type "**file**".
- In addition the form tag must have an encoding type of "**multipart/form-data**".
- Note that the HTTP method must be POST to upload files.
- An extract from the **views/products/create_form.blade.php**

```
<form method="POST" action="/product" enctype="multipart/form-data">
    {{csrf_field()}}
    <p><label>Name: </label><input type="text" name="name" ></p>
    …
    <p><input type="file" name="image"></p>
    <input type="submit" value="Create">
</form>
```

Storing the image and meta data

- When a file is uploaded from a form, PHP places the uploaded file in a temporary directory on the server. The file needs to be moved to a permanent location.
- Laravel provides **store()** on a file upload instance to move the file to a specified location. E.g.:

```
$image_store = request()->file('image')->store(‘products_images', 'public');
```

- **request()->file(‘image’)** retrieves an uploadedFile instance. **‘image’** is the value in name attribute in the upload form.
- **store()** will move the file to the storage directory.
  - The first parameter defines the name of the directory in the storage ‘disk’ to move this file to.
  - The second parameters specifies the ‘disk’ as defined in config/filesystems.php. The default directory for ‘public’ is **storage/app/public**.
- So the above code will move the image to **storage/app/public/products_images**
- The call returns the path and filename, which is then stored in **$image_store**.

- An extract from ProductController:

```
public function store(Request $request)    {
    $image_store = request()->file('image')->store('products_images', 'public');
    $product = new Product();
    $product->name = $request->name;
    $product->price = $request->price;
    $product->manufacturer_id = $request->manufacturer;
    $product->image = $image_store;
    $product->save();
    return redirect("product/$product->id");
}
```

Display Uploaded Image

- To display the uploaded image, simply embed the path, i.e. **{{url($product->image)}}**, in an img tag.
- However, the storage directory containing the image is not in the public directory, so is not accessible from outside.
- To make the storage directory accessible, we can create a **symbolic link** of the storage directory in the public directory. E.g. execute the following command in the product project's directory:

```
 ln -s ~/html/<path to products directory>/storage/app/public/products_images
public/products_images
```

- This will create a **public/products_images** directory that contains the content in **storage/app/public/products_images**.
- **public/products_images** is a symbolic link to **storage/app/public/products_images**
- Note: if the **ln** command isn't correct, e.g. the path to the storage directory is incorrect, the link will still be created and no error report. However, you won't be able to access the symbolic link directory/file.

- In the product example, we display the product image in the details page. An extract from **show.blade.php**:

```
@section('content')
    <h1>{{$product->name}}</h1>
    <img src="{{url($product->image)}}" alt="product image"
style="width:300px;height:300px;">
    <p>Price: {{$product->price}}</p>
    <p>Made by: {{$product->manufacturer->name}}</p>
    ....
@endsection
```

Laravel tip – Clear Cache

- If you copied a Laravel project as a base to work on, you should clear any configuration cache before start working on the copy.
- Cache can be cleared using the artisan command:

```
php artisan config:cache
```