

Web forms in *FuelPHP* framework

Web Technologies II
Krišs Rauhvargers, 2013

Web forms

- There are several ways user can interact with the server application:
 - GET parameters of an URL address
 - Can be guessed by the user knowing the pattern, e.g. `http://google.lv/?q=test`
 - Can be generated by a server-side script set as *href* of an anchor
 - Web forms, submitted either as GET or as POST data
 - text input boxes, buttons, etc.

HTML form elements

test

input type="search"

Kā Tu vēlies sevi dēvēt? (Šo gan ieraksti)

input type="text"

Tavs e-pasts? (Nav obligāti; citiem netiks rādīts)

input type="text"

Mājas lapas adrese? (Ja nav, nenorādi)

input type="text"

☐ Saņemt turpmākos komentārus uz norādīto e-pasta adresi?

☐ Ja vēlies rakstīt bez garumzīmēm (glaazshskjuunju ruukjiishi), iekļekšē un sāc rakstīt.

input type="checkbox"

Tavs komentārs: (Izpaudies brīvi)

textarea

Atruna par moderāciju. Daži vārdi, var gadīties, ka ir iz melnās listes (*viagra* and *stuff*).
Tādi komentāri tiek aizturēti, pirms parādās lapā. Ja Tavs komentārs neparādās uzreiz,

Pievienot komentāru

input type="submit"

Receiving form data on the server side

- To receive data:
 - Form should have the `action` attribute set
 - Each field to be submitted must have a **name** attribute
- Depending on the method, data values are available as `$_POST` or `$_GET`
 - `<input type="text" name="title">`
 - results in
 - `$_POST["title"]`
 - containing the value user wrote in input box

Field type specifics

- Textarea-s preserve line breaks
- Checkbox-es are not submitted if not checked
 - workaround: add a hidden element with the same name before the checkbox. Its value will be submitted instead
- Multi-select list box names have to end with a [], e.g. "colors[]" to be readable on the server

Forms in FuelPHP

- A *view* may contain HTML form(s)
 - multiple forms in a page are OK
- The form can be submitted to a controller action
 - "action" attribute of the form can be full URL address
 - `http://localhost/eventual/event/edit`
 - or relative address, but be careful (results will depend on the current address)
 - `delete/`
 - if omitted, the form will be posted back to the same controller action

Forms in FuelPHP

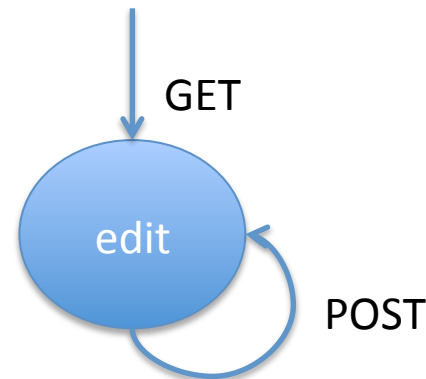
- It's typical that a controller action submits data back to itself.
- This way, the controller action may be called in two ways:
 - initial load, when user opens a form.
 - Typically, a GET request to the action.
 - post-back, when user has edited data and is submitting the form
 - Typically, a POST request, because of form data

You can catch the post-backs in FuelPHP:

`action_index`: opens on all requests

`get_index`: opens on GET requests

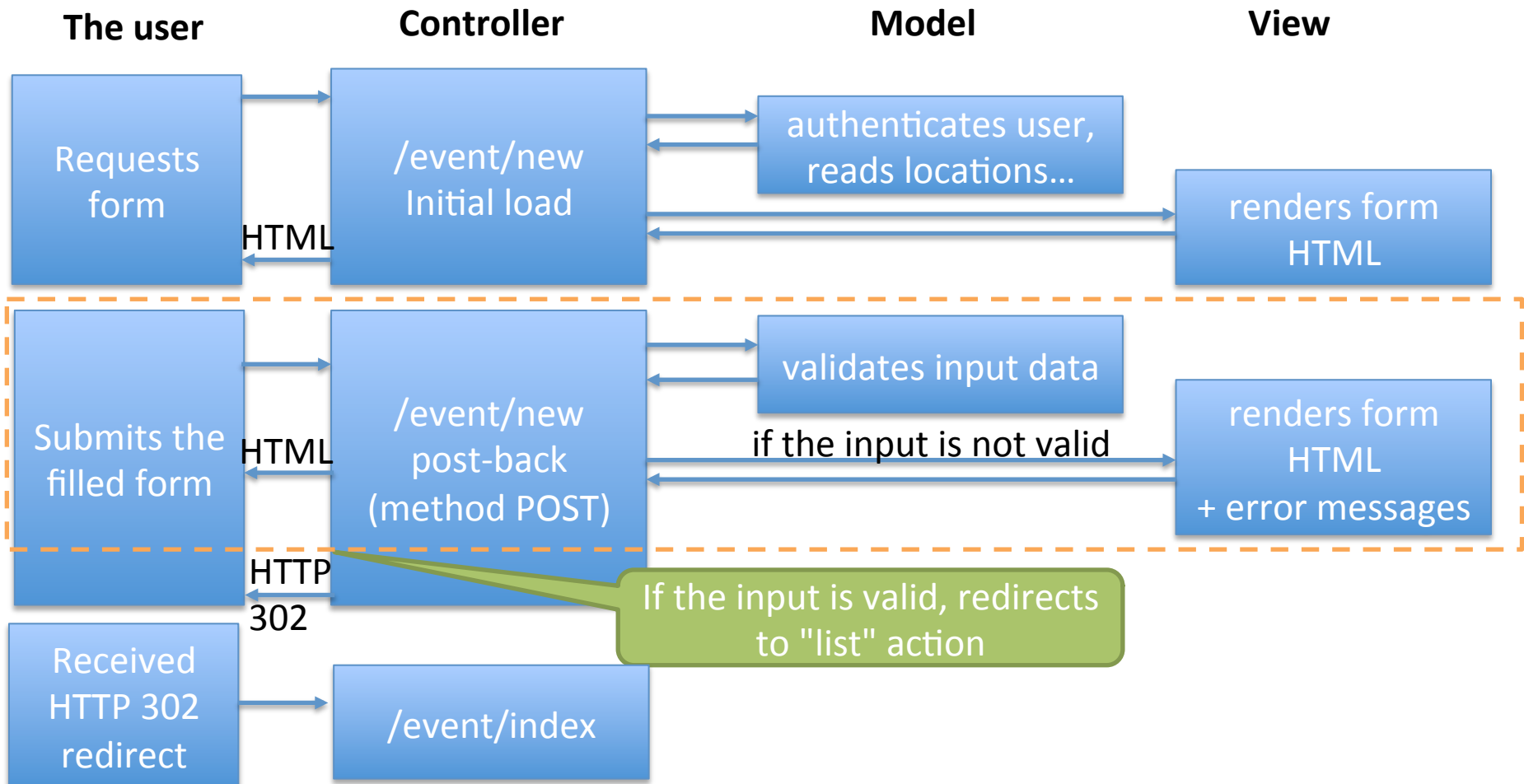
`post_index`: only on POST requests



user input sent back to "edit" action

Web form life cycle in *FuelPHP*

Scenario: the user wants to create a new event.



"Input" class in FuelPHP

- You may work with `$_POST`, `$_GET` arrays in FuelPHP
- "Input" class wraps around all inputs and provides utility methods

```
//without Input class
```

```
if (isset($_GET["title"])) {  
    $title = $_GET["title"];  
} else {  
    $title = "";  
}
```

```
//using input class:
```

```
$title = Input::get("title", "");
```

Input class utility methods

- `Input::method()` – what method was used to call current action
 - Useful for detecting if the form is loaded the first time or it's a post-back

```
if (Input::method() == "POST") {  
    ...  
}
```
- `Input::is_ajax()` – is the current request a full page load, or an AJAX post-back
- `Input::uri` – current action uri address

GENERATING FORM HTML

Form helper class

- Form class can be used to generate HTML forms
 - Code it generates is sometimes really simple, e.g.
 - `Form::close()` generates a plain `"</form>"` tag
- Form class provides static methods, no need to create an instance
- Useful when default values have to be set

```
echo Form::input('title',  
    Input::post('title',  
        isset($blog) ? $blog->title : '' )  
    );
```

Example of extensive use of Form class

Every form element can be generated by Form::something(). This example proves it.

```
1 <?php echo Form::open(); ?>
2 <p>
3     <?php echo Form::label('Name', 'name'); ?>
4     <?php echo Form::input('name', Input::post('name', isset($comment) ? $comment->name : '')); ?>
5 </p>
6 <p>
7     <?php echo Form::label('Comment', 'comment'); ?>
8     <?php echo Form::textarea('comment', Input::post('comment', isset($comment) ? $comment->comment : '')); ?>
9 </p>
10 <?php echo Form::hidden('mid', Input::post('mid', isset($message) ? $message : '')); ?>
11 <div class="actions">
12     <?php echo Form::submit(); ?> </div>
13 <?php echo Form::close(); ?>
```

Screenshot from "FuelPHP crash course",
<http://ucf.github.com/fuelphp-crash-course/>

Validation class

- Validation class can be used for validating typical form inputs
 - required fields
 - minimum, maximum values
 - minimum, maximum field length
- Custom validators can be used, see:
 - http://fuelphp.com/docs/classes/validation/validation.html#extending_validation
 - example:
https://github.com/naivists/TTL_2012/blob/master/fuel/app/classes/model/orm/location.php#L38

Using Validation class

Simple way: Validation works with POST data, addressed by field name

```
$validation = \Validation::forge('create');
$validation->add('title', 'Title')
    ->add_rule('required')
    ->add_rule('min_length[5]');
if ($validation->run())
{
    // successful case, validation succeeded!
    $goodtitle = $val->validated("title");
}
else
{
    // validation failed
}
```

Validating ORM models

- An ORM model is not always "valid".
 - Does a country without a name make sense?
 - A user without a login name?
- Each model class should implement `validate()` which inspects the current model instance
 - There is no standard method to override, just a convention.
 - Classes generated by "scaffold" have it
 - Used by the `Fieldset` class

A self-validating model:

Model code

```
class Model_Event extends Model
  public static function validate($factory)
  {
    $val = Validation::forge($factory);

    $val->add_field('event_title',
                  'Event Title', 'required|max_length[255]');

    $val->add_field('event_date',
                  'Event Date', 'required');

    return $val;
  }
}
```

Controller action code

```
$val = Model_Event::validate('create');
if ($val->run()) {
  //do something
}
```

FEATURE RICH FORMS

"Web 1.0" forms

- Historically, forms could use features provided in HTML
- User friendly interface features missing:
 - plain text input fields
 - fields require specific formatting, but allow wrong inputs
 - form validation requires additional client-server round-trip
- Both developer laziness and technology limitations were to blame

Enhancing user controls

- Since the raise of JavaScript, client object-model changes are possible
 - e.g. initial validation at the client side
- "Progressive enhancement" approach:
 - JavaScript-enabled clients are provided with rich functionality using scripting and browser features
 - "dumb clients" get a simple "Web 1.0" form
 - But they **do** get it and it is working without JS!

Enhancing input controls: date

- Nearly every country has its own date format
 - 13.11.2012 : Latvia, Germany
 - 11/13/2012 : United States
- People think of dates in terms of calendar weeks/months, not numbers
- Good representation: a small calendar
- Solution: a date picker:
 - activated when user focuses the date field
 - formats date according to format used in the system
 - helps visualize the calendar

Date picker

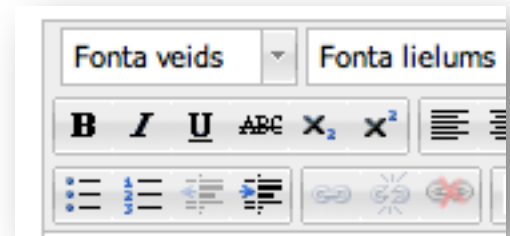
- jQuery UI Date picker:
 - <http://jqueryui.com/datepicker/>
- Twitter *Bootstrap* datepicker:
 - <http://www.eyecon.ro/bootstrap-datepicker/>
- Moo Tools date picker
 - <http://www.monkeyphysics.com/mootools/script/2/datepicker#examples>
- YUI calendar:
 - <http://developer.yahoo.com/yui/calendar/>

Time picker

- Sometimes, time input is needed as well.
- Time picker "as in" jQuery UI:
 - <http://fgelinas.com/code/timepicker/>
- Moo Tools date & time picker
 - <http://aryweb.nl/projects/mootools-datepicker/>
- Bootstrap time picker
 - <http://jdewit.github.com/bootstrap-timepicker/>
- Date and time picker for jQuery UI
 - <http://trentrichardson.com/examples/timepicker/>

Text editors

- Textarea is useful for plain-text inputs.
- We want formatting.
 - Bold, italic, bulleted lists.
- Progressive enhancement in this case:
 - dumb clients get a textarea.
 - others get a rich text editor
 - this is possible because of "contentEditable" DOM property (<http://html5demos.com/contenteditable>)



Rich text editors

- TinyMCE: classic, used in e-Studijas
 - <http://www.tinymce.com>
- Aloha editor: quite recent, they call themselves "the HTML5 editor"
 - <http://www.aloha-editor.org>
- CK Editor
 - <http://ckeditor.com/demo>

Multiple select lists

- Multiple selects are not well received by users.
- We would like to see the picked and un-picked items separately
- jQuery UI multi select:
 - <http://quasipartikel.at/multiselect/>
- As checkbox dropdown (jQuery plugin)
 - <http://code.google.com/p/dropdown-check-list>
- Multiselect project
 - <http://loudev.com>

Select boxes

- Default select boxes miss "find" operation
- Cannot add new items to the list of options
- Chosen
 - <http://harvesthq.github.com/chosen/>
- Select2 (fork of *Chosen*, supports larger data sets)
 - <http://ivaynberg.github.com/select2/>

File uploads in FuelPHP

- Form element as usually:

```
<input type="file" name="eventposter" />
```

- Don't forget the

```
<form enctype="multipart/form-data"> !
```

- **\$_FILE**

- File upload helper: Upload class

- Some configuration at

```
fuel/core/config/upload.php
```

or

```
fuel/app/config/upload.php
```

Where to put uploaded files

- Files are part of your web site
 - upload under `/public/`
 - use direct links in all pages
- Needs access control:
 - Upload under `/fuel/app`
 - Use a download script (authorization as first step)
 - example:

https://github.com/naivists/TTII_2012/blob/master/fuel/app/classes/controller/event.php#L159

File uploads

- The default "Browse..." button is unhandy:
 - cannot set allowed file extensions
 - only one file per "browse" click
 - does not support drag & drop (except some browsers)
- When uploading larger files, upload progress meter would be nice.

File upload tools

- Uploadify: Most popular uploader, jQuery compatible
 - uses Flash or HTML5
 - <http://www.uploadify.com>
- PlUpload: Relatively new
 - uses Flash, Silverlight, Google Gears, HTML5
 - <http://www.plupload.com>
- NB! Both tools measure "bytes sent". To get "bytes received", AJAX requests for polling have to be used.

Other notes on file uploads

- Things get complex when using validation and uploading files:
 - situation: user picks a file for upload, but forgets to fill out the title field
 - the form is submitted, it is not valid
 - PHP keeps uploaded files only until the end of request (other web engines are similar)
- All user uploads have to be collected and temporarily stored on the server until form data is valid
 - "Remembering" them in current user session would be a good idea

Mobile UI improvements

- "jQuery Mobile" helps create "finger tappable" web pages.
 - If available, uses native controls of the given platform (e.g. percentage slider)
 - If not available, renders "something similar"
- Such interface is usable on a computer as well
- <http://jquerymobile.com>

SOLVING DATA RELATIONSHIPS

One-to-many

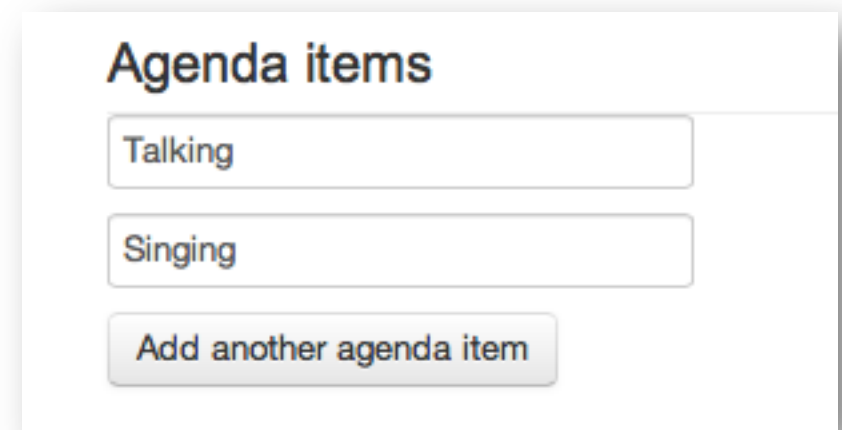
- Example:
 - An event consists of agenda items
 - When the event is added, user wants to add agenda items.
- No simple solution without scripting
 - cannot predict the number of agenda items, hence cannot render an input form
- in "web1.0":
 - During registration of event, no agenda items can be added
 - When item is created, can add items one-by-one.

One-to-many (cont'd)

- If implementing adding related items without JavaScript
 - "create" form of the "agenda" entity receives event id as GET parameter
 - <http://eventual.org/agenda/add/?event=1>
 - the form loads event data and displays it
 - when agenda item is saved, adds event id value to it.

One-to-many

- If using HTML+JavaScript, can dynamically add new item input boxes.
- Typically involves a "template" row being copied to create a new element
- At the server side, will have to "guess" the field names generated at the client



The screenshot shows a web form titled "Agenda items". It contains two text input fields, one with the text "Talking" and another with "Singing". Below these fields is a button labeled "Add another agenda item". This illustrates the one-to-many relationship where a single form structure is used to create multiple items.

Choosing from a classifier

- Many-to-one(fixed count) problem
 - e.g. an event belongs to a location
 - or, a location belongs to a country
- User has to make choice of the existing values.
- Historically, a <select> is used
 - But if there are a lot of items?

Searching the classifier

- When there are too many parent items
 - Categorizing, looking for smaller amount of items
 - Searching in the classifier
- jQuery Autocomplete (<http://jqueryui.com/autocomplete/>) or Select2 can be used
 - The classifier entity (e.g. location) should implement "find" action
 - Ajax calls to "find" will be performed during search

Choosing from a classifier (+option to add)

- The classifier may not contain the target element
 - e.g. The even takes place in location which is not registered yet
- Must provide solution for adding new classifier entry
- Must not lose the input data
- If adding new classifier item is very simple (single field), can use *Select2*

Adding new items in a pop-up

- Common solution:
 - a modal popup window with "create" form of the classifier
 - when the new item is saved, popup closes and selected value in parent form changes

The screenshot displays a web application interface with a modal popup. The background is a dimmed view of a 'New Event' form, which includes fields for 'Title of the', 'Description', 'Date', and 'Location'. The 'Location' field is a dropdown menu currently showing 'Pick a location'. Overlaid on this is a modal window titled 'Create a new location' with a close button (X) in the top right corner. Inside the modal, there is a 'Title' text input field, a 'Country' dropdown menu showing 'Pick a country', and a blue 'Save' button at the bottom.

Creation of related items in the same form

Location

Pick a location

Add new location

Save

If the "create" form is not too large, can load it in as a "sub-form".

At the server, first the location will have to be saved, then the event.

Location

Pick a location

Create a new location

Title

Country

Pick a country

Save

N:N relationships

- Frequently, N:N relationships have to be solved
 - e.g. Books:Authors. A book can have multiple authors. Each author can have multiple books.
- *Select2* "tagging" can be good help, if the related category can be described by one-two words