auxiliary

Objektorientiertes Framework für PHP5

Agenda

- Hintergründe zu auxiliary
- Basisfunktionalität
- Aktueller Status
- Ein paar Begriffe
- Beispiel: Aufbau einer Seite
- Spotlight: Features
- Selbstkritik
- Ausblick

Hintergründe zu auxiliary

- Komplett objektorientiert, PHP5 benötigt (Nutzung der OO-Möglichkeiten)
- MySQL3 (Umstieg auf MySQL4 steht bevor)
- Nach Installation komplett über Website steuerbar (Templates, Inhalte etc.)
- Basisframework nutzbar als (sehr) einfaches CMS für kleine Websites
- Mittels eigener Module zur gewünschten Webapplikation ausbaubar
- Lizenz: BSD
- Website: http://auxiliary.kl-s.com/

Basisfunktionalität

- Projektmanager zum Verwalten mehrerer Websites über eine andere Site
- Templatesystem (klein und einfach, Vermeidung von Logik in Templates)
- Einfaches CMS (Artikel, untereinander gruppierbar)
- Einfache Benutzerverwaltung: Registrierung, Login, (Admin-)Rechte (erweiterbar)
- Internationalisierung
- Viele Klassen zur Unterstützung der Entwicklung eigener Module

Aktueller Status

- Entwicklung nicht abgeschlossen
- Schnittstellenänderungen nicht ausgeschlossen
- Nicht ausreichend getestet
- Noch keine Zusatzmodule verfügbar
- Neu: RFCs begleiten die Entwicklung
- Nur ein Entwickler
- Mehr Wünsche als Zeit (und Lust) :)
- Version: 0.2.1pl2 (HEAD: 0.2.2)

Ein paar Begriffe

- RFC: Neue Funktionalität wird zunächst in einem RFC beschrieben, erst dann implementiert (RFCs aktuell nur über CVS verfügbar)
- Modul: „Gekapselte“ Funktionalität
- Submodul: Noch mal „gekapselte“ Funktionalität innerhalb eines Moduls

Beispiel: Aufbau einer Seite (1): www/users/index.php

```
<?php
require('../../cfg/config.php');
auxiliary::load('core/users/website/userindex.class.php');
auxiliary::main('UserIndex');
?>
```

Beispiel: Aufbau einer Seite (2): UserIndex

```
<?php
auxiliary::load('core/users/users.php',
    'core/users/controller/userindexcontroller.class.php',
    'core/lang/website/website.class.php'
);
class UserIndex extends Website
{
    public function __construct()
    {
        $this->module_id        = USERS_MODULE_NAME;
        $this->menu_id          = USERS_MENU_ID_INDEX;
        $this->parent_module_id = USERS_MODULE_NAME;
        $this->parent_menu_id   = USERS_MENU_ID_INDEX;
        parent::__construct('UserIndexController');
    }
?>
```

Beispiel: Aufbau einer Seite (3): Website

- Instanziieren des zuständigen Controllers
- Weiterleiten des Users falls nicht authentifiziert
- Zusammenbau der Seite aus Menü, NavBar, Überschrift, Inhalt, JavaScript
- Start des Controllers, Abruf der Ergebnisse (Inhalt)
- JavaScriptLibrary: Einfügen beliebiger benötigter JavaScripts
- Browser-Caching: 304 Not Modified
- Ausgabe an Browser

Beispiel: Aufbau einer Seite (4): Controller

- Die „eigentliche“ Arbeit
- Authentifizierung (Invalid/InsufficientAuthException)
- Instanziieren von View und Model (Model nicht notwendigerweise)
- Aufbau des Inhaltes: Daten von Model abfragen, in View einwerfen
- Prüfen von Benutzereingaben
- Geschäftslogik („[...] eine seltsame Bezeichnung, weil nur wenige Dinge weniger logisch sind als die sogenannte Geschäftslogik.“ Martin Fowler)

Spotlight: Features (1): Request-Parameter

```
<?php
$request = Request::getInstance();
if (isset($request['data']) == TRUE) {
    echo 'Parameter data ist gesetzt.<br>';
}
var_dump($request['data']);
$request->addFilter('data', 'StringFilter');
var_dump($request['data']);
?>
```

test.php?data=Hello!
Ausgabe:
Parameter data ist gesetzt.
NULL
string(5) "Hello"

- Div. vorgefertigte Filter
- Eigene Request-Klassen möglich, z.B. CLI
- HTTPRequest ist default

Spotlight: Features (2): MVC-Unterstützung

- Vorgefertigte erweiterbare Basisklassen für View und Controller, z.B. FormController, ListingController, FormView, ListingView, SimpleView
- Ein Controller kann einen anderen Controller verwenden, z.B. Listing in einem Formular:

```
<?php
$ul = new UserListingController(/*div. Parameter */);
$this->view->makeEmpty('Zugehörige User:', $ul->getView());
?>
```

oder kompletter Ersatz der eigenen View mit der View des anderen Controller:

```
<?php
$ul = new UserListingController(/*div. Parameter */);
$this->view = $ul->getViewObject();
?>
```

Spotlight: Features (3): Speichern von Daten

- Basisklasse Object bietet Möglichkeiten, Daten zu speichern
- Eigene Klasse erweitert Object
- Speichern: `$myObject->save($data);` entscheidet selbstätig über INSERT oder UPDATE
- Standard: myobject_id, name, registered, (last_changed), status
- Keine Bindung an eine Tabelle!
- Für viele Fälle ausreichend.

Spotlight: Features (4): Event System

- RFC 2006-01-20-1, basierend auf:
- EventDispatcher, a Java package for event-driven development [http://java.schst.net/EventDispatcher]
- NotificationCenter of Apple's Cocoa-Framework [http://developer.apple.com/[...]/Notifications/]
- Component and Event-Driven Architectures in PHP5 [http://schst.net/[...]/event-driven.pdf]

Spotlight: Features (5): Cache

- Manche Operationen sind teuer: Cache nutzen
- Cache wird manchmal automatisch geleert: Update, Installation/Entfernung von Modulen, Änderung von Projektkonfiguration oder Templates
- Details zur Verwendung: RFC 2005-10-04-3
- Aber: Kein Cache für komplette Seiten! Nur Teile der Seite oder Datenstrukturen, deren Erstellung bei jedem Request zu teuer wäre

Spotlight: Features (6): Internationalisierung

- Sprachdateien sind ini-Dateien
- Klasse I18N bietet Zugriff auf Sprachstrings:

```
<?php
$i18n = I18N::getInstance();
echo $i18n->get(MY_MODULE_NAME, 'SECTION', 'hello_world');
?>
```

Falls deutsch: Hallo Welt!
Falls englisch: Hello World!

- Derzeitig mitgeliefert: Deutsch und Englisch
- Menü-Sprachstrings allerdings nur in DB
- ✗ Keine Lokalisierungen einer Sprache möglich (also nur de, nicht de_DE, de_AT usw.)
- Submodul locales bietet Möglichkeiten, Texte mehrsprachig zu speichern

Spotlight: Features (7): Setup/Update

- Installieren: Dateien auf Server, cfg/config.php anpassen, /index.php aufrufen, Formular ausfüllen, fertig
- Update: Dateien auf Server, /index.php aufrufen, Setup-Passwort eingeben, fertig
- Modul installieren: Dateien auf Server, als Admin in axl einloggen » Mitgliederbereich » Modul installieren
- Modul updaten: noch nicht fertig :)
- Modul entfernen: Als Admin in axl einloggen » Mitgliederbereich » Modul deinstallieren

Spotlight: Features (8): JavaScriptLibrary

- Ganz frisch: HEAD only
- JavaScripts in einzelne Funktionalitäten zerlegt
- Jede Seite lädt nur die wirklich benötigte Funktionalität (doppeltes Laden gleicher Funktionalität wird automatisiert unterbunden)
- JavaScript kommt da hin, wo es wirklich hingehört: in den Kopfbereich, nicht wild durch das gesamte HTML verstreut

Selbstkritik

- ✗ Mangelhaftes Testen, da Umfang mittlerweile zu groß
- ✔ Gut: „CMS“ erzwingt einheitliches Design
- ✗ Schlecht: einheitliches Design für viele Dinge zu unflexibel
- ✗ MVC-Umsetzung noch zu unflexibel
- ➔ Kein zentraler Page Controller: gut oder schlecht?
- ✗ Module mit erweiterter Funktionalität lassen auf sich warten (z.B. Newssystem)

Ausblick

- XML-Unterstützung (Serializer/Unserializer, RFC 2005-10-04-1)
- Unittests!
- Verbesserung Tag-Unterstützung
- Suchmaschine basierend auf Tags (RFC 2005-11-11-2)
- Logging
- PDO statt mysql_*
- MySQL4/MySQL5
- AJ(AX)
- Umstieg von CVS auf SVN?

Das wars...

- Niemand eingeschlafen?
- Danke fürs Zuhören.
- Fragen? Hinweise? Anregungen?
- Folien: http://talks.frankkleine.de/