



## symfony from scratch @ sfPHP

A look at the PHP5 symfony platform  
and how to make symfony the  
foundation for your next web project

# Why you will learn

- What is the symfony project?
- What does symfony project provide?
- How is it different from (zend, cake, django, rails)?
- How to get a project started?
- How to integrate symfony with Zend
- How to use symfony + Y! UI to create a complete web platform (php, css, js, patterns)

# What is the symfony project?

- Full stack web application framework built for rapid application development
- Written in PHP5 based on MVC pattern
- Open source and free
- Licensed under the MIT license
- A growing community since October 2005
- Founded by Fabien Potencier, sponsored by Sensio Labs

# Why symfony?

- Agile Development
  - Built on best practices using proven design patterns
  - Built to factor out common patterns and allow developers to focus on application logic
  - Automates the tedious tasks most developers face daily
- Performance / Stability
  - Proven to scale on very active web sites - Y! / Sensio
  - Tested code base (8000+ unit and functional test)
- Maintainability
  - Provides reusable structures (DRY)
  - Enforces consistency amongst developers
- Support
  - Great documentation and very active community

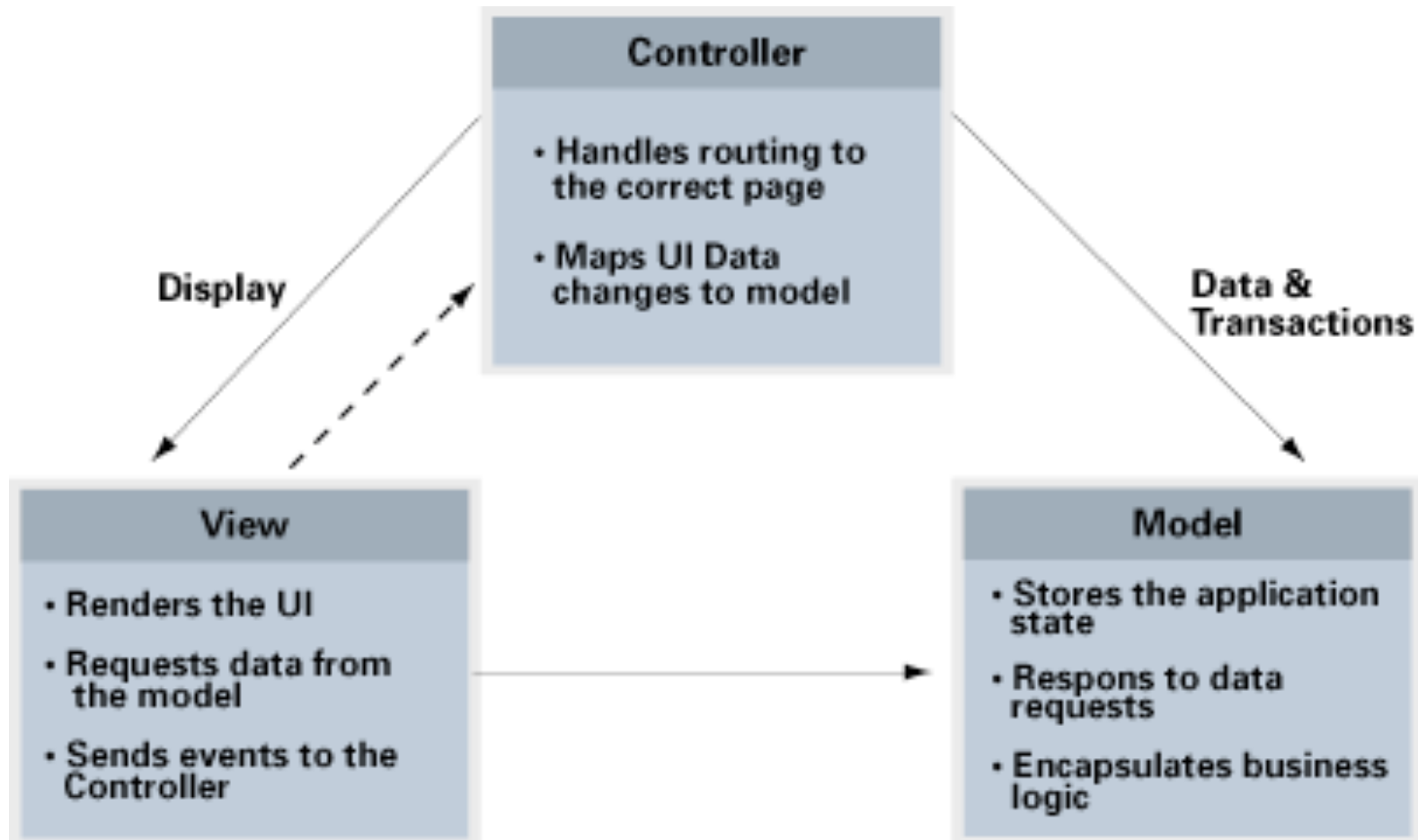
# How is symfony different?

- symfony is not a component library like eZ Components or Zend Framework
- Adopt best ideas from anywhere, using existing code if available (prado, rails, django)
- It is a full-stack framework for building complex web applications
  - When you create a symfony project, you have all of the tools you need to get started.
  - It is not targeted for building simple applications
  - No assembly required
- A flexible and pluggable architecture

# Clean Design

- Built using proven design patterns
  - Model-View-Controller
    - Separation of data access, presentation logic, and business logic
  - Factories for customization
- Built to embrace agile development principles
  - Allows developers to easily follow
    - Extreme Programming / Test Driven Development
    - DRY (Don't Repeat Yourself)

# MVC (stolen from the web)



# Minimal Requirements

- A web server that is capable of serving PHP5
- Database server supported by Creole/PDO
  - MySQL, PostgreSQL, SQLite, Oracle, MSSQL
- Recommended Configuration
  - Apache
    - mod\_rewrite
  - PHP 5.1.x
    - APC (xCache or eAccelerator are also supported)
    - Syck
    - xDebug

# symfony 1.0 -> 1.1

- Evolving into a platform to build your own framework
- Clean separation and communication between components (request, response, forms, validations, i18n, user, etc..)
- Decoupling of dependencies - introduction of Event Dispatcher
- New features: Cache Factories, New Form System

# symfony cli

- symfony cli
  - Provides project management tasks for
    - Creation of projects, applications, and modules
    - Configuring database + project settings
    - Installing and Removing plugins
    - Managing model (regenerating om + loading fixtures)
    - Executing Unit + Functional Tests
    - Deploying project to production
    - Rotating and purging logs
  - Foundation for cli applications and custom tasks

# Configuration System

- Convention over configuration
- Based on YAML
  - YAML is readable, concise, and hierarchy support
- Uses configuration factories to handle different yaml files (custom parsing)
- Cascading Configuration
  - Framework -> Project -> Application -> Module
- Configuration is cached to php

# Environments

- Environment determined by front controller
- Three default environments
  - Production
    - Caching enabled, logging disabled, clean URLs
  - Development
    - Caching disabled, logging enabled, debug toolbar enabled
  - Testing
    - Similar to production without caching
- Add as many environments as you require
  - Staging, QA, alpha...

# Debug Environment

- Web Debug Toolbar
  - Browser access to relevant information
    - Request Parameters / Sessions / Cookies
    - Configuration (symfony/php)
  - Logs & Debug Messages
    - xDebug Integration (Execution Traces)
  - SQL Query Monitor
    - Raw SQL + Query Time
  - Performance Monitor
    - Memory Consumption
    - Timers
  - Cache Monitor
  - Logging

# Routing System

- Front Web Controller dispatches request to appropriate controller based on route
- Routing based on path info or patterns
  - Pattern routing supports regex requirements
- A maintainable `mod_rewrite` with helpers

```
blog_permalink:  
  url: /blog/:permalink  
  param: { module: blog, action: show }  
  requirements: { permalink: .* }
```

# The Controller Layer

- Contains all business logic
- Based on modules and actions
- Actions are controllers
- Modules are a collection of related actions
  - User = login/logout/register/profile
- Modules are represented by a directory structure (config, lib, templates) and a class (actions.class.php/components.class.php)
- Actions can share settings (view, cache, security)
- Actions return views (Success, Error, Custom)

# The Basics = Hello World

Controller – (project/apps/frontend/modules/simple/actions/actions.class.php)

```
class simpleActions extends sfActions
{
    public function executeIndex($request)
    {
        $this->text = 'hello world';
    }
}
```

View - (project/apps/frontend/modules/simple/templates/indexSuccess.php)

```
<?php echo $text; ?>
```

# The Model Layer

- ORM: Propel or Doctrine
- Default is Propel (so that is what we will discuss)
  - Not exactly ActiveRecord = Row Data Gateway + Table Data Gateway Implementation
    - Clean separation of table operation objects and row instances
    - “Object” classes for rows (RDG)
    - “Peer” classes for table operations (TDG)
  - Database abstraction via Creole library
  - Code + DDL Builder + tasks built on Phing
- Customizable Builders
  - Reusable behaviors: (Act As, Paranoid Delete, updated\_at, created\_by)

# The Model Layer

- Built in tools for pagination, sorting, and filters
- Support for behaviors
- Object model built via schema.yml
  - Can be reverse engineered from existing db
  - Support for indexes, references, constraints, and db specific properties (autoincrement, sequences)
- schema -> model generation -> sql -> insert

# Model - Schema

propel:

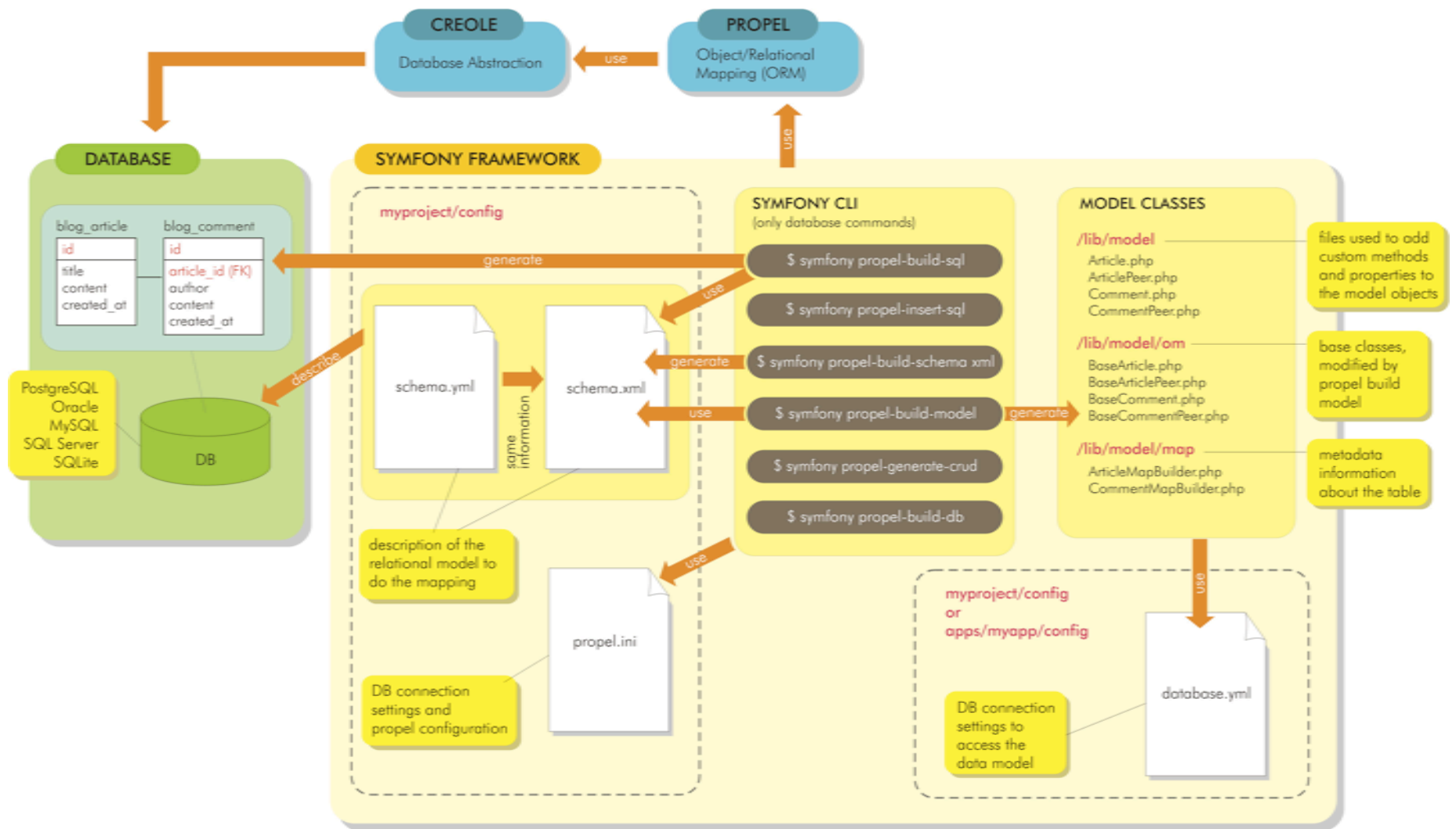
posts:

```
_attributes: { phpName: Post }  
id:         { type: integer, required: true, primaryKey: true, autoIncrement: true }  
title:      varchar(255)  
excerpt:    longvarchar  
body:       longvarchar  
created_at: ~
```

comments:

```
_attributes: { phpName: Comment }  
id:          { type: integer, required: true, primaryKey: true, autoIncrement: true }  
post_id:     { type: integer, primaryKey: true, foreignTable: posts, foreignReference: id, onDelete: cascade }  
author:      varchar(255)  
email:       varchar(255)  
body:       longvarchar  
created_at: ~
```

# The Model Flow



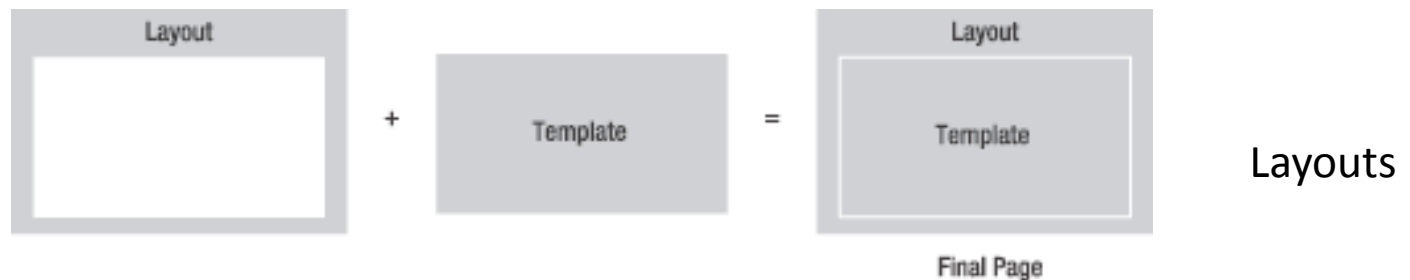
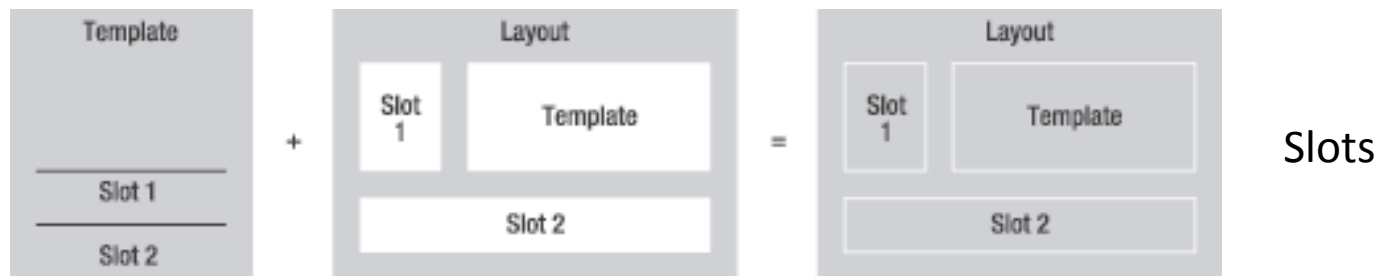
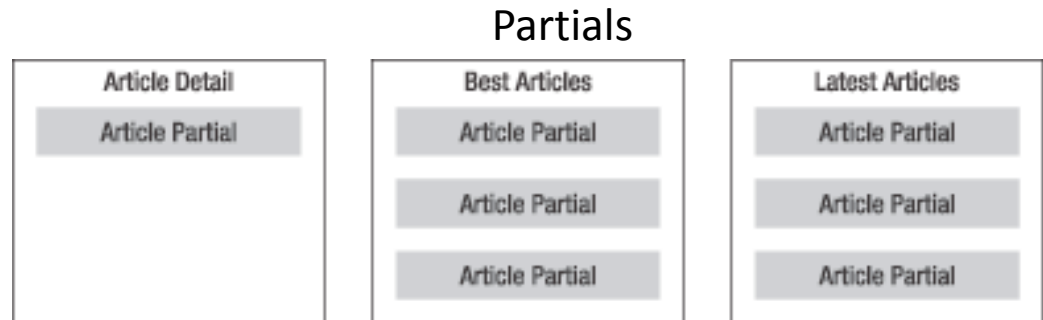
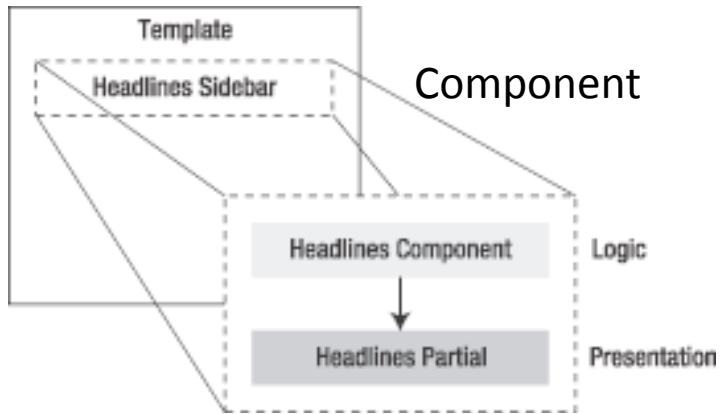
# The View Layer

- PHP as a template language
  - No smarty layer, instead we have helpers
  - Helpers are template functions (think: rails)
  - I18N, Date, Text, Cache, Routing
- View configuration managed via view.yml
  - Meta tags, title, javascript, css, layout, component slots
- Layout
- Partials + Slots
  - Template fragment without business logic
- Components
  - Template fragment with business logic
- Component Slots
  - Slot in layout for contextual content

# Views are organized and reusable

- Layouts are the full page and popup views
- Slots are placeholders in a page
- A partial is a static template fragment (think: copyright notice)
- A component is a template fragment with business logic (think: popular list)
- A component slot is a contextual placeholder for different components (based on context)

# The view layer in pictures



# AJAX Toolkit Integration

- Prototype/script.aculo.us built-in
- Plugins available:
  - sfUJSPlugin -> helpers for UJS via jQuery
  - sfYUIPlugin -> helpers for YUI
  - sfDojoPlugin -> helpers for Dojo
- Build your own helpers

# Easy rich interactions in PHP

- Template Helpers inspired from Rails
  - Based on [prototype/script.aculo.us](http://prototype/script.aculo.us)
- Easy implementations in one line of php
  - AJAX Interactions -> `link_to_remote`
  - Visual Effects -> `visual_effect`
  - Auto Complete -> `input_autocomplete_tag`
  - Inline Editing -> `input_inline_editor_tag`
  - Drag and Drop -> `draggable_element`
  - Rich Text Editing -> `textarea_tag` w/ `rich=tinymce|fck`
  - Rich Calendar -> `input_date_tag`

# The Caching System

- Powerful view cache layer with many adapters: file based, sqlite, memcache, apc, xcache, eaccelerator
- Can bind cache to anything (user/culture/id)
- Components/Partials can be cached individually
- Actions can be cached with or without layout
- View cache manager provides great interface to selectively invalidate based on url pattern
- sfSuperCache can provide full page caching (skipping php process)
- HTTP 1.1 Cache Header Management

# The Form System (1.1)

```
class SigninForm extends sfForm
{
    public function configure()
    {
        $this->setWidgets(array(
            'username' => new sfWidgetFormInput(),
            'password' => new sfWidgetFormInput(array('type' => 'password')),
        ));

        $this->setValidators(array(
            'username' => new sfValidatorString(),
            'password' => new sfValidatorString(),
        ));

        $this->validatorSchema->setPostValidator(new sfGuardValidatorUser());

        $this->widgetSchema->setNameFormat('signin[%s]');
    }
}

<?php echo $form; ?>
```

# Generating interfaces (crud + admin)

- CRUD (Create, Read, Update, Delete)
  - Basic admin for all fields
  - Built to customize/extend
    - Both the skeleton + generated code
- Administration Interfaces
  - Pagination, Filters, Sorting, Actions
  - Views list/tabular
  - Extremely customizable via generator.yml

# I18n & L10n

- Inspired by PRADO
- Flexible Configuration
  - Dictionaries can be XLIFF, gettext, or database
- Caching
- Template Helpers
  - Dealing with text = `__()`
    - Works with complex strings, enforces separation
  - Easy date, time, currency formatting
- Interface and Data Localization
  - Support for internationalization in database tables

# The Plugin System

- Plugins are packages
  - Configuration, Object Model, Libraries, Helpers, Modules, Tasks, Tests, Assets
- Easy to install via PEAR
  - symfony plugin-install or pear install
- Plugins can be overridden/extended
  - Configuration, Object Model, Actions, Templates

# symfony Applications

- **Askeet** - [trac.askeet.com](http://trac.askeet.com)
  - Question and Answer site/tutorial
    - Demo - [Askeet.com](http://Askeet.com)
    - Tutorial - [symfony-project.com/asket](http://symfony-project.com/asket)
- **Snipeet** - [trac.snipeet.com](http://trac.snipeet.com)
  - Snippet repository
    - Demo - [symfony-project.com/snippets](http://symfony-project.com/snippets)
- **Motilee** - [trac.motilee.com](http://trac.motilee.com)
  - Forum System
    - Demo – [motilee.com](http://motilee.com)
- **SteerCMS** - [steercms-project.org](http://steercms-project.org)
  - Content Management System
- **Symfonians** - [symfonians.net](http://symfonians.net)
  - Social Network + Community Showcase
- **Bartertown** – [sourceforge.net/projects/bartertown](http://sourceforge.net/projects/bartertown)
  - Auction System

# symfony Plugins

- ORM
  - sfPropelPlugin, sfDoctrinePlugin
- Javascript Framework Integration
  - sfUJSPlugin, sfJqueryPlugin, sfYUIPlugin, sfExtPlugin, sfDojoPlugin
- Third Party Library Integration
  - sfZendPlugin, sfSwiftMailer, sfJpgraphPlugin, sfChartdirectorPlugin
- Application Functionality
  - sfSimpleCMSPlugin, sfSimpleBlogPlugin, sfSimpleForumPlugin, sfSimpleNewsPlugin, sfMediaLibraryPlugin
- User Authentication
  - sfGuardPlugin, sfOpenIDPlugin
- Ecommerce
  - sfShoppingCartPlugin, sfAuthorizeNetPlugin, sfQuickbooksExportPlugin

# LIME - Unit and Functional Testing

- Based on Test::More Perl library
- Supports
  - Unit Testing
    - Given specific input validate specific output
  - Functional Testing
    - Does this work in context, a complete feature
- TAP compliant output
- Zero Dependencies
- Green is Good, Red is Bad

# A quick example application

- Building a feed aggregator
  - Frontend
    - Login / Logout / Register / Profile
    - Display feeds individually by name
    - Aggregate feeds and create aggregate RSS feed
  - Administration
    - Manage feed categories + feeds
    - Manage users, groups, and permissions

# Installation

- PEAR
  - pear channel-discover pear.symfony-project.com
  - pear install symfony/symfony
- Subversion (svn:externals)
  - svn export <http://svn.symfony-project.com/branches/1.1>  
symfony

# Create your first symfony applicaiton

1. `symfony generate:project sfproject`
2. `symfony generate:app frontend`
3. `symfony generate:module frontend common`

# Configure your project

- Configure your database
  - databases.yml, propel.ini
- Configure your symfony settings
  - settings.yml
- Configure your default view
  - view.yml
- Configure your application settings
  - app.yml

# Configure your databases

all:

propel:

class: sfPropelDatabase

param:

phptype: mysql

hostspect: localhost

username: sfshowcase

password: password

database: sfshowcase

persistent: true

encoding: utf8

compat\_assoc\_lower: true

# Configure you symfony settings

use\_database: on

use\_security: on

use\_flash: on

i18n: on

suffix: .

no\_script\_name: on

# Install plugins for project

```
symfony plugin:install sfGuardPlugin
```

```
symfony plugin:install sfFeed2Plugin
```

# Integrate your own PHP libraries

- Autoloading hooks for easy integration of
  - Zend Framework
  - EZ Components
  - Swift Mailer
  - PEAR
  - Your own libraries

# Integrate the Zend framework

- Download Zend and install in lib/vendor/zf
- Chain Zend Autoloading

```
class frontendConfiguration extends sfApplicationConfiguration
```

```
{
```

```
/**
```

```
 * Initializes the current configuration.
```

```
*/
```

```
public function initialize()
```

```
{
```

```
    parent::initialize();
```

```
    set_include_path($sf_zend_lib_dir.PATH_SEPARATOR.get_include_path());
```

```
    require_once($sf_zend_lib_dir.'/Zend/Loader.php');
```

```
    spl_autoload_register(array('Zend_Loader', 'loadClass'));
```

```
}
```

```
}
```

# Creating a schema/object model

# Install sfGuardPlugin

Enable modules in frontend settings.yml

.settings:

```
enabled_modules: [sfGuardAuth, sfGuardGroup,  
sfGuardUser, sfGuardPermission]
```

# Install sfGuardPlugin

Add remember me filter (filters.yml)

```
security_filter:  
  class: sfGuardBasicSecurityFilter
```

Customize default security actions

```
login_module:      sfGuardAuth  
login_action:      signin  
secure_module:     sfGuardAuth  
secure_action:     secure
```

# Rebuild object model (for new plugins)

```
symfony propel:build-all-load frontend
```

# Creating an administration area

```
symfony propel:init-admin frontend feeds Feed  
symfony propel:init-admin frontend  
feed_categories FeedCategory
```

# Protecting our administration area

- Create local modules (sfGuardGroup...)
- Add a config/security.yml and set credentials

all:

```
is_secure: true
```

```
credentials: [ admin ]
```

# Adding a task to aggregate feeds

- Fetch feeds using sfFeedPlugin
- Cache feeds for display on frontend

# Adding routes

# feeds

feeds\_index:

url: /feeds

param: { module: feeds, action: index }

feeds\_show:

url: /feeds/:name

param: { module: feeds, action: show }

feeds\_admin:

url: /administration/feeds/:action/\*

param: { module: feeds\_admin, action: list }

users\_admin:

url: /administration/users/:action/\*

param: { module: sfGuardUser, action: list }

# Adding caching to our application

- Edit cache.yml

# Testing our application

```
// create a new test browser
```

```
$browser = new sfTestBrowser();
```

```
$browser->
```

```
  get('/feeds/index')->
```

```
  isStatusCode(200)->
```

```
  isRequestParameter('module', 'feeds')->
```

```
  isRequestParameter('action', 'index')->
```

```
  checkResponseElement('body', '/techcrunch/');
```

# Deploying our application

```
symfony project:freeze  
symfony project:deploy
```

# Create a del.icio.us component

Create actions/components.class.php

```
public function executeDelicious()  
{  
    $delicious = new Zend_Service_Delicious  
    (sfConfig::get('app_delicious_username'),  
    sfConfig::get('app_delicious_password'));  
    $this->posts = $delicious->getRecentPosts  
    (sfConfig::get('app_delicious_tag'), sfConfig::get  
    ('app_delicious_max', 10));  
}
```

# Create a del.icio.us partial

templates/\_delicious.php

```
<ul>
```

```
<?php foreach($posts as $post): ?>
```

```
<li><?php echo link_to($post->getTitle(),  
$post->getUrl()); ?></li>
```

```
<?php endforeach; ?>
```

```
</ul>
```

# Where to go from here?

- Read Documentation
- Work through the Askeet advent
- Step by step tutorials
- Finding example code
- Finding help

# Documentation

- The Definitive Guide to symfony
  - <http://symfony-project.com/book/trunk>
  - Released open source 1/29/2007
  - Licensed under GFDL
- API Documentation
  - Good coverage
- Wiki
  - Many useful guides and how to
  - Many user contributed tips

# Askeet Advent

- Askeet.com
  - <http://symfony-project.com/askeet>
- 24 Day Tutorial on
- How to build a real web 2.0 application
- In-depth coverage of all aspects of symfony

# Tutorials

- Askeet
- My first project
- Building an ajaxified drag and drop shopping cart
- Sortable lists using ajax
- Degradable ajax pagination

# Example Code

- Askeet
  - <http://trac.askeet.com>
- Snippets
  - <http://symfony-project.com/snippets/>
- Snipeet
  - <http://trac.snipeet.com>

# Finding Help

- Forums
  - <http://www.symfony-project.com/forum>
- IRC
  - [irc.freenode.net/#symfony](irc:freenode.net/#symfony)
- Mailing List (Google Groups)
  - [symfony-users@googlegroups.com](mailto:symfony-users@googlegroups.com)
  - archived & searchable

# Real World Performance

- Hello World in 10-15ms
  - APC / Syck
- symfony can provide many features
  - disable the ones you will not use
- Build intelligently, cache effectively
- sfOptimizerPlugin / sfSuperCache
- Yahoo! Bookmarks / Y! Answers
- symfony-project.com
  - digg, slashdot, techcrunch, ajaxian

# What does Y! change to scale

- Drop the frontend ORM and push down the stack (SOA)
- Precompiled i18n
- Y! Yslow Rules (frontend performance)



Yahoo! is hiring frontend and backend engineers talk to Marcus Chan for more info.

(so is Education.com and Current Media)

Questions?

Thanks for listening

# The Web Workflow

The **User** asks for a **Resource** in a Browser

The Browser sends a **Request** to the Server

The Server sends back a **Response**

The Browser displays the **Resource** to the **User**