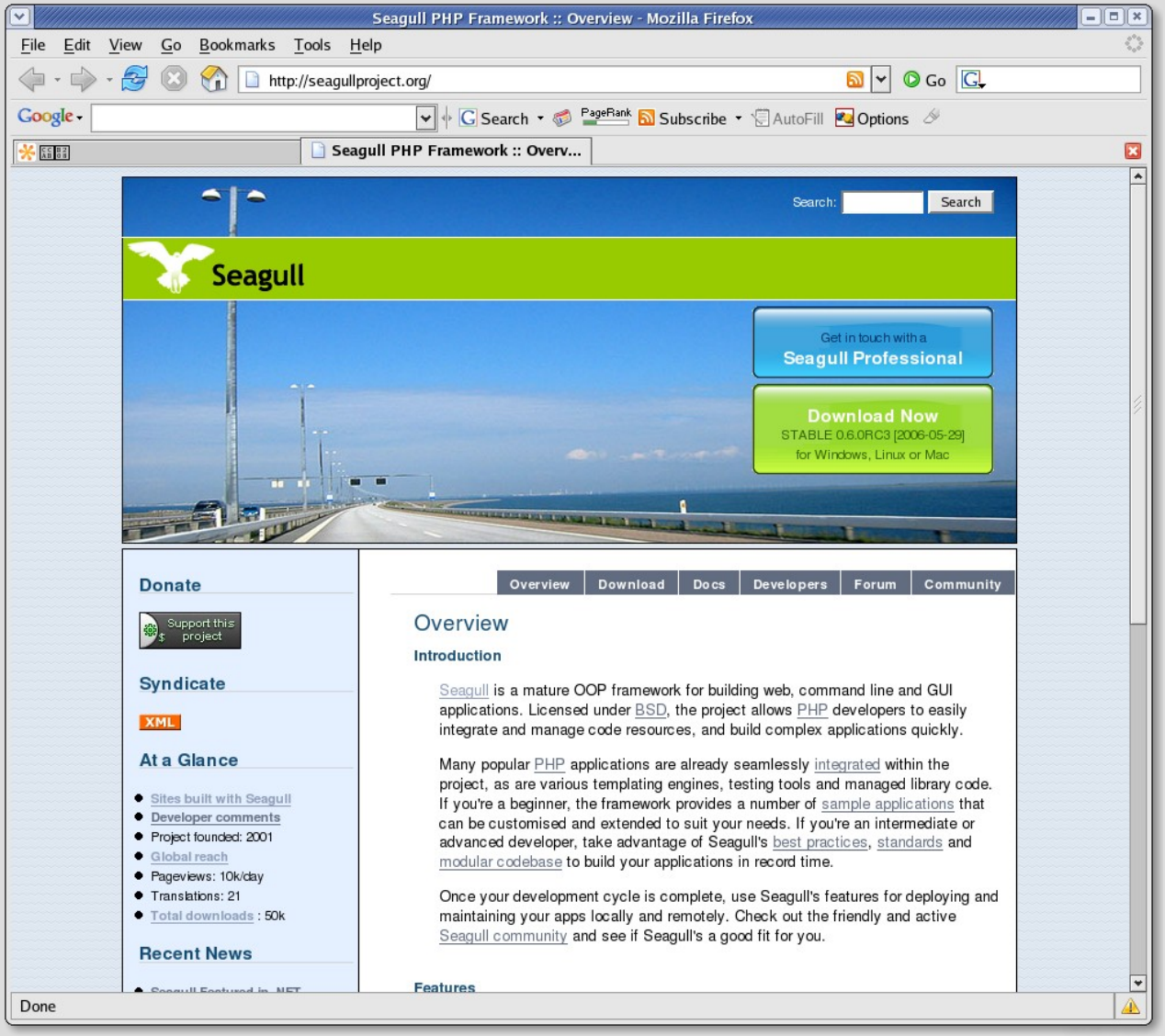
# Working with the Seagull Framework

By Demian Turner, Seagull Systems

# seagullproject.org



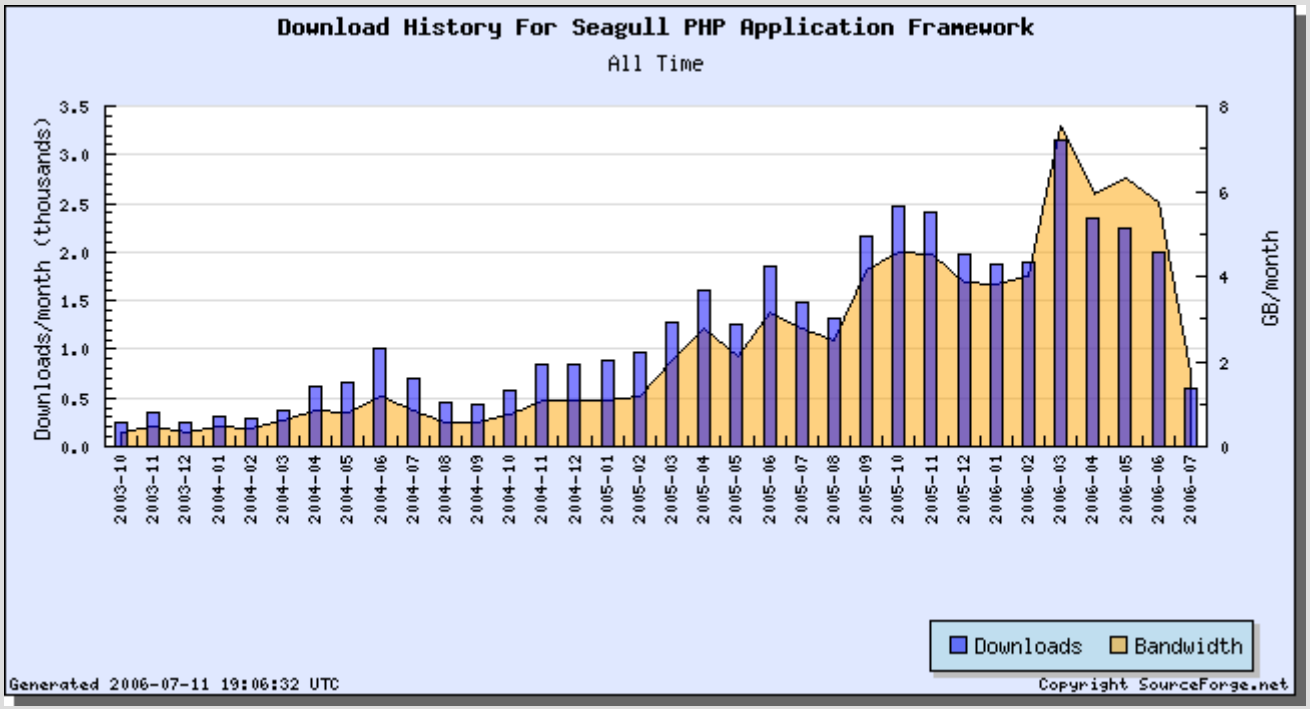
# Who is Demian Turner?

- Developing websites since 1996, using PHP since 1999
- Committer on several open source projects: Seagull, SimpleTest, Max Media Manager
- Involved in PEAR project submitting bug reports and bugfixes
- Presented at and organised few conferences
- Founding member of PHP-London

# Seagull Project

- Open Source framework project
- Started in 2001
- 40 000+ downloads
- 1000 registered users in 72 countries
- 21 translations
- Monthly releases for last 3 years
- Approaching 1.0 release

# Downloads since 2003



# What does Seagull do?

- Rapid development toolkit
- Framework tools and components
- PHP 4/5 compatible
- enterprise ready, scalable, performance conscious design
- integration with PEAR libraries
- User management tools: A & A
- i18n, l10n
- BSD licensed

# Project values

- Software Best practices
- Coding standards
- Web standards
- OOP modular code
- Exploit strengths of PHP while not pretending to be Java, Ruby, etc
- Refactoring
- Unit Testing
- Agile/XP Methodology

# Community

- Mailing list
- Trac/wiki
- Forum
- #seagull
- Seagull Developer Network
- http://frappr.com/seagull
- http://seagullproject.org
- http://seagullsystems.com

# Why use a framework?

- Do you have the sense you're cut and pasting a lot of code?
- Are you grafting together bits of other peoples' solutions, ending up with a mish-mash of coding styles and approaches?
- Have trouble getting new developers on the team up to speed with existing projects?
- Problems dividing project tasks up amongst developers because there was code overlap?

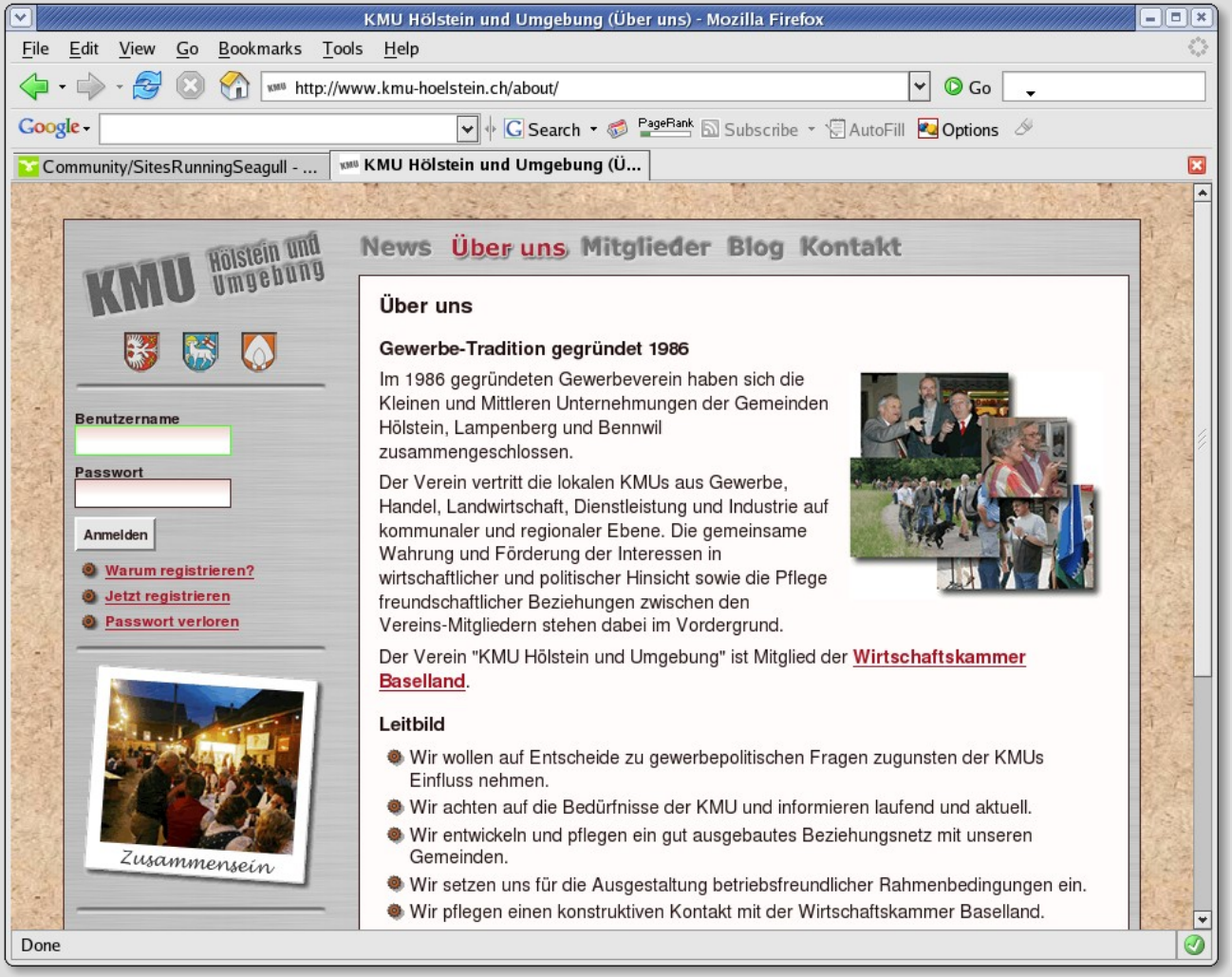
# Seagull framework features

- Minimal requirements
- Simple install
- Modularity
- Quality Control
- Stability of codebase
- Consistency and coding standards
- Standards compliant
- Multi platform/DB vendor support
- Multiple input sources
- Multiple output formats
- Data validation
- Front controller
- Highly configurable
- Third party integration
- Skinable interface
- Caching and performance
- I18N, L10N
- Authentication and authorisation
- Specialised SQL data structures
- Developer friendly
- Designer friendly
- Not developed in isolation
- Open source advantage
- Web based editing
- Included modules

# Example sites /1


http://ernestranglin.calabashmusic.com/

# Example sites /2

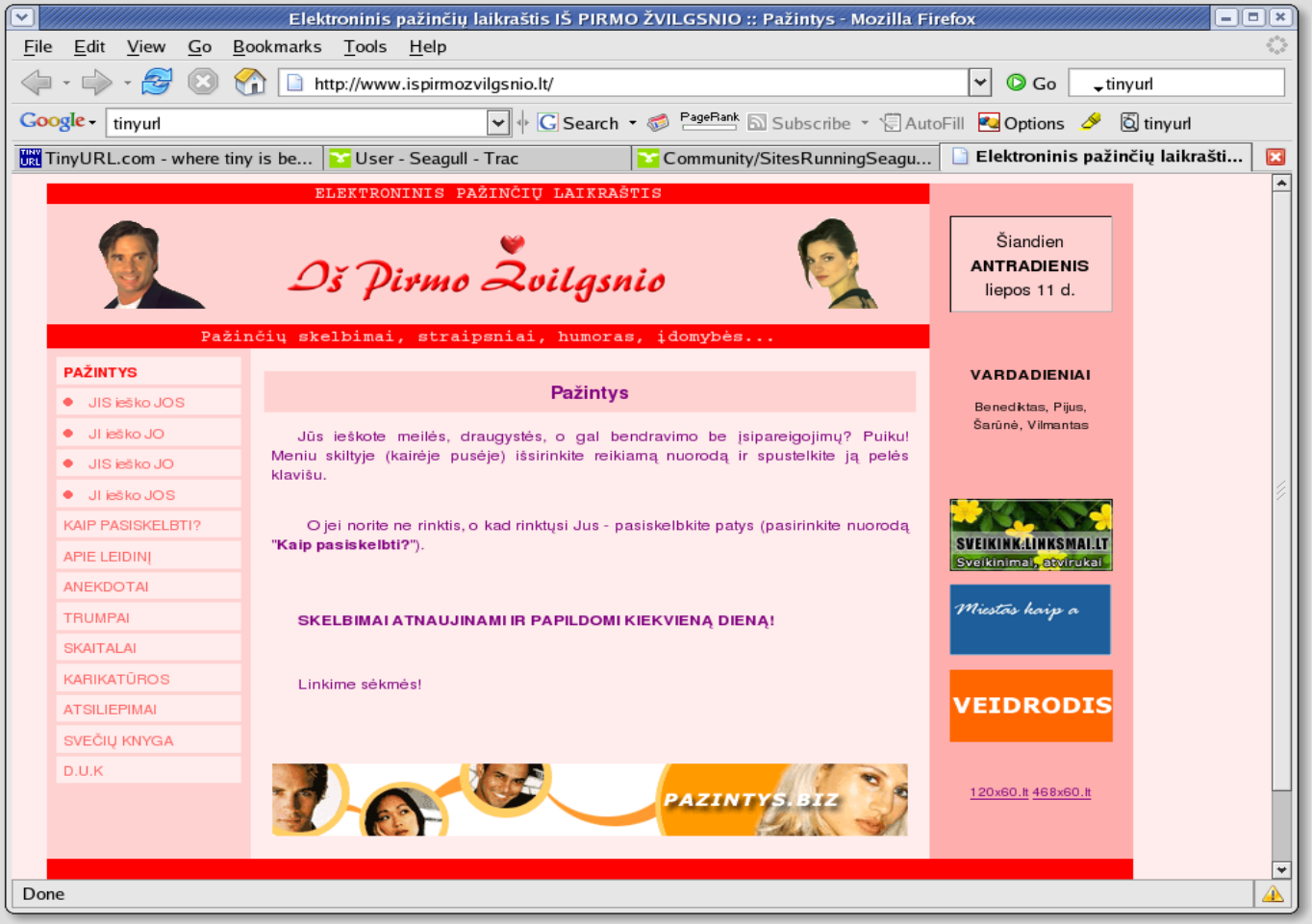



http://www.kmu-hoelstein.ch/

# Example sites /3

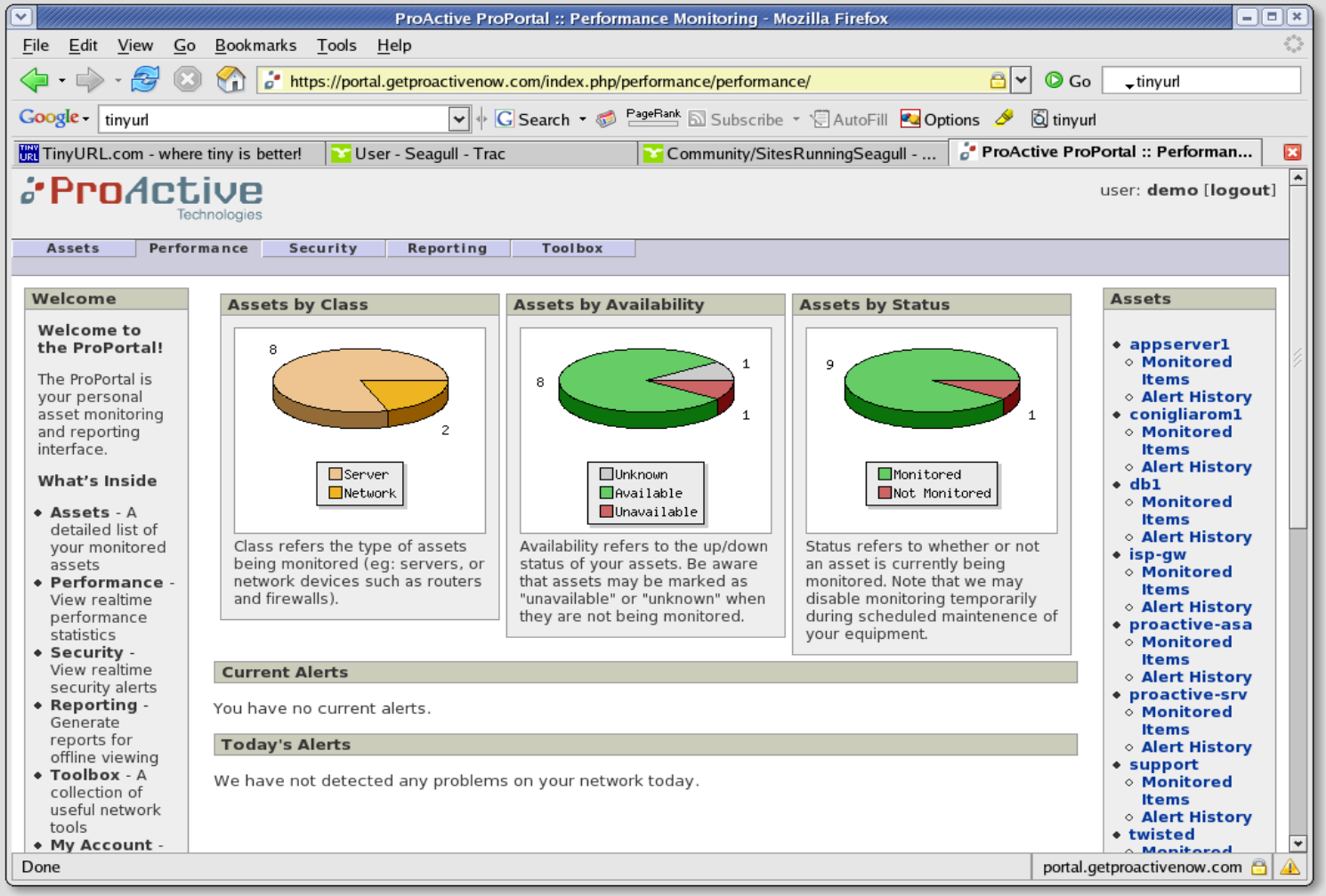

http://www.carroweb.com.br/

# Example sites /4



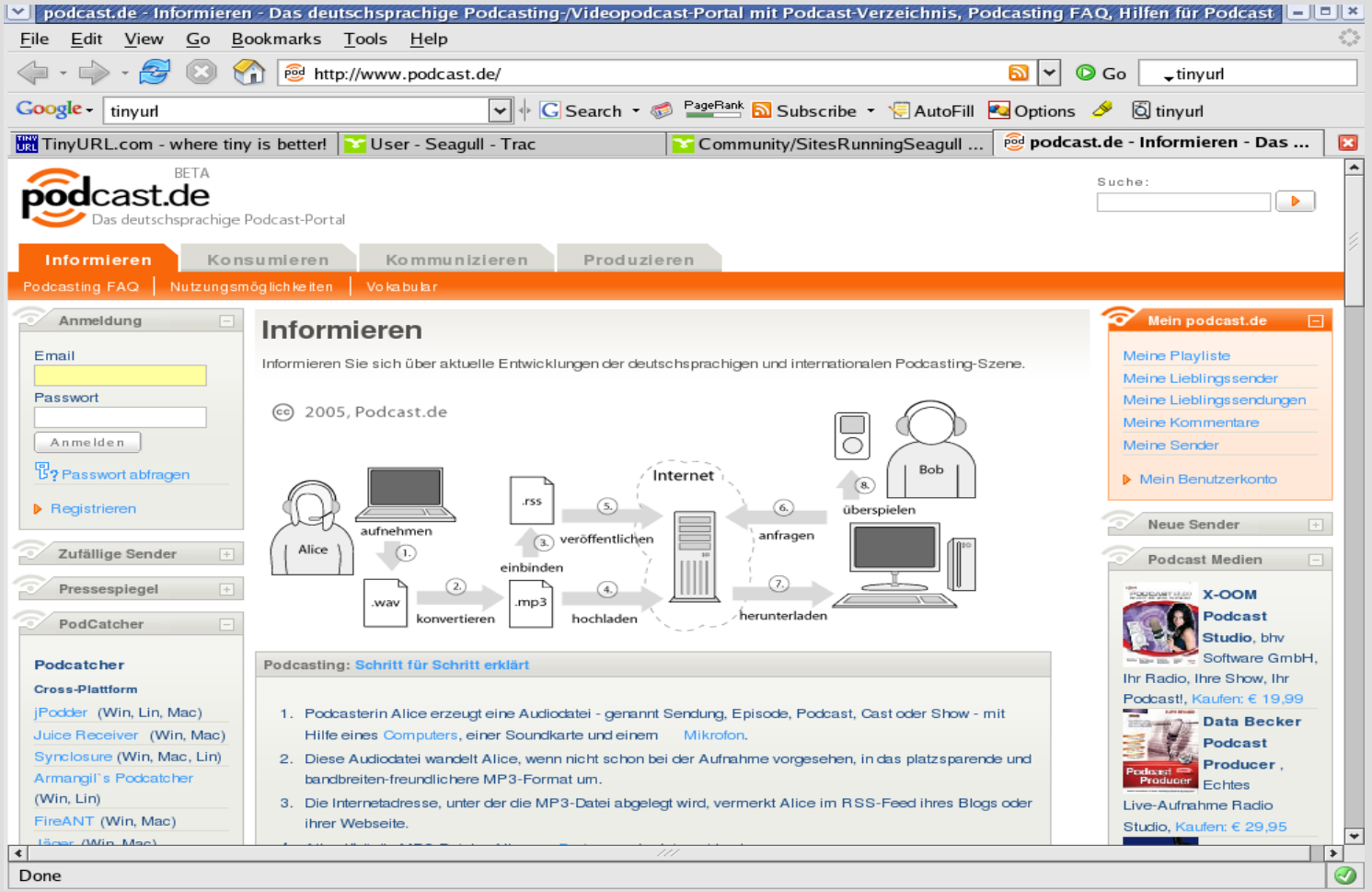

http://www.ispirmozvilgsnio.lt/

# Example sites /5



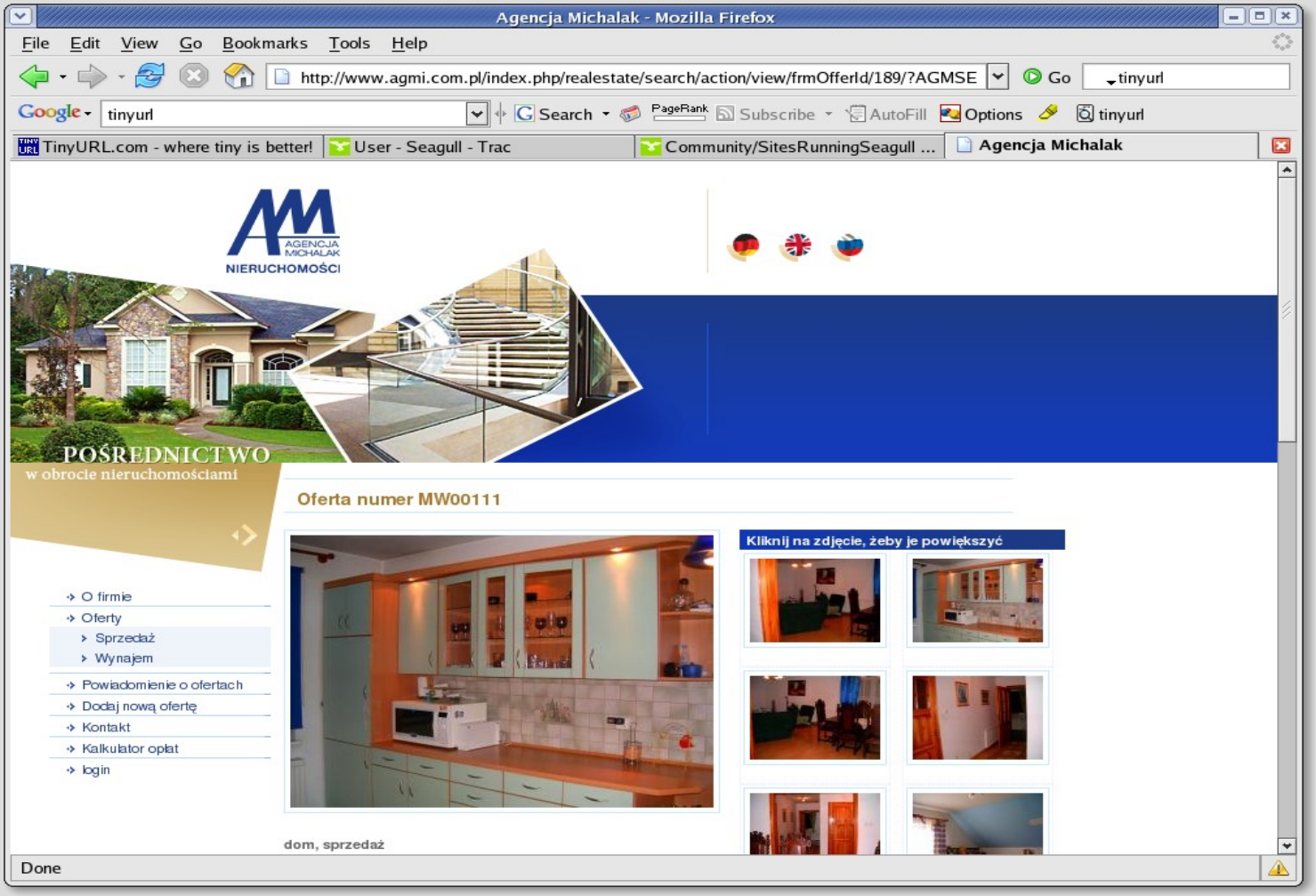

https://portal.getproactivenow.com/index.php/performance/

# Example sites /6




http://www.podcast.de/

# Example sites /7

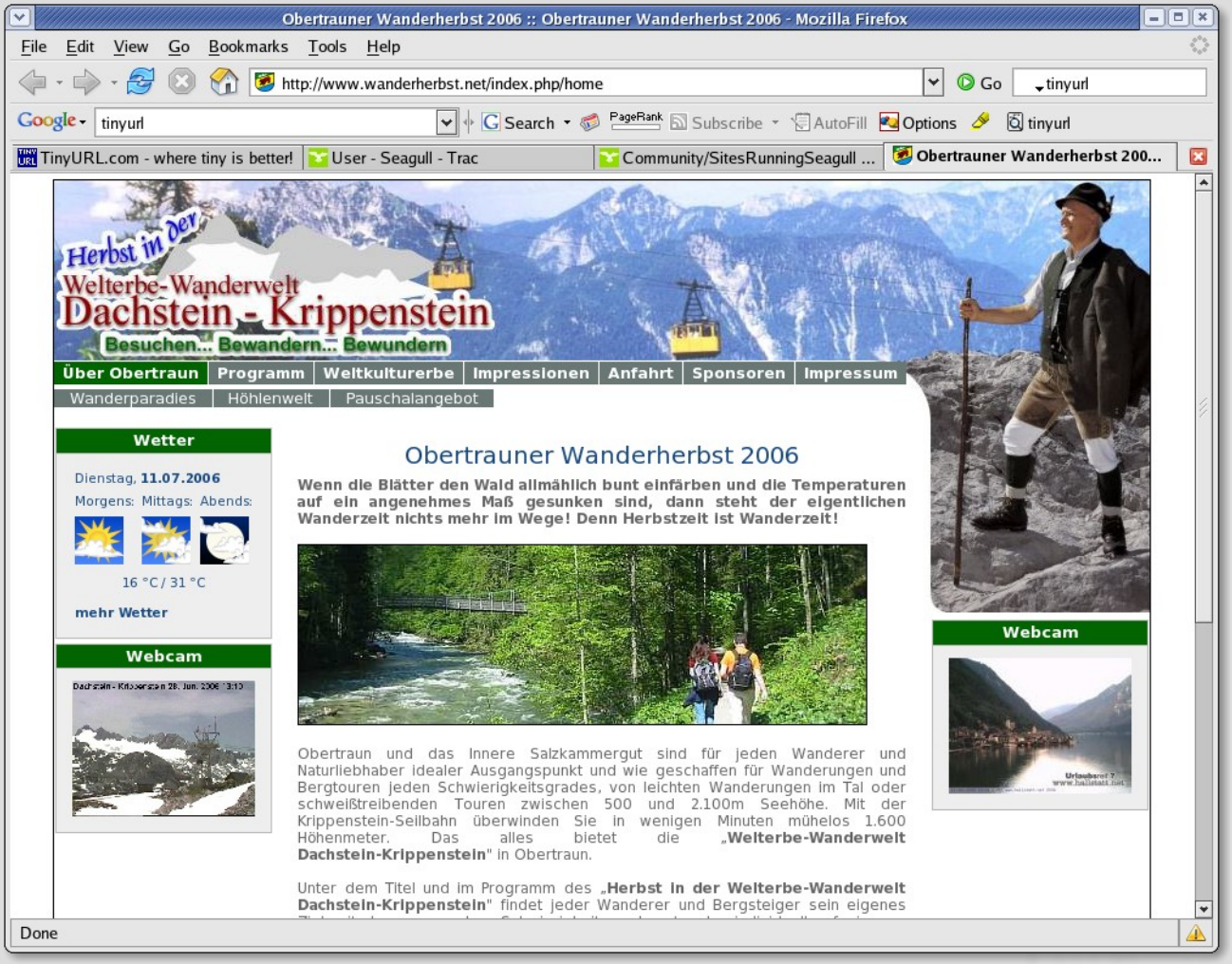



http://www.agmi.com.pl/

# Example sites /8

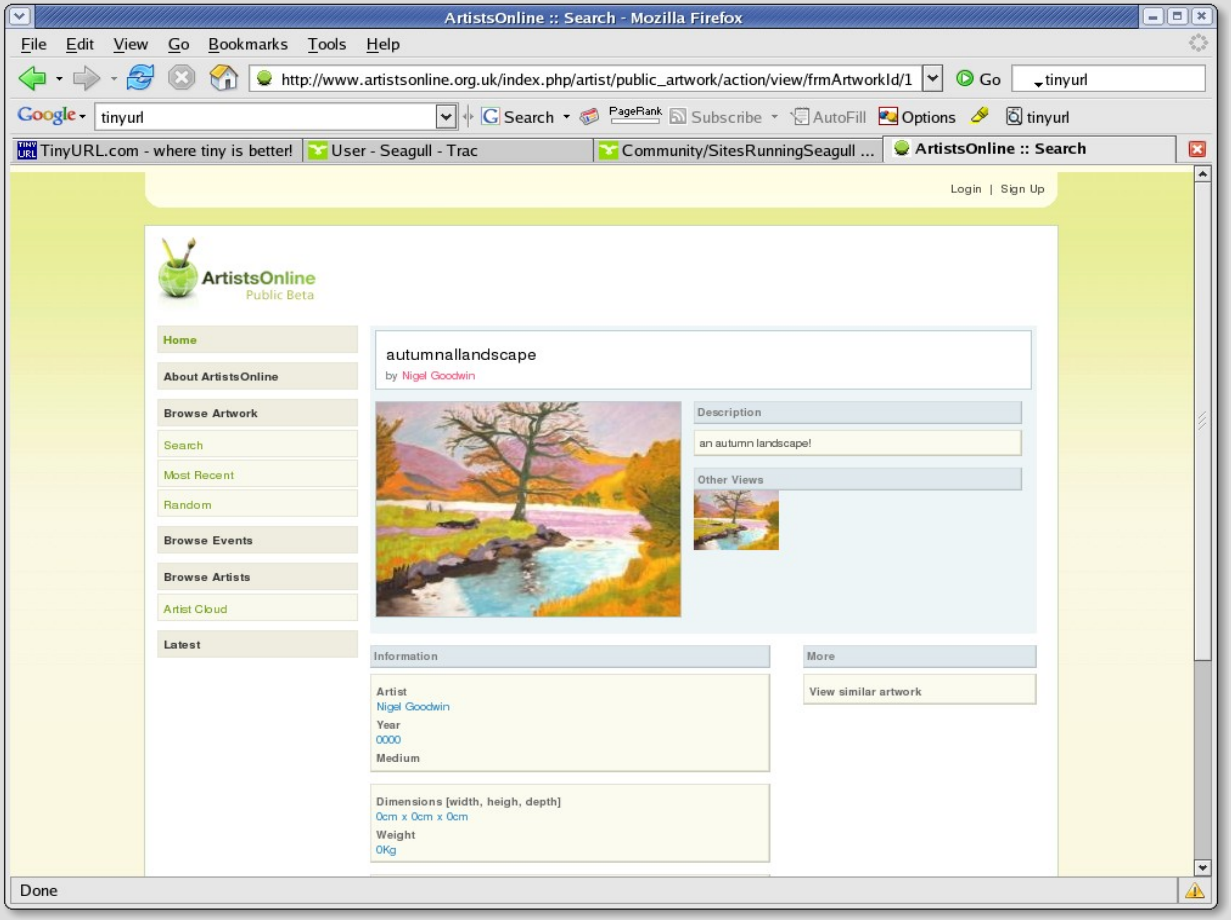

http://www.casanuts.com/

# Example sites /9




http://www.wanderherbst.net/

# Example sites /10




http://www.artistsonline.org.uk/

# Other example sites

- http://www.c17.net/ - library application used by 530 libraries all over Spain
- Power station monitoring app, Slovenia
- http://tinyurl.com/s5utu Shoreditch TV, VOD app, UK

# PEAR Integration

- Project inspired by PEAR
- Seagull acts as glue for PEAR components
- PEAR Installer available since 2000
- Useful for installing:
  - PEAR packages
  - Seagull application
  - Seagull modules
- 44 PEAR libs currently bundled with Seagull
- RAD spirit: focus on completing app ASAP
- Don't reinvent the wheel

# Other libraries

- Seagull integrates favourably with well-designed libraries
- Attention to namespacing, PHP coding standards
- Recommended with:
  - Zend Framework
  - ezComponents

# Testing

- Simpletest package used by default
- Built in test runner
- Tests are easy and fast to develop
- Encourage clean coding
- TDD / XP
- Web tests
- Selenium
- Example user module tests
- Demo

# Example Install

- Install from svn
- Using the installer
- Default settings
- Login as admin

# Quick site example

- Create homepage
- About
- FAQs
- Contact Us
- Register
- Blocks

# Seagull Concepts / Intro

The Seagull project is a set of classes and suggested development approach that aims to make writing programs faster and easier. The belief is that by providing a methodology, and the tools required to handle arbitrary resources and workflow demands, the programmer can dedicate more time and energy to solving the key problems at hand, whether they are refining algorithms, imposing domain-specific logic, creating an attractive and user-friendly user

# Seagull Concepts / Workflow

- the program is initialised by executing an arbitrary list of tasks
- the calling arguments, or request, are analysed and stored
- a configurable set of filters are called on the program input (and later output)
- the event implicit in the request is processed
- Any observers listening to that event are triggered
- the output is returned to the caller

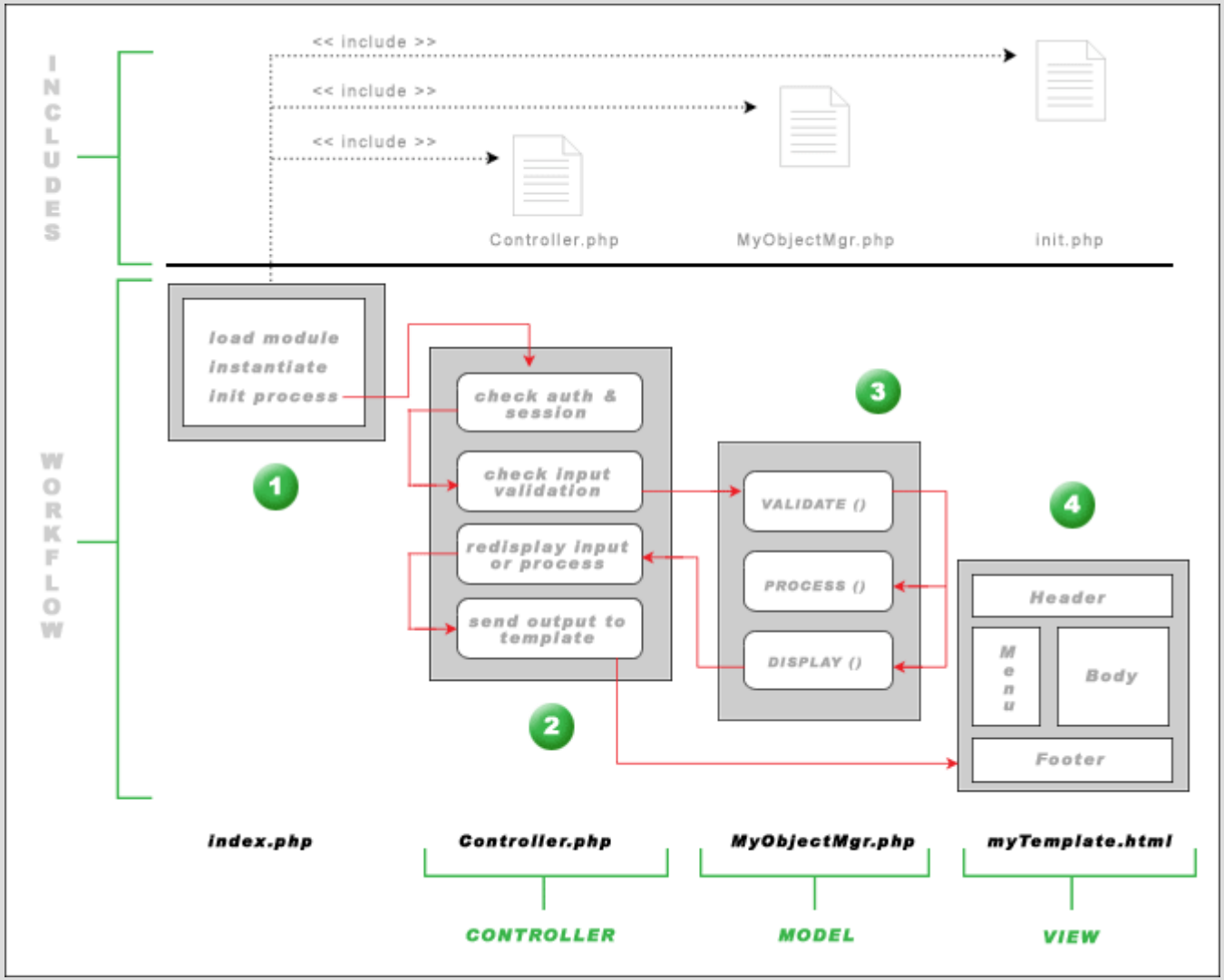
# Seagull Concepts / Tools

- a taxonomy builder and nested set helper
- configuration management
- database abstraction
- object relational mapping
- object delegation
- http upload/download helpers
- an emailer
- error management
- filter management
- front and application controllers
- locale handling
- configurable content objects
- subject/observer mechanism
- a parameter handler
- a registry
- request abstraction
- session handling
- string manipulation
- task management
- db and file-based translation tools
- configurable URI parsing
- a wizard helper

# Library Organisation

- Controllers – workflow
- Core/PEAR libs – general
- Tasks - mostly used to do setup tasks
- Filters - modify the request and response objects as they pass through the filter chain
- Detailed info: http://tinyurl.com/ou6p4

# Workflow Diagram



# Modules, Managers and Controllers /1

The majority of work done by the developer when building as Seagull-based app is in implementing his/her own modules. A module is a logical grouping of functionality. In the User module, for example, you will find code that manages

* users
* preferences
* permissions
* roles

# Modules, Managers and Controllers /2

A module is made up of one or more managers. A manager is simply a page controller object which groups together a number of related actions that can be performed on a single entity or business object, for example a User, a Transaction, a ShoppingCart, etc. Typical actions that a manager controls are:

* add
* insert
* edit
* update
* delete

# Modules, Managers and Controllers /3

**Application Controller:** All managers, or Page Controllers, inherit from SGL_Manager which acts as the Application Controller. The workflow set out in SGL_Manager is a good generic choice for most web applications. However you are not limited to implementing the validate/process/display workflow set out by this class. You could easily create another application controller, and have your page controllers extend that.

# Modules, Managers and Controllers /4

**Exception:** A good example would be in the case of building a REST server, where no template management is needed, no display() method, just XML to PHP transformation on the input stream, vice versa on the output, and data fetching. All permission checks could be replaced by simple HTTP authentication (see http://del.icio.us/help/api/), no session would be required, so you would end up with a very lightweight app.

# Modules, Managers and Controllers /5

**Front Controller:** Seagull also uses another higher level controller that helps simplify code responsibility, the Front Crontroller. This object is responsible for handling all requests from user agents and delegating responsibility to the Filter Chain, which is a simple list of input/output filters and the target Core Processor.

# Modules, Managers and Controllers /6

**Front Controller:** Various filters in the chain build and clean the request object, resolve the relevant manager and action details from the URI, execute the domain logic and finally send the appropriate headers.

The Front Controller is also responsible for initialising the Seagull environment which involves setting up paths, parsing the config, etc.

# Modules, Managers and Controllers /7

- Code example of Front Controller

# Modules, Managers and Controllers /8

- Validate Process Display workflow
- **validate**: the raw $_REQUEST array is passed to this method (wrapped in the SGL_Request object), validations performed, and acceptable data mapped to an $input object

# Modules, Managers and Controllers /9

- **process**
  - *if valid*: if all data is valid, the $input object is passed to the process method, which redirects to the relevant action method derived from URI params.
  - *if invalid*: if one or more validations have failed, the request is deemed to be invalid, and the data passed directly to the display() method with appropriate error messages, ie, to be presented back to the user for correction. The process method is bypassed.

# Modules, Managers and Controllers /10

- **display:** this method offers the opportunity to add some post-processing to the data that all actions have in common. For example, if you have 5 methods that all need to build combobox data for forms, here is the place to do it once for all of them.

# Modules, Managers and Controllers /11

Practical example:

- a contact form is posted with field data deemed to be valid
- business logic executed to dispatch the details via email and save the request in the database
- a confirmation message built for displaying to the user
- the emailSent.html template specified
- the HTML renderer instantiated, using the Flexy template engine as specified in the config
- the template rendered with vars populated from $output data, or loaded from the cache if it exists
- text/html headers generated and response sent to the browser client

# Using Seagull Resources

- Seagull site
- API docs
- Sourceforge
- Wiki / Trac
- Forum
- #seagull

# Working with Trac

- Wiki / tutorials
- Tickets
- Registering
- Submitting a ticket
- Milestones
- Bugs

# Creating a Module /1

- The test assignment will involve creating a Bounty module, duplicating the functionality of the one found at http://www.horde.org/bounties/
- Bounties are an open source tradition and are used by many projects
- Finished result will be used on Seagull project site

# Creating a Module /2

Requirements:

- List of active bounties
- List of suggested bounties
- Suggest a new bounty
- Bounty detail screen
- Sponsor a bounty
- Admin screen (manage statuses)

# Creating a Module /3

- Analysis of requirements
- Develop schema
- Public and admin managers
- Module generator
- Create sample data
- Using seagull-rebuild
- Demonstrate error handling
- Validation
- Pair programming
- Source code management

# Creating a Module /4

Bounty enhancements:

- Config options
- Add captcha support
- Using a data access object
- Implement caching
- Email notifications (observer)
- Require authentication for edits
- Localize
- Write tests