ARCADA

# Developing Advanced Web Applications with the Yii Framework

Sam Stenvall

Degree Thesis

Information and Media Technology

2014

| | |
|---|---|
| EXAMENSARBETE | |
| Arcada | |
| | |
| Utbildningsprogram: | Informations- och medieteknik |
| | |
| Identifikationsnummer: | 4645 |
| Författare: | Sam Stenvall |
| Arbetets namn: | Utveckling av avancerade webbtillämpningar med ramverket Yii |
| Handledare (Arcada): | Hanne Karlsson |
| | |
| Uppdragsgivare: | |
| | |

Sammandrag:

Syftet med arbetet är att beskriva PHP-ramverket Yii med fokus på hur man kan använda det för att bygga webbapplikationer. Arbetet baserar sig på programmet XBMC Video Server som skapats med hjälp av ramverket. Programmet är en fristående webbapplikation som möjliggör strömning och nedladdning av den media som användaren har tillgänglig i XBMC. Programmet har från början publicerats som öppen källkod. I arbetet har jag strävat efter att illustrera ramverkets användning med exempel från XBMC Video Server för att kunna ge en praktisk inblick i hur det kan användas i verkliga livet för mer nischade projekt. Arbetet börjar med en introduktion till Yii-ramverket, hur det har uppstått och vilka alternativ det finns till det. Vidare beskrivs ytligt hur ramverket används rent praktiskt, med fokus på sådana funktioner och teknologier som används i XBMC Video Server. Illustrationer i form av korta kodsnuttar används för att ge en bättre överblick. I arbetets andra del beskrivs i korthet pakethanteraren Composer som är en integral del i XBMC Video Server. I den tredje och sista delen beskrivs XBMC Video Server både från ett tekniskt- och ett användargränssnittsperspektiv. Illustrationer i form av skärmbilder från programmets centrala delar används för att ge en helhetsbild av det som texten beskriver.

| | |
|---|---|
| Nyckelord: | Yii, PHP, ramverk, videoströmning, öppen källkod, XBMC |
| Sidantal: | 61 |
| Språk: | Engelska |
| Datum för godkännande: | 15.12.2014 |

| | |
|---|---|
| DEGREE THESIS | |
| Arcada | |
| | |
| Degree Programme: | Information and Media Technology |
| | |
| Identification number: | 4645 |
| Author: | Sam Stenvall |
| Title: | Developing Advanced Web Applications with the Yii Framework |
| Supervisor (Arcada): | Hanne Karlsson |
| | |
| Commissioned by: | |
| | |

Abstract:

The purpose of the thesis is to describe and evaluate a PHP framework called Yii, with a focus on how to use it to build web applications. The thesis is backed by an application called XBMC Video Server which has been developed using the framework. XBMC Video Server is a standalone web-based web application which enables streaming and downloading of media from XBMC, a popular entertainment center software. XBMC Video Server is published as free software. In the thesis I've tried to illustrate how the Yii framework is used by using examples from XBMC Video Server in order to provide an insight into how it can be used when developing more unique projects. The thesis begins with an introduction to the Yii framework itself, its history and some of the available alternatives to it. Furthermore the framework is analyzed from a practical perspective with focus on features and technologies that have been utilized in XBMC Video Server. To give the reader a better overview, illustrations in the form of short code samples are used. The dependency manager Composer is described shortly in the second part of the thesis, since it's an integral part of XBMC Video Server. In the third and final part the XBMC Video Server application itself is described both from a technical and a user interface standpoint. Illustrations in the form of screen shots are used to give a better picture of the information that the text is trying to convey.

| | |
|---|---|
| Keywords: | Yii, PHP, frameworks, video streaming, open source, XBMC |
| Number of pages: | 61 |
| Language: | English |
| Date of acceptance: | 15.12.2014 |

# CONTENTS

**1. Introduction** 9

1.1 Background 10
1.2 Methods and goals 11
1.3 Constraints 11

**2. The Yii framework** 13

2.1 Overview 13
2.2 Installation and basic architecture 13
*2.2.1 The bootstrap script 14*
*2.2.2 Request routing 14*
*2.2.3 Directory structure 15*
2.3 Components 17
*2.3.1 Application components 17*
2.4 Models 18
*2.4.1 Derived model classes 18*
*2.4.2 Rules and scenarios 19*
2.5 Controllers 20
*2.5.1 Filters 21*
*2.5.2 Layouts and views 25*
*2.5.3 Widgets 26*
2.6 Caching 27
*2.6.1 Data caching 29*
*2.6.2 Fragment caching 30*
*2.6.3 Cache invalidation 31*
2.7 Modules 33
2.8 Extensions 33

**3. Composer** 34

3.1 Introduction 34
3.2 The *composer.json* file 35
*3.2.1 The composer.lock file 36*
3.3 Packagist 37
3.4 Integrating Composer with Yii 38
*3.4.1 Using Composer dependencies as application components 38*

**4. XBMC Video Server** 39

4.1 Background 39
*4.1.1 XBMC 40*

4.2 Functionality overview ..... 41
4.3 Design ..... 42
*4.3.1 The API layer ..... 42*
*4.3.2 Browse pages ..... 45*
*4.3.3 Details pages ..... 48*
4.3.4 Settings and administration ..... 53

**5. Conclusions ..... 56**

**References ..... 59**

**Appendice 1 - Sammanfattning på svenska**

## FIGURES

Figure 1 A skeleton Yii project directory structure ........ 15

Figure 2 A validator method ........ 20

Figure 3 A controller action ........ 21

Figure 4 A method filter ........ 23

Figure 5 An example of how the access control filter works ........ 24

Figure 6 Rendering a widget using CController::widget() ........ 27

Figure 7 Rendering a widget using beginWidget() and endWidget() ........ 27

Figure 8 Configuring cache components ........ 29

Figure 9 How to use data caching ........ 30

Figure 10 Fragment caching in action ........ 31

Figure 11 How cache invalidation can be accomplished ........ 32

Figure 12 A composer.json file in its simplest form ........ 36

Figure 13 A composer.json file which references a private repository ........ 37

Figure 14 Example usage of the simple-json-rpc-client library ........ 43

Figure 15 Example of how to use the VideoLibrary static helper ........ 44

Figure 16 Illustration of how VideoLibrary can map raw objects to a desired class ........ 44

Figure 17 The browse page in the XBMC Video Server application ........ 46

Figure 18 the Recently added episodes page ........ 48

Figure 19 A details page for a movie ........ 49

Figure 20 Watch/Download dialog ........ 50

Figure 21 Grid view of the available seasons for a TV show ........ 51

Figure 22 Example of an expanded season listing all the episodes for that season ........ 52

Figure 23 Partial view of the application settings page ........ 54

Figure 24 The system log ........ 55

## TERMS AND ABBREVIATIONS

| | |
|---|---|
| **AJAX** | Asynchronous JavaScript and XML |
| **APC** | Alternative PHP Cache |
| **API** | Application programming interface |
| **CMS** | Content management system |
| **Codeception** | A testing framework for PHP |
| **CRUD** | Create, read, update and delete |
| **CSS** | Cascading Style Sheets |
| **FFmpeg** | A set of libraries and programs for handling multimedia data |
| **HTPC** | Home Theater PC, a computer dedicated to watching media |
| **IMDb** | Internet Movie Database |
| **JSON** | JavaScript Object Notation |
| **JSON-RPC** | JavaScript Object Notation Remote Procedure Protocol |
| **M3U** | A file format for multimedia playlists |
| **Memcached** | In-memory key-value store for small chunks of arbitrary data |
| **MP4** | MPEG-4 Part 14, a video and audio container format |
| **MVC** | Model-View-Controller |
| **OpenELEC** | Open Embedded Linux Entertainment Center |
| **PHP** | PHP: Hypertext Preprocessor |
| **phpdoc** | A special code comment block that describes e.g. a method definition |
| **phpass** | A PHP library for implementing proper password hashing |
| **PDO** | PHP Data Objects, a database layer abstraction |
| **PLS** | A file format that stores multimedia playlists |
| **PSR** | PHP Standards Recommendation |
| **RAR** | Roshal ARchive |
| **SQL** | Structured Query Language |
| **SQLite** | An SQL database embedded in a single file |
| **TCP** | Transmission Control Protocol |
| **WebSocket** | A protocol that provides full-duplex communications channels over a single TCP connection |
| **XBMC** | XBMC Media Center, a popular open source media center |
| **XSPF** | XML Shareable Playlist Format |

## ACKNOWLEDGEMENTS

I would like to thank the XBMC developer community for creating their media center application, without which the application developed for this thesis would not have been possible. Especially I'd like to thank Sascha Montellese (@Montellese) and Tobias Arrskog (@topfs2) for their crucial work on implementing the JSON-RPC API that XBMC uses since without it this application wouldn't be possible.

In addition I would like to thank Christoffer Niska for being sort of a mentor over the years; he's been of tremendous help in learning how to use the Yii framework. He has also written some of the components used in my application, which have saved me and countless others from having to write them ourselves.

Finally I'd like to thank my mentor Hanne Karlsson for assisting me in writing this thesis.

# 1. INTRODUCTION

The purpose of this thesis is to examine the *Yii* framework, how it is used, and present an example of an application made with it. As a part of this thesis I have created a web application called *XBMC Video Server* [1], which is a supplementary tool to access media from the widely used *XBMC* media center software. The idea is to give the reader enough knowledge about the architecture and usage of the Yii framework to be able to see its merits. The application I've chosen to portray fits this purpose well since it's relatively unique.

The thesis is divided into three distinct parts. The first part shows how the Yii framework works, how its various components fit together, and how they can be used. When appropriate, examples from the XBMC Video Server application are used. The second part describes a dependency manager known as *Composer* [2]. While technically not part of the Yii framework, I have chosen to include it here since it is a very integral part of PHP development nowadays, including development of the XBMC Video Server application.

The third and final part will showcase the XBMC Video Server application itself. Beginning with some background information on what it is used for, I will illustrate some of the underlying architecture and design as well as a general overview of how the application works, what features it has, and so on.

Last but not least, I will reflect on the properties of the Yii framework as well as on the XBMC Video Server application.

Due to the nature of the subject, practically all my sources are web based. There are a few books about the Yii framework [3], but the main reference guide has always been its website [4]. Since citing electronic sources using the Harvard system may be confusing to the reader I've chosen to forgo that in favor of the ISO 690 numerical reference system with square brackets [5].

This thesis uses a couple of typographic conventions to distinguish among various kinds of text:

- Code lines, commands, method and variable names appear in a `mono-spaced typeface`

- Placeholders in syntax descriptions appear surrounded by `<chevrons>`
- *Italic type* is used for new terms as well as for file names and paths

## 1.1 Background

The methods and tools used to create web applications have changed and improved drastically over the last few years. It is no longer common to design a web site completely from scratch – not even for small projects. Instead web developers have come to rely on various frameworks (among other things) to ease development of both small and large scale web applications.

There are three main groups of frameworks used today. The framework best suited depends on the task at hand and the amount of customization the developer is looking to make.

The first group consists of various fairly simple *CMS* systems. These include frameworks such as *WordPress* and *Joomla* [6]. They make it easy to quickly set up new web sites like blogs, company pages and information portals. There are many third-party plugins and extensions that a developer may decide to integrate into the application to enhance or extend the framework's functionality.

Not all CMS systems are simple company or portfolio pages. For more advanced requirements many people opt for more complete CMS frameworks, of which *Drupal* is arguable the most famous one [7]. Frameworks like Drupal can be used to create almost any kind of web site, be it a simple blog or a fully featured multilingual web store.

Last but not least there are frameworks that are purely code, which means they don't provide any point-and-click functionality like the previous types of frameworks. Popular frameworks in this category include *Laravel*, *Symfony* and Yii [8]. These framework types are the most powerful and flexible of all since the developer is in full control. Coding frameworks are often used for more specialized web applications where you'd have to do a lot of coding yourself no matter which framework you decide to use. They also tend to be the most efficient in regards to both resource usage and performance since there is less overhead than in running a complete CMS solution underneath the surface.

## 1.2 Methods and goals

XBMC Video Server is published on *Github* as free software under the *GPL v3* license. The idea behind this is to be able to receive potential code contributions from the community as well as to function as a showcase of what I've accomplished as a developer.

Since the thesis is mainly about how to use the Yii framework (which the application is built upon), XBMC Video Server is used as a source of examples on how to use the various features available in the framework. Since the source code is freely available, the concepts explained in the thesis can easily be studied in more detail through code that's actually in use somewhere. The original idea was to use XBMC Video Server purely as an example of how the Yii framework can be used to build advanced web applications, though over time the application grew to such an extent that I decided to take the opportunity to use this thesis as a form of show case for it.

After reading the thesis the reader should be familiar enough with the Yii framework to be able to examine and understand on a higher level the source code of XBMC Video Server or any other moderately complex Yii application. This means that this is by no means a complete guide on how to use the Yii framework.

## 1.3 Constraints

The Yii framework is a so called "full stack" framework. That means that it provides all kinds of functionality, all of which are unlikely to be used by a single project. In the case of XBMC Video Server this is even more true since it doesn't use a traditional database for most of its data models, something which is generally common in web applications and which Yii provides a lot of functionality for. This means that this thesis properly covers only the parts that the application uses. Specifically, the following major concepts are not discussed in greater detail:

- The active record patterns
- Component events and behaviors
- Modules and extensions

On the subject of caching, the data caching mechanism will receive most attention, followed by fragment caching which will be explained briefly. Page caching will not be discussed since it is often achieved using different means, such as dedicated HTTP caching software like *Varnish* [9].

Yii has support for unit testing using *PHPUnit* [10], nevertheless testing will not be covered in this thesis since the subject is too broad for this context.

There's a new major version (2.0) of the Yii framework currently under development [11]. This version modernizes the framework drastically, making it more modular and fixes many of its current issues. However, this thesis will focus only on the current stable release (version 1.1) since that's what the XBMC Video Server application has been developed with, although in the conclusions section I will reflect a little bit on some of the areas that are likely to improve with the new version of the framework.

# 2. THE YII FRAMEWORK

This chapter is in largely based on the official guide to using the Yii framework [12], which is why there won't be any references to it in the chapter itself. The examples, however, are original and at times taken directly from the XBMC Video Server application (depending on the line count).

## 2.1 Overview

The Yii framework is fairly new and has undergone many fundamental changes since its inception in 2008. It was originally created by Qiang Xue while he was working on the *Prado* framework (another PHP framework at the time). After about a year the work spent on the project materialized in the first public release, version 1.0. Through user and developer feedback some changes to the core architecture were made which after some time resulted in the 1.1 release, which still today remains the stable and current version of the framework. It is also this version that this thesis is focused around. [13]

Since 2012 there has been ongoing work on a new major version which will essentially be a rewrite of the whole framework based on modern concepts such as namespacing, something that is relatively new in PHP [14] and not available at the time when Yii 1.1 was created (since Yii 1.1 is targeted at PHP version 5.1). Version 2.0 brings many improvements to the code base, notably it better integrates with Composer, it has improved AJAX support, it uses *Codeception* for unit testing, and database management has become more powerful [11].

## 2.2 Installation and basic architecture

In order to start developing applications with the Yii framework it has to be downloaded and included somehow. One way to do this is to download the complete framework as a ZIP file from the framework's web site [15]. After unzipping the files to a location of choice, a skeleton web application can be created using the `yiic` command line tool. Yii can also be installed as a Composer dependency. The framework is published on

*Packagist* under the name `yiisoft/yii` [16]. Chapter 3 contains more information about Composer and Packagist in general.

### 2.2.1 The bootstrap script

When developing a web application with Yii, the whole application is contained within the Yii execution path since all requests are usually routed to *index.php*, which functions as the *bootstrap* script. The purpose of the bootstrap script is three-fold:

- Include the framework. In Yii this is a file named *yii.php*, which includes the rest as necessary.
- Read the project configuration file (which is required in order to set up the Yii application) into a variable
- Run `Yii::createWebApplication($config)` which starts the application and continues handling the request

### 2.2.2 Request routing

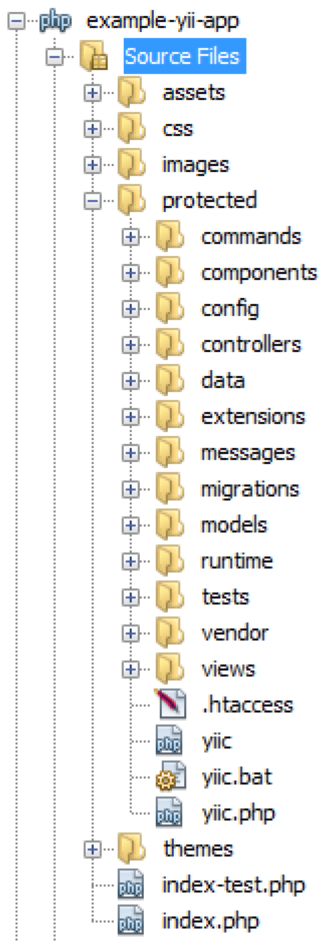
When the application is started a web application instance is created. The application instance is a singleton which is available at all times using `Yii::app()`. The application determines the route that should be executed based on automatic and/or user-defined URL mappings. In the standard configuration a typical request URL might include *index.php?r=user/profile*, where the *r* parameter holds the desired route.

Routes in Yii come in two forms; *controller/route* and *module/controller/route*. In the first variation the controller is assumed to belong to the application itself (which is typically the case) while in the second variation an application module is specified, which means that the controller is part of the specified module, not of the application. While modules will not be covered deeply, there is a short section on them in chapter 2.6.

Once the request has been parsed and a route has been determined, the execution is handed over to the specified controller action (see chapter 2.4).

### 2.2.3 Directory structure

A Yii project created using the `yiic webapp` command will have a default directory structure, as illustrated in Figure 1:



```
example-yii-app
    Source Files
        assets
        css
        images
        protected
            commands
            components
            config
            controllers
            data
            extensions
            messages
            migrations
            models
            runtime
            tests
            vendor
            views
            .htaccess
            yiic
            yiic.bat
            yiic.php
        themes
        index-test.php
        index.php
```

*Figure 1 A skeleton Yii project directory structure*

At the highest level there is the bootstrap script (*index.php*), the *assets/* directory (which is where Yii publishes certain files, such as scripts and style sheets), *css/* and *images/* which are where publicly accessible files like style sheets and images should be placed, and last but not least the *protected/* directory.

The application files are located in the *protected/* directory. The absolute path to the directory can be accessed in code through `Yii::app()->basePath` and it is not supposed to be accessible through the web server [17]. This access to the folder is usually restricted through the use of an *.htaccess* file. Restricting access to *protected/* is important since it's common to store some sensitive data in it, such as passwords, e-mail addresses or API tokens. Storing such files in a web-accessible path is not normally a problem, but if the files use a file extension unknown to the web server (such as “.inc”), the file can be treated as plain text and thus served to the browser, revealing all of its contents.

Inside the *protected/* directory is a bunch of different subdirectories. The structure here is not set in stone (some of the directories created by the `yiic webapp` command are even empty), meaning the developer can modify it to better fit the application structure. There are however a few directories that most projects use.

The *config/* directory contains the configuration file for the application. The file is usually named *main.php* and is parsed and passed to `Yii::createWebApplication()` in the bootstrap script. The configuration file is generally used to configure the various application components in Yii (more on that in chapter 2.3.1) and the directory paths that should be scanned when attempting to include PHP files. By default, Yii looks in *protected/components/* and *protected/models/*, but it is easy to extend this by modifying the `import` section in the configuration file.

### *Framework files*

As mentioned earlier, the Yii framework is used by including the *yii.php* file, which includes the rest of the framework code as necessary. As can be seen in Figure 1, the actual framework files are seemingly nowhere to be found. There are a couple of good reasons for this:

- the framework files should not be accessible through a browser, so it makes sense to put them in a directory not available to the web server
- it is a good idea to keep the framework code out of the project's source control repository since it is big and technically not part of the project's source code

As mentioned in the first chapter, the only place where the location of the framework code is actually needed is in the *index.php* bootstrap script.

Since Yii was invented before the advent of namespaces in PHP (starting with version 5.3) it exposes all its files under the root namespace. To avoid naming collisions, all classes and files are prefixed by the letter C, except the static class `Yii`. The class contains some various getter methods and various helpers, of which the most important are `Yii::app()` which returns the `CApplication` instance and `Yii::t()` which is used to make strings translatable.

## 2.3 Components

The smallest building block in Yii is a *component*. A component is anything that extends `CComponent`. `CComponent` is a thin base class which provides automatic getters and setters, event handling functionality, and behavior functionality (not discussed in this thesis). While a bare component is not very useful, a particular subclass of components called *application components* is an integral part of the Yii framework.

### 2.3.1 Application components

An application component is anything that extends `CApplicationComponent` or implements the `IApplicationComponent` interface. Application components are classes that act like singletons and are accessible through `Yii::app()->component`. Application components are specified using the main configuration file.

The use of application components is very common in Yii applications since Yii comes with a fairly large set of often needed components which are available to the developer. Some of the most commonly used ones are:

- `CClientScript`, accessible via `Yii::app()->clientScript`. This class handles registering JavaScript and CSS files and snippets. Once a page is rendered, the client script component injects the HTML for the included scripts and styles in the <head> section of the document. This means that the developer doesn't have to include all the required scripts and styles in the application layout, instead it can be done dynamically depending on when they're needed.
- `CWebUser` – accessible via `Yii::app()->user`. This object represents the current user on the web site, be it a guest or an authenticated user. One of the most common uses of this class is to display a one-time message on the page, such as a notification after a user has successfully authenticated. This can be

conveniently done through `Yii::app()->user->setFlash(<level>, <message>)`.

- `CDbConnection` – accessible via `Yii::app()->db`. This object represents a connection to a database and can be used to execute queries and retrieve results. It is also used behind the scenes when using database abstraction layers like the active record model, described shortly in the following chapter.

## 2.4 Models

All model classes in Yii extend from the common `CModel` class. Any model is by definition also a component since `CModel` extends `CComponent`. This means all models are able to use magic getters and setters, something that can be quite useful. The `CModel` class is the most basic type of model, which means it doesn't provide much functionality. Its main purpose is to provide *validation* and *scenario* support.

### 2.4.1 Derived model classes

Apart from `CModel` itself there are two base classes for models, `CActiveRecord` and `CFormModel`. `CActiveRecord` is used to represent both a single database record as well as the database table in question. It provides an easy to use object-oriented interface for performing CRUD operations on a database, as well as handling relations between different models. This means the developer does not have to write any SQL commands in order to perform basic tasks, something which normally is quite time-consuming and prone to error.

`CFormModel` is as its name suggests designed to model forms on a web page. Form models are very useful for validating form input, especially if the form does not represent a single database model. If a form only contains fields for one specific database model there is no immediate benefit to using a form model instead of the database model directly since the validation logic would most likely be duplicated. Nevertheless, form models are often used to acquire input from the user that is not tied directly to any particular model.

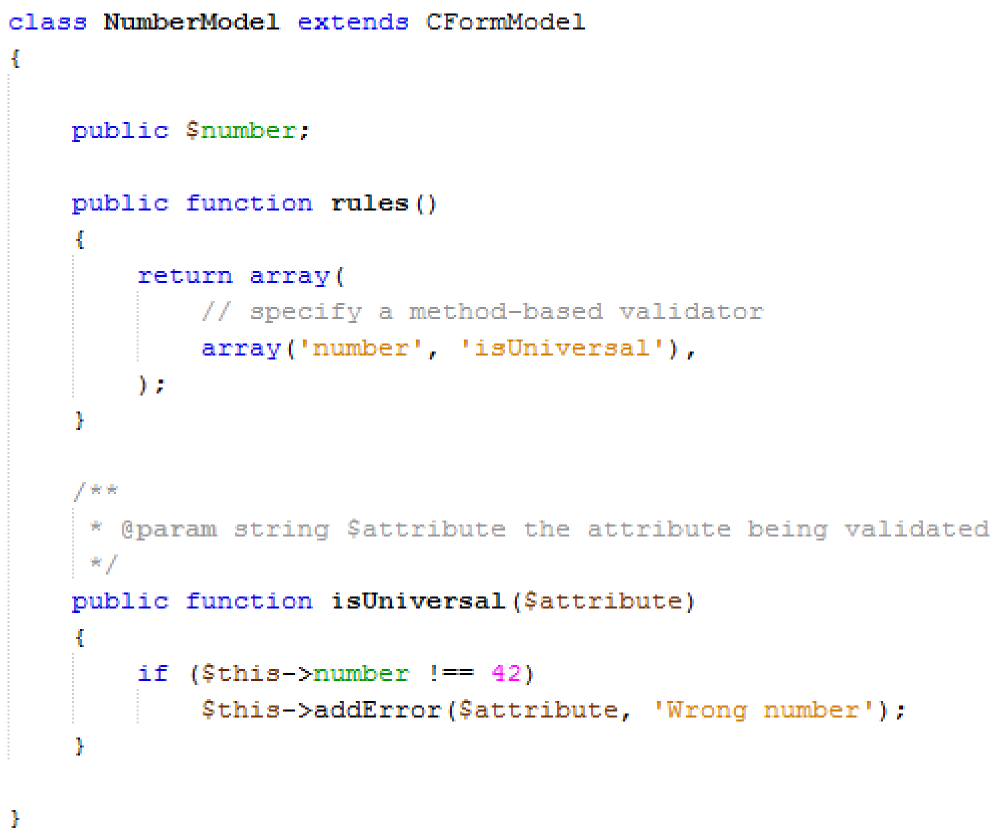
### 2.4.2 Rules and scenarios

Yii has a concept of validation rules and scenarios when validating models. Each model *attribute* can have zero or more validation rules associated with it. A validation rule can e.g. define that a “username” attribute must be unique (that means that there can be no other model in the same database table with that value) and not empty, or that a particular attribute must be an integer-only number.

The model rules are used when validating a model. This is usually done by calling `CModel::validate()` or `CModel::save()`. Saving a model thus validates the model before it is saved, and only if the validation passes is the model actually saved. If validation fails, the list of errors is returned from `CModel::getErrors()`. The return value from this method is used by form helper classes to indicate to the user why a form field didn’t validate.

A *validator* in Yii can either be one of the built-in validators, a class validator (that is any class that extends the base `CValidator` class), or a validator method (a class method that takes two arguments, `$attribute` and `$params`).

In Figure 2, a simple form model is shown. The model has one attribute, `number`, which is validated using the `isUniversal()` method validator. The validator uses the `CModel::addError()` method to signal that the only value of `number` to pass the validation is 42.



```
class NumberModel extends CFormModel
{

    public $number;

    public function rules()
    {
        return array(
            // specify a method-based validator
            array('number', 'isUniversal'),
        );
    }

    /**
     * @param string $attribute the attribute being validated
     */
    public function isUniversal($attribute)
    {
        if ($this->number !== 42)
            $this->addError($attribute, 'Wrong number');
    }

}
```

*Figure 2 A validator method*

A validation rule can limit itself to a specific scenario. By default, Yii uses two different scenarios for models, “insert” and “update”. The scenario is set to “insert” if a model is considered new (it has not been stored in the database yet); otherwise it is set to “update”. This can be used to perform different validation based on other factors. For example, a “User” model can require the password field to be repeated once when the password should be changed while not requiring that when the model is first created.

For a more complex example of how model validation can be used, refer to the `Backend` model in XBMC Video Server [18].

## 2.5 Controllers

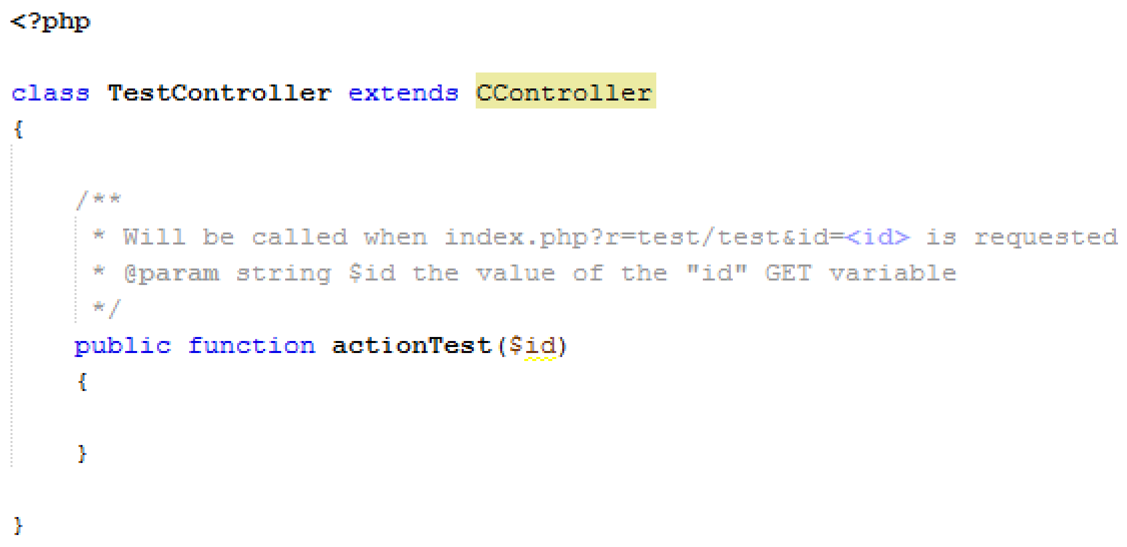
*Controllers* are the heart of any Yii application. There are a couple of things that determines when a class becomes a controller:

- it extends the `CController` base class

- its filename ends with "Controller"
- it is placed in the *protected/controllers* directory

When Yii receives a web request it automatically runs an action depending on the requested route. As described in section 2.1.2, a route has the form *controller/action* or optionally *module/controller/action* (see the "Modules" section later in this chapter). This means that the route "user/login" would trigger a "Login" action in the class `UserController`, which corresponds to the file *protected/controllers/UserController.php*.

An action is simply a public class method which name begins with "action", e.g. `actionLogin()`. If the method defines any parameters the values are mapped from the request URL. If a parameter is not defined with a default value, Yii will throw a HTTP 400 Invalid Request exception if the parameter is missing from the URL. Figure 3 shows how a controller like this can be implemented.



```php
<?php

class TestController extends CController
{

    /**
     * Will be called when index.php?r=test/test&id=<id> is requested
     * @param string $id the value of the "id" GET variable
     */
    public function actionTest($id)
    {

    }

}
```

*Figure 3 A controller action*

### 2.5.1 Filters

Sometimes it is necessary to perform some checks before running a controller action, or perform some sort of cleanup after the action has executed. This is where *filters* come into play. When an action is executed, Yii checks for any filters defined in the requested controller. Filters are defined by overriding `CController::filter()` and returning an array representing the filter configuration for the controller.

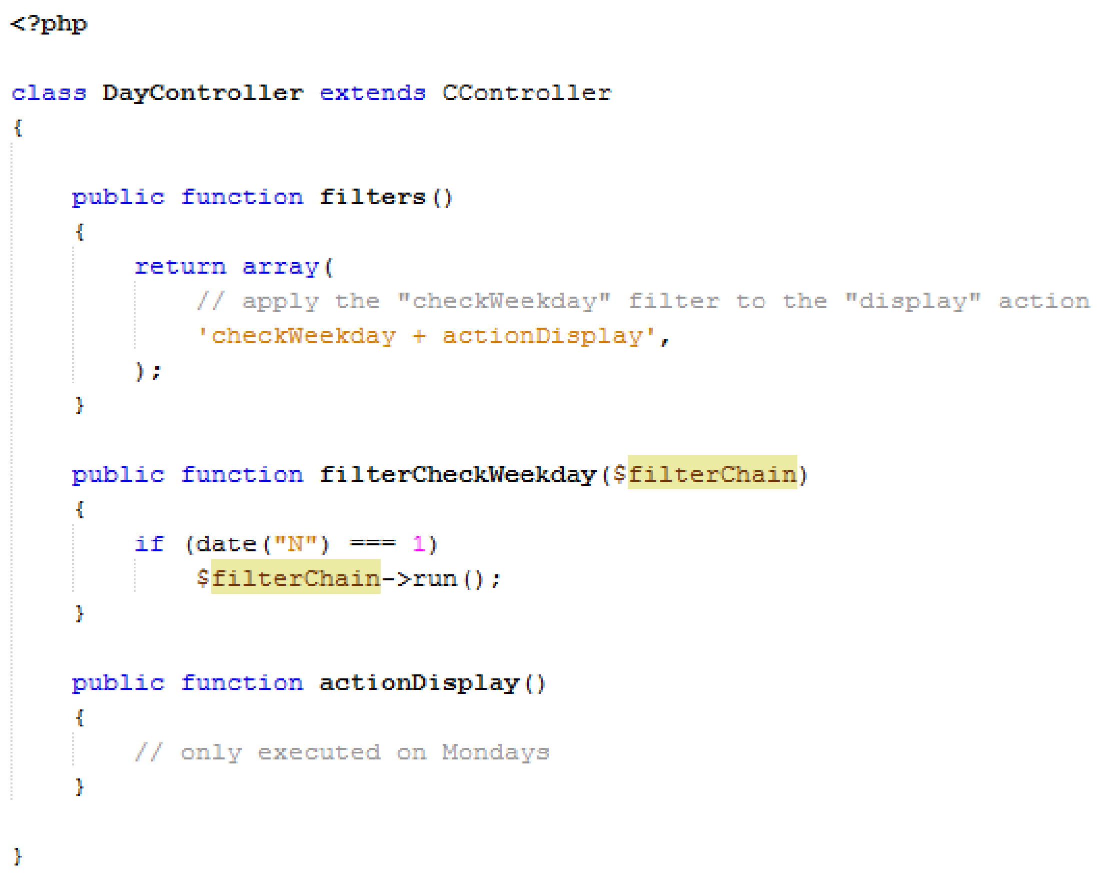
A filter is a piece of code that runs either before the action (called a *pre-filter*) or after the action (a *post-filter*). Filters are useful for factoring out common prerequisite code from the various actions to avoid code duplication. Yii ships with a few built-in filters but also gives the developer the ability to define own filters. The following subchapters describe the filter types and some of the built-in filter types in more detail.

While filters like access control usually apply to all actions in a controller, some filters may only apply to a single action. This is not an issue since the granularity of each filter can be controlled.

### *Method filters*

The simplest type of filter is a *method filter*. Like the name implies it is simply a method in the controller class, prefixed with the word "filter". Method filters are always prefilters, i.e. they always run before the action is executed. Every method filter should take a single `CFilterChain` parameter. If the action should be executed, a call to `CFilterChain::run()` must be made, otherwise execution will stop.

The example below illustrates a rather naïve filter which prevents an action from being executed unless the current day is a Monday.



```php
<?php

class DayController extends CController
{

    public function filters()
    {
        return array(
            // apply the "checkWeekday" filter to the "display" action
            'checkWeekday + actionDisplay',
        );
    }

    public function filterCheckWeekday($filterChain)
    {
        if (date("N") === 1)
            $filterChain->run();
    }

    public function actionDisplay()
    {
        // only executed on Mondays
    }

}
```

*Figure 4 A method filter*

### *Class filters*

*Class filters* are more powerful than method filters since they can act as both pre and/or post-filters. In addition to that they can also take parameters when configured, which allows them to be more generic.

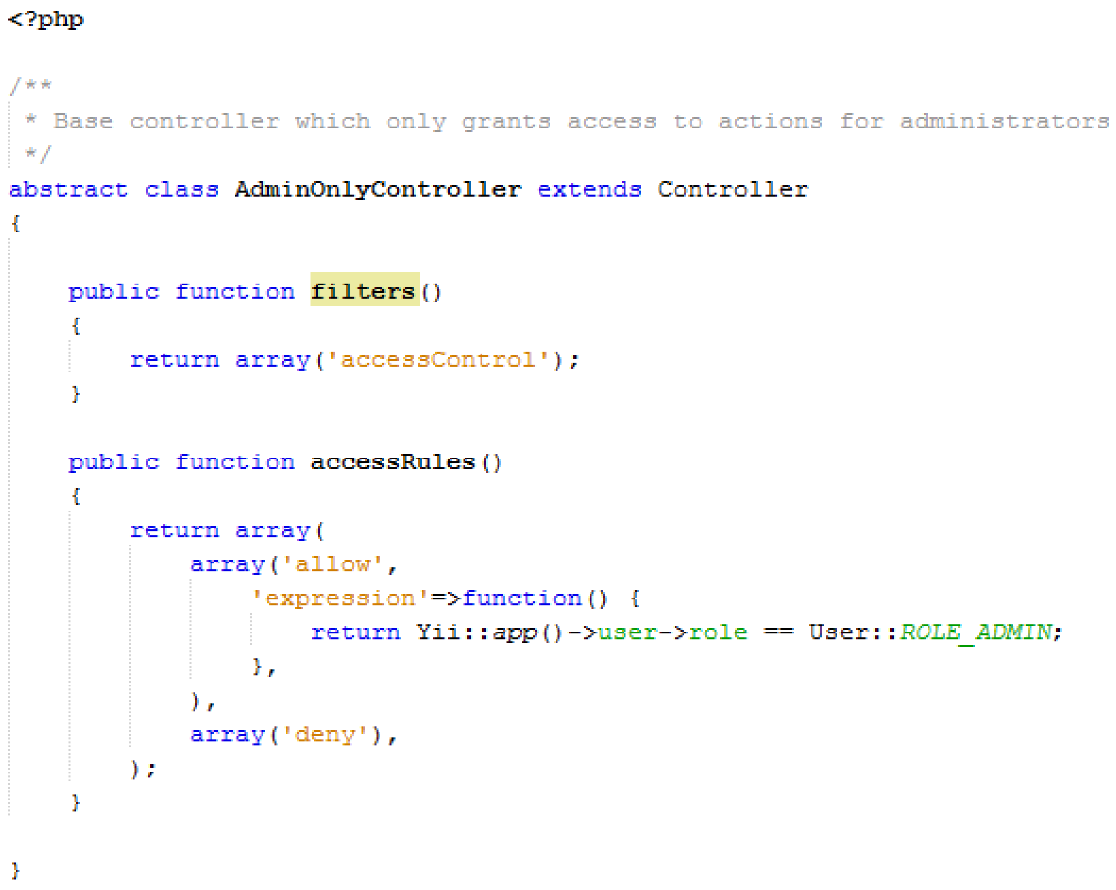
A class filter is any class that extends `CFilter` and whose name ends with "Filter". `CFilter` provides two protected methods that should be overridden to implement the desired functionality; `CFilter::preFilter()` and `CFilter::postFilter()`.

### *Access control*

One of the most common filters is an access control mechanism. An access control filter can place limits on which and by whom actions can be performed. For example, a user that is not logged in (generally referred to as a "guest" in Yii) may only be able to view a blog post while an authenticated user can also update a post, if it was written by that user. Furthermore, an administrator can update and delete any post.

Since access control is such a common requirement in web applications, Yii ships with an integrated class filter called `CAccessControlFilter` which handles this. The access control rules are defined by overriding the `CController::accessRules()` method and returning an appropriate array. The contents of the array determine how the access control is carried out by the filter. Each element in the array is another array whose first value is either “allow” or “deny”, optionally followed by other key value pairs that narrow down the scope of the access rule.

Figure 5 shows a base class for controllers which ensures all actions are executable only by administrators (i.e. users who have the `User::ROLE_ADMIN` role). The return value of the anonymous function determines whether the “allow” rule should be used, otherwise the filter continues to the next rule which flatly denies the request.



```php
<?php

/**
 * Base controller which only grants access to actions for administrators
 */
abstract class AdminOnlyController extends Controller
{

    public function filters()
    {
        return array('accessControl');
    }

    public function accessRules()
    {
        return array(
            array('allow',
                'expression'=>function() {
                    return Yii::app()->user->role == User::ROLE_ADMIN;
                },
            ),
            array('deny'),
        );
    }

}
```

*Figure 5 An example of how the access control filter works*

The example in Figure 5 illustrates an abstract base controller which restricts access to any action for users who are not administrators (the logic that determines whether a user has a specific role is not related to the access control filter itself, it is something the developer has to implement separately).

### 2.5.2 Layouts and views

Every web application needs to render something to the browser, usually HTML. When creating very simple sites with only a few distinct pages it may be tempting to just copy the layout to each file that is served to the browser and vary the actual content inside. A smarter approach is to use a master layout view to represent the similar or static parts of the web page (such as the `<head>` and `<body>` tag, excluding the actual page contents) so that they can be reused and easily modified. Yii uses this concept and provides an easy way to use the layout to render individual pages.

In Yii, all view files, including the layouts are placed in the *protected/views* directory. This directory contains subdirectories for each controller and a special *layouts* directory where the layout views are placed.

A *view* is usually rendered from a controller action using the method `CController::render()`. This method takes two parameters; the first is the name of the view to render, the second is an optional array of parameters to pass to the view. Each parameter key will be extracted into a variable that can be used inside the view file. The controller itself determines which layout file should be used. The layout defaults to “main” but can be changed by changing the value of `CController::$layout`.

A view by itself is not very useful, so when `render()` is called it actually places the contents of the rendered view inside a variable named `$content`, which is `echo`ed in the layout file. The effect of this is that the layout and the view are merged into a complete web page, which is sent to the browser when the request is completed.

#### *Partial views*

Sometimes a particular view is generic and thus supposed to be reused by different actions. A common example of this is a form. The same form could be rendered on both the “create” and “update” pages. Instead of duplicating the necessary code, a *partial view* can be used.

A partial view in Yii is very similar to a standard view, except it’s render with `CController::renderPartial()` instead of `render()`. The difference between these methods is that `render()` wraps the layout around the rendered view while `renderPartial()` just renders the output directly, without any surrounding layout. This also means that `renderPartial()` should usually be called from inside a view, not from a controller. A common exception to this rule is an action that is supposed to be called via AJAX to fetch some piece of HTML. In that case the controller should render just the view and nothing more since the whole page, including its layout, has already been sent to the browser.

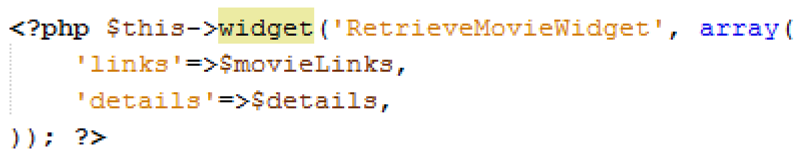
### 2.5.3 Widgets

Yii has a concept of *widgets*, which despite the name is actually just a view in class form. Widgets are very useful when you need to implement more complex views, especially those that display different things depending on various parameters, such as a calendar.

A widget is a class that extends the base class `CWidget`. When an instance of `CWidget` is created, two methods are run; `CWidget::init()` and `CWidget::run()`. The `run()` method is usually the one responsible for doing the actual rendering.

There are two ways of using widgets. Despite being a regular class, widgets aren't meant to be instantiated directly. Instead, the base controller class provides methods for rendering widgets, namely `CController::widget()`, `CController::beginWidget()` and `CController::endWidget()`.

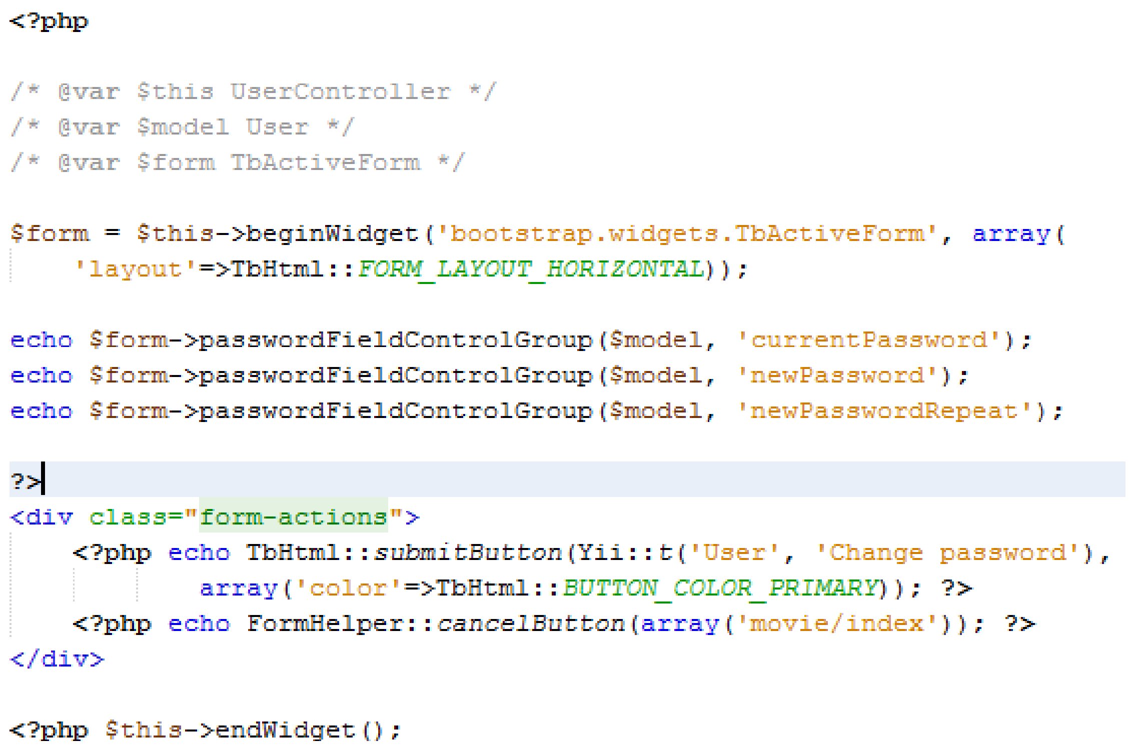
The first method is the easiest and most common way to create a widget. When `widget()` is called, the widget's `init()` and `run()` methods are called immediately. Optional parameters can be passed to `widget()` to configure the widget class, as seen in Figure 6.



```php
<?php $this->widget('RetrieveMovieWidget', array(
    'links'=>$movieLinks,
    'details'=>$details,
)); ?>
```

*Figure 6 Rendering a widget using CController::widget()*

The second way is to use the begin/end methods. In this scenario, `beginWidget()` calls the `init()` method and `endWidget()` calls the `run()` method. This is how so called *active forms* in Yii work (an active form is a form that is tied to a particular model). When using an active form, `beginWidget()` renders the opening `<form>` tag, the caller then renders the form elements before `endWidget()` is called which renders the closing `</form>` tag. An example is shown in Figure 7.



```php
<?php

/* @var $this UserController */
/* @var $model User */
/* @var $form TbActiveForm */

$form = $this->beginWidget('bootstrap.widgets.TbActiveForm', array(
    'layout'=>TbHtml::FORM_LAYOUT_HORIZONTAL));

echo $form->passwordFieldControlGroup($model, 'currentPassword');
echo $form->passwordFieldControlGroup($model, 'newPassword');
echo $form->passwordFieldControlGroup($model, 'newPasswordRepeat');

?>
<div class="form-actions">
    <?php echo TbHtml::submitButton(Yii::t('User', 'Change password'),
            array('color'=>TbHtml::BUTTON_COLOR_PRIMARY)); ?>
    <?php echo FormHelper::cancelButton(array('movie/index')); ?>
</div>

<?php $this->endWidget();
```

*Figure 7 Rendering a widget using beginWidget() and endWidget()*

## 2.6 Caching

A web application is by default stateless since the underlying HTTP protocol has no state. Generally this means that when a page is displayed, all the calculations required to render it have to be done again. Data needs to be fetched from the database and processed; templates need to be compiled and so on. For a small application this may not pose a problem, but the more processing is required in order to display a page, the bigger the chance that the page will load slowly. Various levels of caching are one commonly used solution to this problem.

Yii provides a robust framework for caching content on a web page. There are different caching mechanisms for caching data, depending on the specific scenario:

- *Data caching*. This implies caching the value of a single variable, such as the results of an expensive database query or a complex calculation.
- *Fragment caching*. A fragment is a section of a page, like a list of new products on the front page of a web store.
- *Page caching*. A page of course refers to a complete web page.

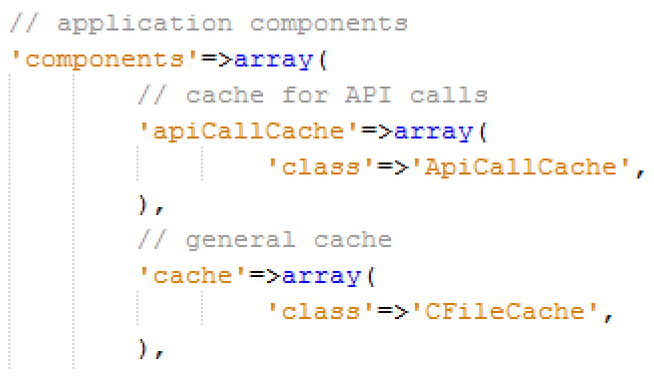
Caching in Yii is accomplished by using a cache component. To control the underlying caching mechanism, the component can be specified in the configuration file. Some of the most commonly used caching components include:

- `CFileCache`. This is the default cache component used. It caches data in a flat files located in *protected/runtime/cache.*
- `CApcCache`. This component utilizes the PHP *APC* extension to store cached data. APC is normally used for *opcode caching* but it can also cache so called "user data". The data is stored in memory and is usually emptied implicitly when the web server is restarted.
- `CMemCache`. Utilizes a memcached server for storing cached data.

Caching can also be used more transparently by various parts of the framework. Two common examples are message caching (a message in Yii is a string translation from one language to another) and database query caching. Regardless of how something is cached, be it explicitly or implicitly, the default cache component is used (unless explicitly changed).

By default, Yii provides a default cache component in `Yii::app()->cache`. The default underlying cache implementation is a `CFileCache`. Quite often it is enough to use a single cache component and store all data there, but in larger applications it may make sense to use different components depending on the size and type of the data to be cached. For example, while a `CFileCache` is very fast compared to performing an SQL query, a `CApcCache` is even faster since the cached data is stored in memory and thus doesn't have to be read from disc. An in-memory cache has other weaknesses though, such as limited storage space, so it can be beneficial to selectively define which cache component is used for what data.

Figure 8 shows how different cache components can be configured. The components are accessible using `Yii::app()->apcCache` and `Yii::app()->cache` respectively.



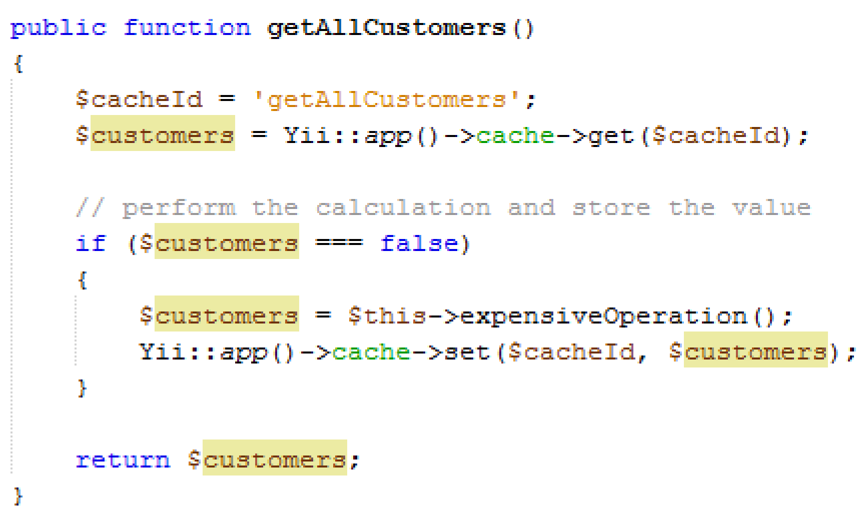
```
// application components
'components'=>array(
        // cache for API calls
        'apiCallCache'=>array(
                'class'=>'ApiCallCache',
        ),
        // general cache
        'cache'=>array(
                'class'=>'CFileCache',
        ),
```

*Figure 8 Configuring cache components*

### 2.6.1 Data caching

Data caching is very useful for hiding a caching mechanism in the lower levels of an application. For example, the results of a method named `getAllCustomers()` may be used to create a data provider which in turn will be used in various views across an application. By encapsulating the caching in `getAllCustomers()`, the upper layers of the code do not have to concern themselves with the caching at all, in fact, they need not even be aware that any caching is taking place.

Data caching is accomplished by using the `CCache::get()` and `CCache::set()` methods. The getter will return the value specified by a "cache ID", or `false` if no value exists in the cache. In that case, the data should be stored in the cache using `set()`. This process is illustrated below in Figure 9.



```php
public function getAllCustomers()
{
    $cacheId = 'getAllCustomers';
    $customers = Yii::app()->cache->get($cacheId);

    // perform the calculation and store the value
    if ($customers === false)
    {
        $customers = $this->expensiveOperation();
        Yii::app()->cache->set($cacheId, $customers);
    }

    return $customers;
}
```

*Figure 9 How to use data caching*

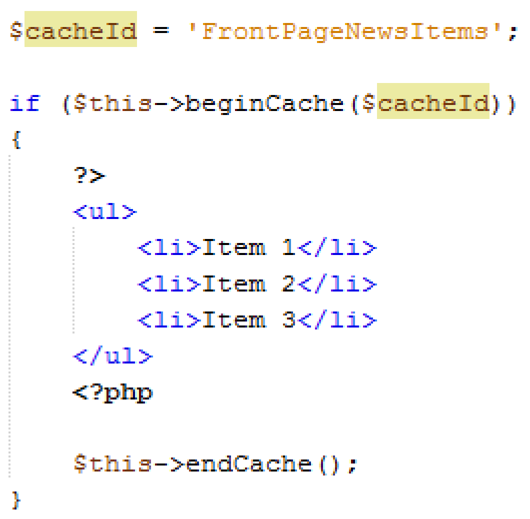
Data caching is not the solution to all performance problems. If a lot of small pieces of data are stored in a `CFileCache` it may actually be slower to fetch the values from the cache than to recalculate them. In cases like that, a faster cache component should be used or caching should be eliminated completely.

### 2.6.2 Fragment caching

Fragment caching involves caching a portion of the final HTML (the *fragment*) that is rendered on a page. This is useful when data caching is not an option, such as when the operations required to render the content are so many that implementing caching for all of them would be infeasible.

A common use case for fragment caching is the main menu of a website. On a dynamic website, the process of building the actual menu structure can be fairly complex. Database queries, user permission checks and the current language are just a few examples of things that may be going on in the background while the menu is rendered.

Fragment caching in Yii is accomplished by the `CBaseController::beginCache()` and `CBaseController::endCache()` methods. Any output between those calls will be stored in the cache and then rendered, except if `beginCache()` returns false, which means the content is already cached and can be served immediately from the cache. Fragment caching is illustrated in Figure 10.



```
$cacheId = 'FrontPageNewsItems';

if ($this->beginCache($cacheId))
{
    ?>
    <ul>
        <li>Item 1</li>
        <li>Item 2</li>
        <li>Item 3</li>
    </ul>
    <?php

    $this->endCache();
}
```

*Figure 10 Fragment caching in action*

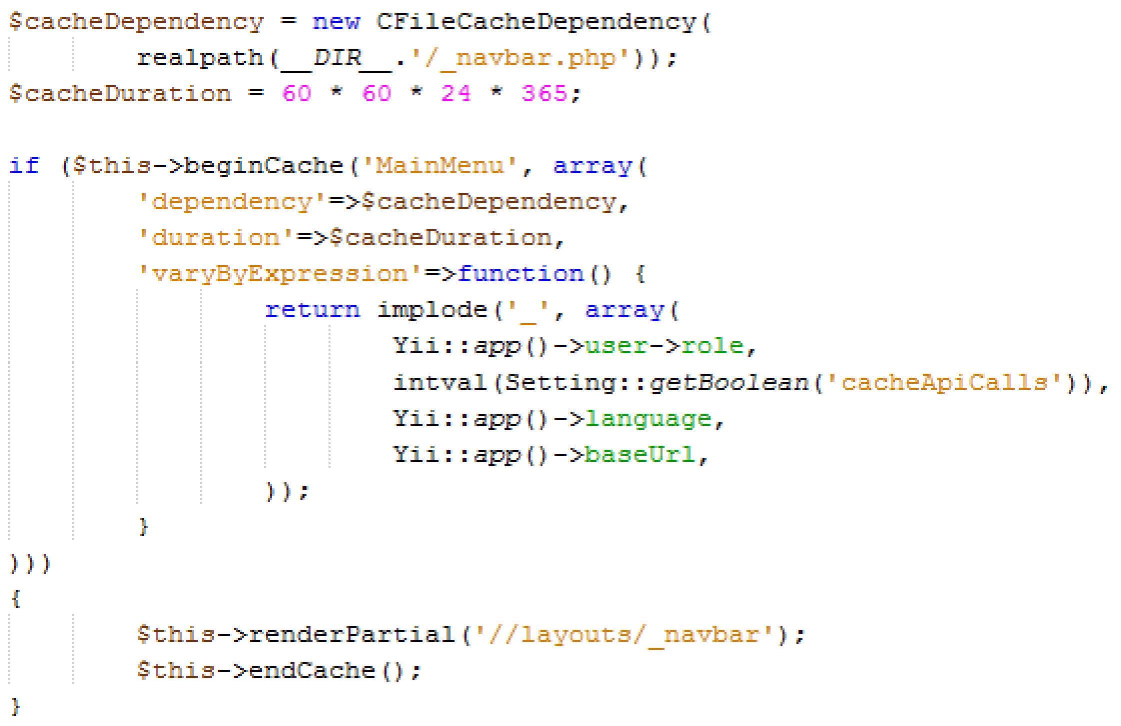
### 2.6.3 Cache invalidation

A recurring problem when caching dynamic data is that the cached data may be outdated compared to the actual data. In some scenarios this might be okay, but imagine a main menu on a page that remains the same after a user changes the site language. A properly designed caching mechanism should take into account all the factors that can invalidate the cached data and automatically react to any changes in these factors. This way the caching can be completely transparent to the user, albeit at the cost of more frequent cache invalidations. The net effect however, is usually a better performing application.

Yii provides three main ways to indicate how and when a cache should be invalidated:

- *Time duration*. The cache will automatically invalidate itself after a certain amount of time has passed. This can be useful for simple caching scenarios where it is not critical that the information displayed is totally up to date.
- *Cache dependency*. A cache dependency is an object that encapsulates the logic that determines whether a cache should be invalidated or not.
- “*Vary by expression*”. This means that a separate cached copy is stored for every variation of a certain value, usually the return value of a closure or a function.

To better illustrate how all these methods can be used to create complex cache invalidation logic, let’s look at an example (Figure 11):



```php
$cacheDependency = new CFileCacheDependency(
        realpath(__DIR__.'/_navbar.php'));
$cacheDuration = 60 * 60 * 24 * 365;

if ($this->beginCache('MainMenu', array(
        'dependency'=>$cacheDependency,
        'duration'=>$cacheDuration,
        'varyByExpression'=>function() {
                return implode('_', array(
                        Yii::app()->user->role,
                        intval(Setting::getBoolean('cacheApiCalls')),
                        Yii::app()->language,
                        Yii::app()->baseUrl,
                ));
        }
)))
{
        $this->renderPartial('//layouts/_navbar');
        $this->endCache();
}
```

*Figure 11 How cache invalidation can be accomplished*

The snippet above defines the cache invalidation logic for the main menu on a page. The menu is rendered by `renderPartial()`, and the objective is to avoid performing that operation by all means. All three methods explained previously are used in this example:

- the cache is valid for a year (the duration parameter) unless the dependency doesn’t change
- a `CFileCacheDependency` is used to invalidate the cache whenever the file that contains the menu contents changes
- last but not least, the cached copy is varied by an expression, in this case the value of a string comprised of various other expressions, such as the current language, the current base URL (so that if the application is moved to another physical file location the links will be updated) and the user’s role (administrators have more items in the menu than normal users).

## 2.7 Modules

*Modules* in Yii are practically applications within an application. Modules have their own directory structure that mimics the base path of the main application (the *protected/* directory), including its own controllers etc. Requests that are handled by a module have the module's name prefixed to the route, e.g. *admin/user/create*, which corresponds to `UserController::actionCreate()` in the `Admin` module.

## 2.8 Extensions

Extensions are any piece of code, be it an application component, a behavior, or anything else, that is not a part of the Yii framework. Extensions are generally placed in *protected/extensions*, and Yii includes a default “ext” path alias for that directory to ease importing of extensions.

The Yii framework web site has a collection of community-developed extensions that can be used, although nowadays more and more extensions are available through Composer instead (see the next chapter).

# 3. COMPOSER

Whenever an application is developed, the developer usually strives to minimize the amount of code that has to be written, especially if it's been written before by someone else. In cases like that it makes sense to include third-party code in the project instead of "reinventing the wheel". This is a situation where a *dependency manager* enters the picture. Without a dependency manager, if an application requires e.g. framework "X" and libraries "Y" and "Z", the alternatives for the developer would be to:

- Include the source code for the dependencies in the project's source tree. This bloats the size of the code that each developer has to check out and makes handling updates to the dependencies a bit cumbersome
- Include the frameworks as sub-repositories in the project source control repository. This is better than straight off copying the dependencies into the source tree itself, but it still makes version management a bit complicated.
- Leave the dependency management to the end-user completely. This is seldom preferred since the deployment instructions have to be very clear as to where to put the files in question, which versions to use, and so on.

A dependency manager solves this problem in a manageable way. The developer only has to compile a list of the third-party projects required along with their respective versions. The dependency manager will then take care of acquiring those dependencies and installing them in a common location.

## 3.1 Introduction

Composer is a tool for dependency management in PHP [19]. Composer itself is written in PHP and is shipped as a single PHP Archive file (PHAR).

Composer only started to take off after the introduction of PHP 5.3, which introduced namespaces [14]. The ability to declare classes under namespaces paved the way for a set of rules for naming classes, and thus autoloading them as well. The first of these standards is today known as PSR-0 [20].

When you want to use a class in PHP, you have to include the file in which the class is declared, either by calling `require` or `include`. Naturally, this becomes tedious for any but the smallest web applications. By using a standard way of naming and namespacing classes, tools like Composer can figure out where to find the corresponding files once installed. For example, if you attempt to use a class named `touki\FTP\Connection`, Composer's autoloader will know (thanks to PSR) that it should look for a file named `Connection.php` in the directory *touki/FTP*.

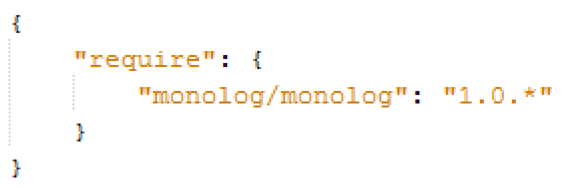
In order to start using Composer, one must download and install it. The easiest way to do this is to run the command `curl -sS https://getcomposer.org/installer | php` in the project's root directory. This downloads a file named *composer.phar*, which is the actual Composer executable. To install Composer globally on the computer (which is often preferred if the computer hosts many different projects) the file can be renamed to just `composer` and moved to a location within the user's path (e.g. */usr/local/bin* on Unix-based systems). This way the user can type `composer` instead of `php composer.phar` regardless of the current directory.

## 3.2 The *composer.json* file

The next step is to create a *composer.json* file in the project's root directory. This file lists the project dependencies, including which versions of the individual libraries should be used. Once the project dependencies have been defined they can be installed by running `php composer.phar install`. The install command downloads all the dependencies into a directory named *vendor*.

After installing all dependencies, Composer creates an auto-generated class loader in *vendor/autoload.php*. This file knows which namespaces exist in which folders, which means it's the only file that a developer has to include in his project in order to use the installed dependencies. As additional dependencies are added and installed, the class loader file is updated accordingly to make the new dependencies available.

The composer.json file is like its name suggests comprised of JSON. The content is a single JSON object which in its simplest form can look like Figure 12.



```
{
    "require": {
        "monolog/monolog": "1.0.*"
    }
}
```

*Figure 12 A composer.json file in its simplest form*

As illustrated, the file contains a single JSON object. Inside it the `require` property is an object that lists the dependencies one by one. Each object's attribute is the name of the package and the value is the desired version. In the example above, the "monolog/monolog" project is listed as a dependency. The defined version string means that any version beginning with 1.0 is desired.

Projects required through Composer are advised to use semantic versioning [21]; otherwise the desired version required cannot be correctly determined. Semantic versioning means that the project version number should be in the form *major.minor.patch*, e.g. 2.5.1. The major version is bumped whenever an incompatible API change is made, the minor version is bumped when new functionality is added without breaking existing functionality, and finally the patch version is bumped whenever a simple bug fix is made.

### 3.2.1 The *composer.lock* file

Once the defined dependencies have been installed, a lock file named *composer.lock* is created. The lock file contains information about the dependencies installed, such as which exact version was installed. This file should be included along with *composer.json* in the project's source control [22].

The lock file is an important tool in assuring that anyone who installs the project dependencies gets the exact same versions as the original developer intended. Going back to the example with "monolog/monolog" from the previous chapter, let's imagine that the developer gets version 1.0.2 and develops against this specific version. Later when a different developer is about to install the dependency, the newest version of "monolog/monolog" may be 1.0.4. Without the lock file, when the developer installs the dependencies Composer would download version 1.0.4, which may or may not work with the existing code. If a lock file would be present, Composer would have installed

}

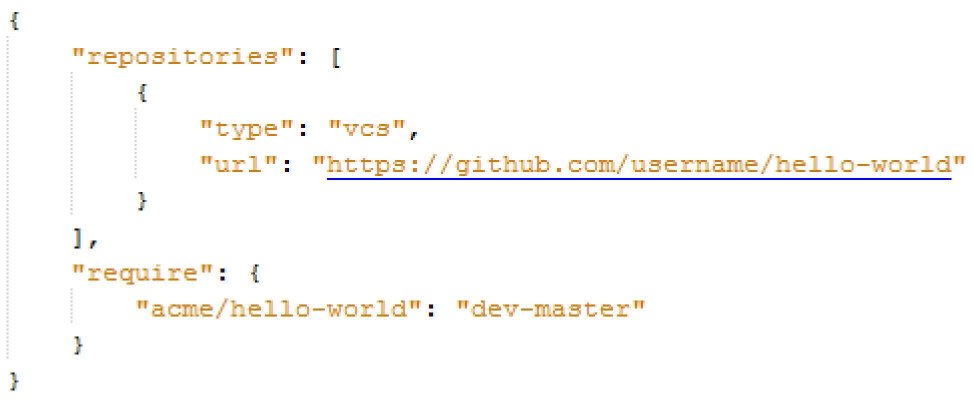
version 1.0.2 instead, despite a newer version being available. This is why it is recommended to include the lock file in the project's source control [22].

Going further, if a developer decides that a dependency should be upgraded to the latest version he can run `composer update <dependency>` to update a single dependency, or simply `composer update` to pull the latest version of all configured dependencies. This command will automatically update the lock file so that the next time someone runs `composer install`, the newest version will be downloaded.

## 3.3 Packagist

In the example with "monolog/monolog" in the previous chapters it is clear that the *composer.json* file doesn't specify where to acquire the dependencies from. What Composer actually does is look up the dependencies one by one through a service called Packagist. Packagist is the main repository for Composer dependencies [23]. Through Packagist, anyone can submit Composer-compatible projects which then become available to anyone using Composer. By linking an existing Github account to Packagist the site can be automatically updated whenever a developer pushes commits to a published project, including new tags.

Sometimes you'll find that a particular project is not available on Packagist. This doesn't pose a problem since Composer supports specifying package sources manually. This way, private projects (e.g. private repositories on Bitbucket or git projects hosted on an internal company server) can be used as dependencies too.



```
{
    "repositories": [
        {
            "type": "vcs",
            "url": "https://github.com/username/hello-world"
        }
    ],
    "require": {
        "acme/hello-world": "dev-master"
    }
}
```

*Figure 13 A composer.json file which references a private repository*

In Figure 13, the “acme/hello-world” project is apparently not available on Packagist, so a `repositories` property has been defined. This property is an array of objects, each defining a particular repository. When running `composer install`, Composer will notice that “acme/hello-world” is not published on Packagist and check the user-defined repositories instead.

## 3.4 Integrating Composer with Yii

As previously mentioned, Composer comes with its own autoloader which is updated whenever new dependencies are installed or updated. This loader needs to be included somewhere in order for PHP to find dependencies directly using their namespaced name. This can be achieved by adding the required `include` statement to the Yii bootstrap script. Once the autoloader is included¸ Composer dependencies can be used normally from anywhere inside the Yii application.

### 3.4.1 Using Composer dependencies as application components

While including the Composer autoloader is generally enough in order to use the installed dependencies, there is a caveat when they are to be used as application components. Like mentioned in the early chapters, an application component is defined in the main Yii configuration file. The name of the class is specified as a string here. To find the class, Yii attempts to load the class directly using any registered autoloaders, unless the string is a class path, in which case it attempts to include the file based on that path. While this shouldn’t normally pose a problem, there’s an outstanding bug in Yii’s loading mechanism that prevents e.g. `'class'=>\monolog\Monolog`, from working [24].

Luckily, there is a decent workaround for this, namely to specify the Composer-generated *vendor/* directory as a path alias and use a path when specifying the class name. The path alias is defined by calling `Yii::setPathOfAlias('composer', '/path/to/vendor');` in the bootstrap script. After that the “monolog/monolog” class can be defined as `'class'=>'composer.monolog.Monolog'`.

# 4. XBMC VIDEO SERVER

This section describes the XBMC Video Server application that was developed as a part of this thesis.

## 4.1 Background

Today's home Internet connections are faster than ever before which have given rise to new possibilities regarding what can be done with them. Traditionally people come into contact with video streaming through third-party services such as YouTube or Netflix. To use these types of services reliably, all you really need is a decent amount of downstream bandwidth [25].

It is due to the increase in upstream bandwidth over the last few years that a previously unthinkable scenario has become reality, namely to stream video files from your home network over the Internet, bypassing the need for any third-party service.

Video streaming isn't very different from audio streaming, except that it requires a lot more bandwidth. However, one important caveat is the software available to ease the experience. It has always been possible to set up a simple HTTP server that serves directories, although it is not very convenient. There is the issue about access control (it is generally a bad idea to expose a large part of your local file system to the public Internet), and traditionally the user interface is not easily customizable, although recent developments have improved this area a lot [26]. It may work satisfactorily if all you want is to get access to your files, but people want a more complete experience than that. If you're going to be browsing and streaming media, you want as much accompanying details about the media as possible, such as cover art, playback duration, et cetera.

For music streaming this problem has been practically solved a long time ago. There are many very capable open source projects aimed at providing a good interface for exposing your private music collection to the Internet, complete with audio file tag parsing to get at the metadata. Examples of such applications include Ampache (the name itself is a combination of Winamp, a classic music player for PCs, and Apache, the most popular web server in use today) and Subsonic (kind of a personal Spotify application). [27]

None of the existing tools and projects is particularly well suited to streaming movies and TV shows. First, video files usually don't contain any metadata like music files often do [28], instead the only hint as to the contents of the file is the filename itself. This means that in order to provide a rich user experience, some online service would have to be consulted to retrieve metadata for a particular file. That in itself is not a huge problem, but once you have a way of gathering metadata you need to store it somewhere and preferably not directly in the file system.

Another concern is how to deal with multi-part files, e.g. a movie that is split over two files as well as files that are split into RAR archives. The latter is common with media downloaded from the Internet [29]. While some media players can play files contained in RAR archives natively, they still require direct access to all the subsequent archive parts, something that is not easily accomplishable when files are served over the web [30].

To counteract these problems I decided to base my application on a widely used piece of software called XBMC.

### 4.1.1 XBMC

*XBMC* [31], nowadays known as Kodi [32], is an award-winning free and open source software media player and entertainment hub [33]. Originally developed as a media player for hacked Xbox consoles back in 2002, it has grown to become one of the largest and most widely known media center solutions in use today. Since the original Xbox gradually became more and more outdated, the project was ported to the Linux and Windows operating systems. In recent years it has seen continued growth and is now available on Android as well as other ARM platforms, Apple iOS, Mac OS X, and FreeBSD (in addition to Linux and Windows).

XBMC maintains what it calls a *library* over the media files in your computer. You configure a set of *sources* (which can be e.g. local directories or remote networks shares), specify what type of data the source contains (be it movies, TV shows, music et cetera), and what *scraper* should be used to gather metadata about the contents (IMDb for example). XBMC then scans the sources and builds a database containing all the information that has been gathered. This solves one of the previously mentioned problems, namely how to categorize and extract metadata from various media files.

XBMC contains an integrated web server which can be used to control the application as well as to access files in its virtual file system. The virtual file system is comprised of the media sources one has configured as well as special paths for extra data, such as artwork and other images. This way a third-party application can access the media files that XBMC has scanned into its library, a crucial feature for XBMC Video Server. The virtual file system is exposed through `/vfs` on the web server [34].

Clients wishing to communicate with XBMC may do so through a *JSON-RPC* API. JSON-RPC is a relatively simple RPC protocol based on sending JSON objects back and forth. The JSON-RPC protocol is transport agnostic, which means that it is up to the developer to define how exactly the RPC is implemented. XBMC supports five different transports: [35]

- A Python transport, used only by XBMC add-ons
- HTTP POST requests (the de facto standard for JSON-RPC servers)
- HTTP GET requests where the request body is sent in a query parameter named `request`
- Raw TCP sockets. In this mode a client sends raw JSON-RPC requests over a TCP socket to port 9090 on the machine running XBMC. This feature has to be enabled explicitly in XBMC in order to work.
- Using WebSockets

XBMC Video Server relies solely on the HTTP POST API since it is very easy to integrate in a language like PHP that is designed for use on the web.

## 4.2 Functionality overview

XBMC Video Server allows users to expose their XBMC library contents to the web through an easy to use interface. The user can browse the library and view details about specific movies, TV shows or episodes. The browse pages can be filtered according to various parameters, such as genre, year, quality, and rating.

The details page for an item contains basic information about the item in question, such as artwork, runtime, plot, and so on, as well as a button for watching the media.

The application itself requires users to authenticate with a user name and password, which means that there is a concept of user roles. There are three distinct roles; administrators, users and spectators. Administrators can create new users and change the global application settings, such as language. Normal users can only consume content on the site (and change their own password) and finally, a spectator can only browse content, not download or stream anything. The spectator role isn't very useful, it was added mainly because someone requested it [36].

The application settings allow certain aspects of the application's functionality to be changed. Some of the more important ones include the ability to cache API call results (in order to speed up the application, especially on slower platforms) and check users against a white-list before allowing them to log in.

XBMC Video Server supports adding multiple *backends*. A backend in this context is an instance of XBMC that the application connects to and displays information from. Certain operations can be performed on the backend, such as triggering a library update.

## 4.3 Design

This chapter will explain how the application is designed and how to various parts work together.

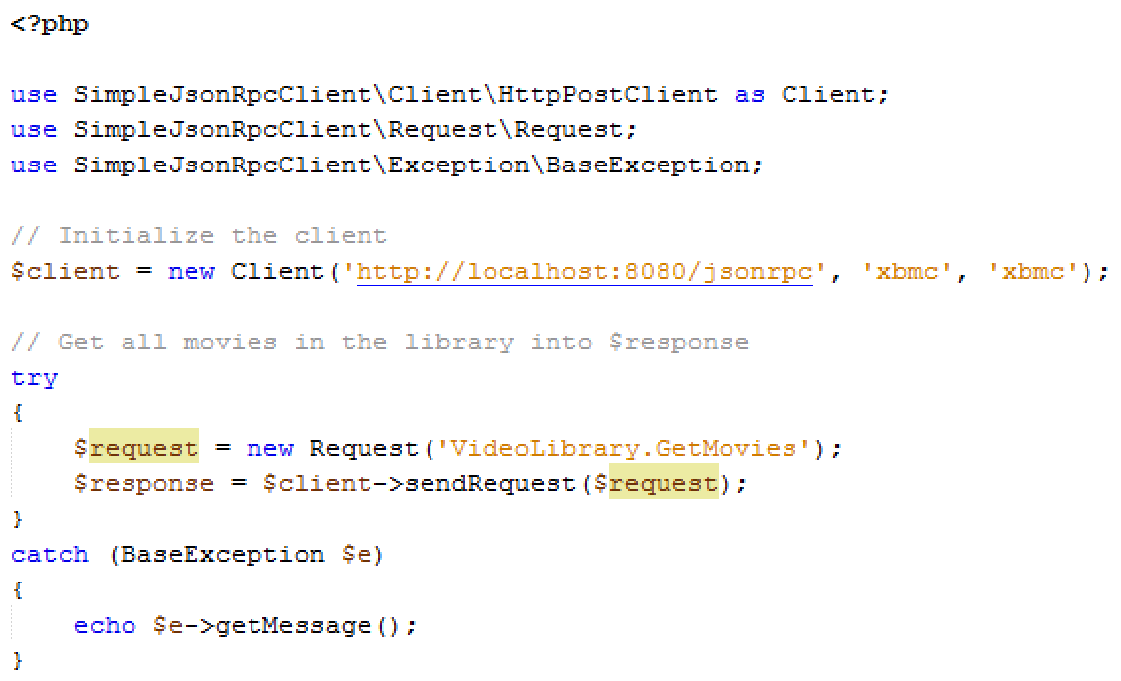
### 4.3.1 The API layer

As mentioned earlier, all communication with XBMC goes through its JSON-RPC API. When I started working on this application I noticed that there weren't any suitable JSON-RPC clients for PHP available so I decided to write my own. The result was an independent library called *simple-json-rpc-client*. The library is also published on Packagist so that it can be used as a dependency to any project that uses Composer.

The JSON-RPC client is pretty simple (hence its name). It consists of request and response objects and a client. Since the JSON-RPC specification doesn't specify any particular transport for the API calls, an interface is used which all clients must implement. The library includes a default client which uses HTTP POST requests for communication. The HTTP client is taken from the *Zend* framework.

To use the client the user creates a `Request` object and calls a method on the client object which sends the request to the server and returns a `Response` object. The `Response` object is a PHP representation of the raw JSON response.

Figure 14 shows a real example of how simple-json-rpc-client can be used.

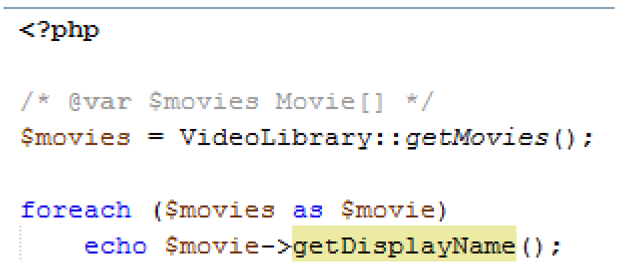


```php
<?php

use SimpleJsonRpcClient\Client\HttpPostClient as Client;
use SimpleJsonRpcClient\Request\Request;
use SimpleJsonRpcClient\Exception\BaseException;

// Initialize the client
$client = new Client('http://localhost:8080/jsonrpc', 'xbmc', 'xbmc');

// Get all movies in the library into $response
try
{
    $request = new Request('VideoLibrary.GetMovies');
    $response = $client->sendRequest($request);
}
catch (BaseException $e)
{
    echo $e->getMessage();
}
```

*Figure 14 Example usage of the simple-json-rpc-client library*

In XBMC Video Server, the interface for the JSON-RPC client is contained within an application component called simply `XBMC`. Like all application components in Yii it can be accessed through `Yii::app()->xbmc`. The component provides wrappers for sending requests and receiving eventual responses. It is at this layer in the application that the API call cache is implemented. Since most of the data returned from the API (such as the list of available movies) doesn't change very often, and processing the data takes a while (especially with large result sets), it makes sense to cache it. By containing the functionality here the rest of the application doesn't need to be aware of any caching mechanism.

The actual calls to the API are mostly performed by a static helper class called `VideoLibrary`. It provides convenient methods for fetching specific data from XBMC and provides an abstraction in case the underlying API changes sometime in the future (the XBMC API has gone through a few major revisions since its inception [37]). The illustration in Figure 15 shows an example of how it can be used.

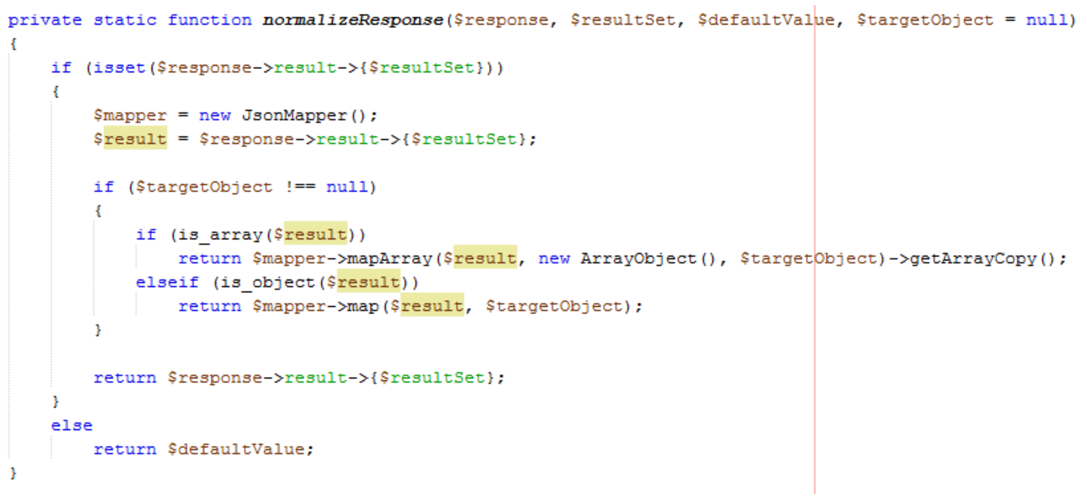


```php
<?php

/* @var $movies Movie[] */
$movies = VideoLibrary::getMovies();

foreach ($movies as $movie)
    echo $movie->getDisplayName();
```

*Figure 15 Example of how to use the VideoLibrary static helper*

The JSON-RPC client returns API responses as raw PHP objects, i.e. instances of `stdClass`. This means that the objects don't have any methods and their properties are not known, making automatic code completion in editors impossible. To solve this problem I've used a mapper library called *jsonmapper*.

Jsonmapper's purpose is to map JSON objects to real objects. It does this by parsing the *phpdoc* annotation blocks from a class in order to determine which properties should be mapped where. Using this library, The `VideoLibrary` component can dynamically map the API results to corresponding classes. This enables the application as a whole to work with instances of `Movie`, `TVShow` and `Episode` etc. instead of raw objects. Figure 16 shows how this mapping library is integrated into the `VideoLibrary` class.



```php
private static function normalizeResponse($response, $resultSet, $defaultValue, $targetObject = null)
{
    if (isset($response->result->{$resultSet}))
    {
        $mapper = new JsonMapper();
        $result = $response->result->{$resultSet};

        if ($targetObject !== null)
        {
            if (is_array($result))
                return $mapper->mapArray($result, new ArrayObject(), $targetObject)->getArrayCopy();
            elseif (is_object($result))
                return $mapper->map($result, $targetObject);
        }

        return $response->result->{$resultSet};
    }
    else
        return $defaultValue;
}
```

*Figure 16 Illustration of how VideoLibrary can map raw objects to a desired class*

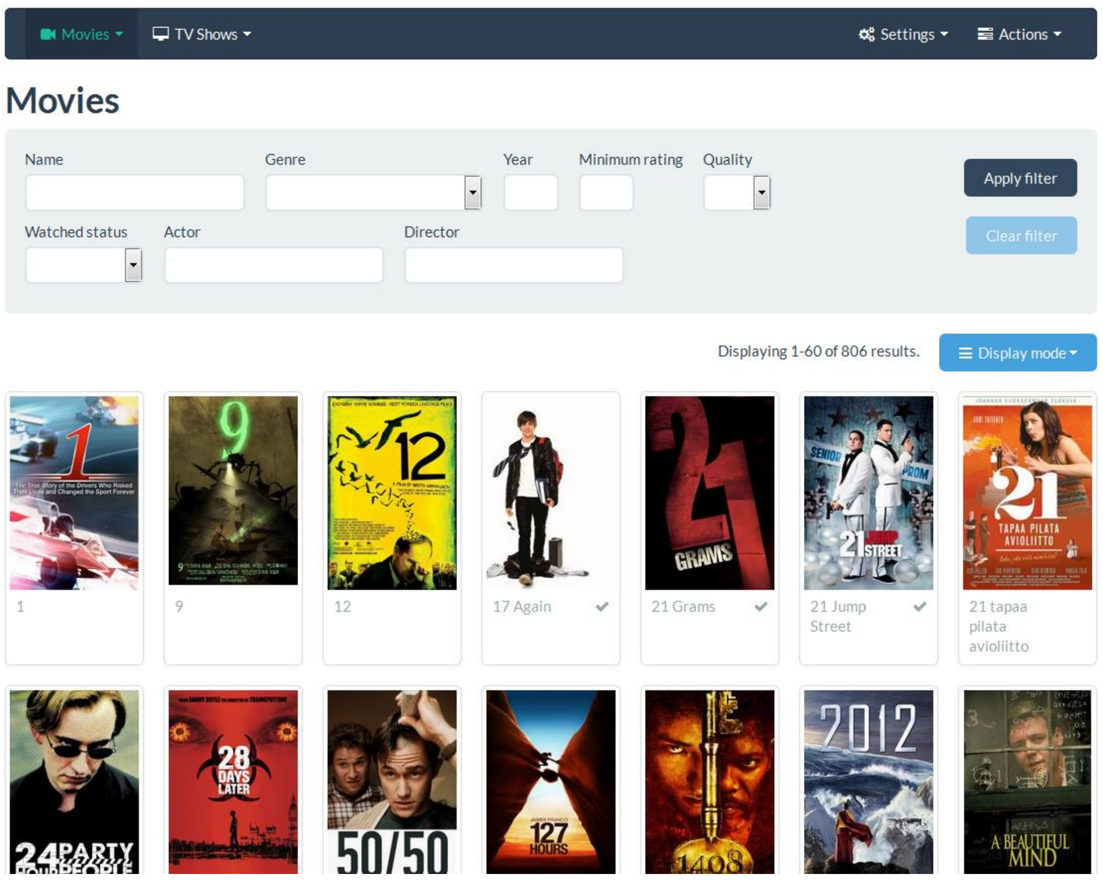
The `normalizeResponse` method can be a bit difficult to understand when taken out of context like this. In essence, it converts raw responses (a `stdClass` instance or an array of such instances) into an object or objects of a more concrete type (specified by `$targetObject`) using the `map` and `mapArray` library methods. The method is also used to provide a default value in case the response doesn’t contain the result expected (hence the “*normalize”* in the method name).

When I started working on XBMC Video Server I didn't use jsonmapper at all, mostly because at the beginning it wasn't that cumbersome to work with the raw objects directly. As the application grew I noticed that a lot of code started to get duplicated so I decided to start mapping the response objects to actual classes. Once again, the application grew and exposed some pretty serious performance issues in the jsonmapper library.

After some investigation I managed to add some runtime caching to the most critical parts of the analysis code. It took me a while but my changes were ultimately merged upstream and are now part of the library as of version 0.4.0 [38].

### 4.3.2 Browse pages

The first page a user sees after a successful login is the “Movies – Browse” page. The browse page contains a list of all movies and a filter form which is used to narrow down the results. The results view can be toggled between two modes; *grid mode* and *list mode*. In grid mode, the artwork and title for each item is shown in a grid. List mode means the results are displayed as a simple list containing title, year, genre, rating and runtime.



Displaying 1-60 of 806 results.

*Figure 17 The browse page in the XBMC Video Server application*

The pages for browsing movies and TV shows are very similar. There are practically only two differences; the available options in the filter form and the columns in the result when viewed in list mode. Because of this the filter form functionality has been factored out into two separate classes, both deriving from a common base class. The same is done for the results since the only thing that differs is the columns that are rendered.

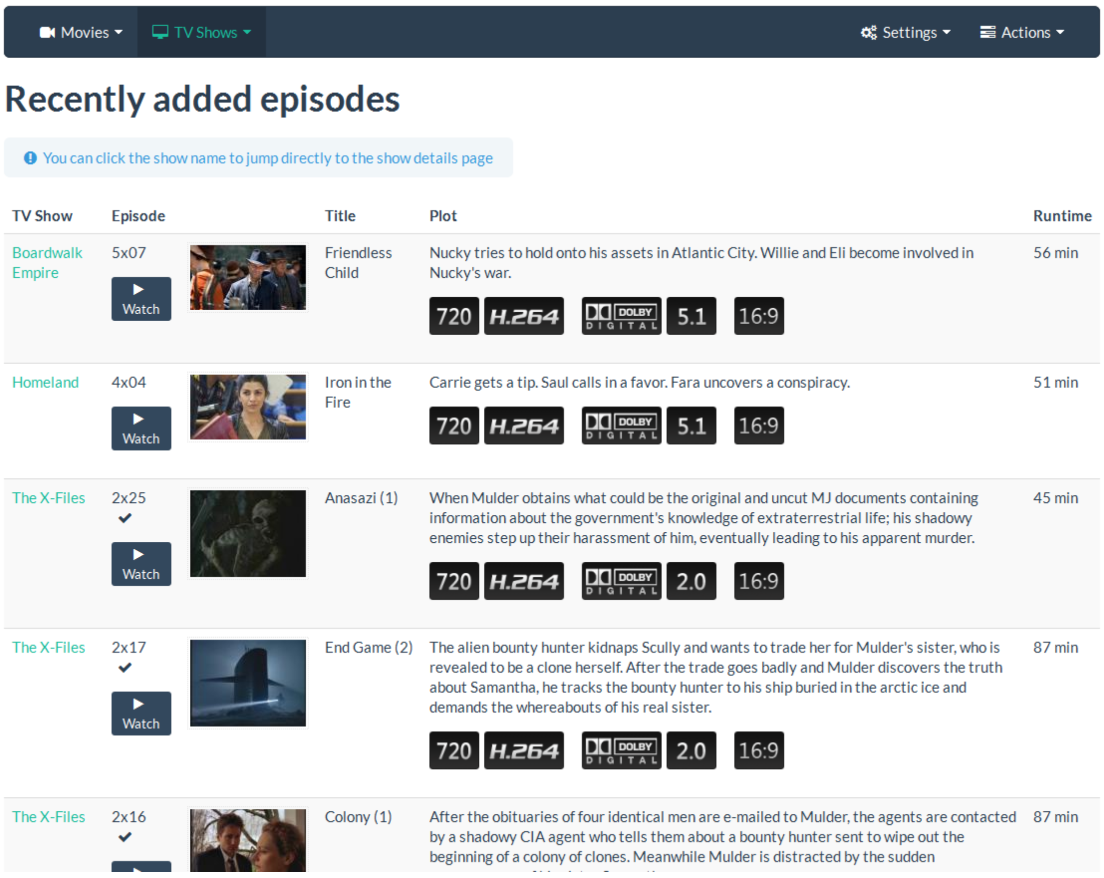
One of the biggest challenges with the browse pages was image handling. Each item in the result is supposed to display the item artwork, which is usually a poster. The images available from the API are quite large, sometimes hundreds of kilobytes in size, which is not suitable for a web page. Additionally an administrator may increase the default page size to more than 60 items which further increases the amount of data the browser has to request.

In order to combat this issue the application uses two techniques. One is to resize images on-the-fly and store them on disk so that they're accessible later. The other trick is to defer loading the actual image in the browser until the image is visible inside the view port, something often referred to as *lazy loading*. The image resizing is done using a third-party package called *php-image-resize* [39].

Lazy loading of images is done using JavaScript. In this particular case a small script called *jQuery Unveil* [40] is used. JQuery Unveil works by adding a placeholder image (often an animated loading icon) as the `src` attribute of the image tag and the actual image URL as a `data-src` attribute. Data attributes is a new feature in HTML5 that allows developers to add arbitrary attributes to tags which can later be accessed through JavaScript [41]. Once the image element is within the *view port* (the area of the web page that is visible on the screen), the value in the `src` attribute is replaced by that of the `data-src` attribute which triggers the browser to load the image from the server. The performance gain achieved by this technique is most noticeable on long pages with lots of images, and it is also commonly used on pages with infinite scrolling.

Despite the fact that the results are paginated (defaulting to 60 results per page), all data is actually fetched from the API and the pagination is done in the application instead. While this sounds inefficient, the results from the API can be cached. This means that no matter what page the user requests the data is already on the server, alleviating the need to request it all again from the API. It is possible to request just a range of items from a list through the API, though as that would result in a different API call for each page it would end up being less efficient than retrieving the full list from cache and filtering it in the application manually.

While the "Browse" pages are the main way of accessing the content in the library, there are also "Recently added" pages for both movies and TV shows. These pages show the newest items in the library. The "Recently added movies" page looks just like the browse page except that it has no filter mechanism. The corresponding page for TV shows on the other hand has a slightly different look. It is a large list where the season and episode, plot and runtime of the episode can be seen (see Figure 18).



Recently added episodes

You can click the show name to jump directly to the show details page

| TV Show | Episode | Title | Plot | Runtime |
|---|---|---|---|---|
| Boardwalk Empire | 5x07 Watch | Friendless Child | Nucky tries to hold onto his assets in Atlantic City. Willie and Eli become involved in Nucky's war. 720 H.264 DOLBY DIGITAL 5.1 16:9 | 56 min |
| Homeland | 4x04 Watch | Iron in the Fire | Carrie gets a tip. Saul calls in a favor. Fara uncovers a conspiracy. 720 H.264 DOLBY DIGITAL 5.1 16:9 | 51 min |
| The X-Files | 2x25 ✓ Watch | Anasazi (1) | When Mulder obtains what could be the original and uncut MJ documents containing information about the government's knowledge of extraterrestrial life; his shadowy enemies step up their harassment of him, eventually leading to his apparent murder. 720 H.264 DOLBY DIGITAL 2.0 16:9 | 45 min |
| The X-Files | 2x17 ✓ Watch | End Game (2) | The alien bounty hunter kidnaps Scully and wants to trade her for Mulder's sister, who is revealed to be a clone herself. After the trade goes badly and Mulder discovers the truth about Samantha, he tracks the bounty hunter to his ship buried in the arctic ice and demands the whereabouts of his real sister. 720 H.264 DOLBY DIGITAL 2.0 16:9 | 87 min |
| The X-Files | 2x16 ✓ | Colony (1) | After the obituaries of four identical men are e-mailed to Mulder, the agents are contacted by a shadowy CIA agent who tells them about a bounty hunter sent to wipe out the beginning of a colony of clones. Meanwhile Mulder is distracted by the sudden | 87 min |

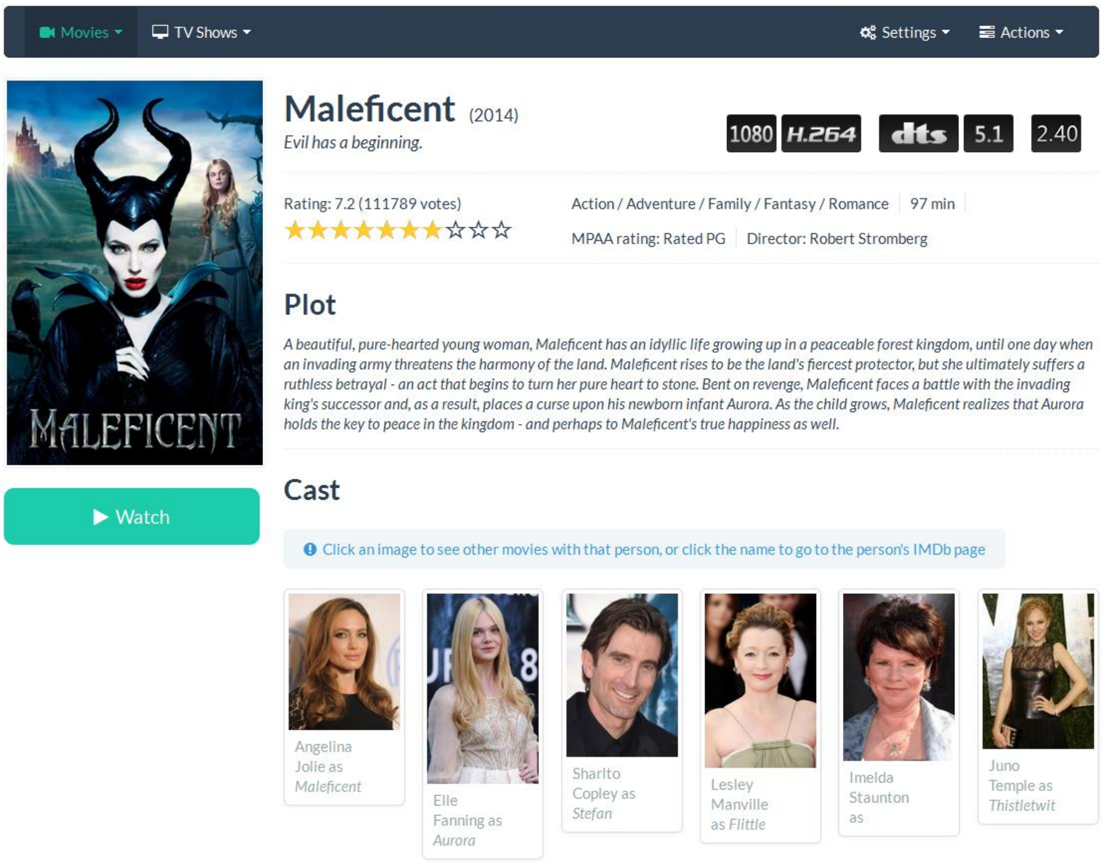
*Figure 18 the Recently added episodes page*

### 4.3.3 Details pages

In the context of this application, a "details" page is one you get to when you click an item on the browse page (be it a movie or a TV show). The content and visual appearance of these pages varies depending on the content type.

#### *Movie details*

A movie details page is quite simple. It fetches information about the movie from the API and displays it on the page. In Figure 19, a typical details page is shown. It contains basic information about the movie itself, such as title, year, genre and plot, as well as the so called stream details (seen as black badges) which describe the technical details of the file in question. In this particular case we're dealing with a standard Blu-Ray version of a movie.



Maleficent (2014)
*Evil has a beginning.*

Rating: 7.2 (111789 votes)
★★★★★★★☆☆☆

Action / Adventure / Family / Fantasy / Romance | 97 min
MPAA rating: Rated PG | Director: Robert Stromberg

Plot

*A beautiful, pure-hearted young woman, Maleficent has an idyllic life growing up in a peaceable forest kingdom, until one day when an invading army threatens the harmony of the land. Maleficent rises to be the land's fiercest protector, but she ultimately suffers a ruthless betrayal - an act that begins to turn her pure heart to stone. Bent on revenge, Maleficent faces a battle with the invading king's successor and, as a result, places a curse upon his newborn infant Aurora. As the child grows, Maleficent realizes that Aurora holds the key to peace in the kingdom - and perhaps to Maleficent's true happiness as well.*

Cast

Click an image to see other movies with that person, or click the name to go to the person's IMDb page

Angelina Jolie as *Maleficent*
Elle Fanning as *Aurora*
Sharlto Copley as *Stefan*
Lesley Manville as *Flittle*
Imelda Staunton as
Juno Temple as *Thistlewit*

*Figure 19 A details page for a movie*

Even though some of the visual elements are very similar to the details page for TV shows (described in the next chapter) most of the code is not shared. However, what is shared is the widget that renders the buttons for watching an item and its download links.

When the button is pressed, a small dialog opens and the user is presented with various options for how to watch the media item (Figure 20). Currently these options are available:

- *Play the item on the backend*. This option plays the file on the screen connected to the current backend. Only administrators are able to see and use this button.
- *Download as a playlist*. This option serves a small playlist file to the user which can then be opened in a media player in order to play the underlying file or files. The playlist format used can be selected before pressing the button.

- *Watch in browser*. This option redirects the user to an in-browser video player that then streams and plays the file directly in the browser. This option is only available for files which are in a format known to be playable directly in the major browsers.
- Direct link to the item. Useful for downloading the file locally for later viewing.

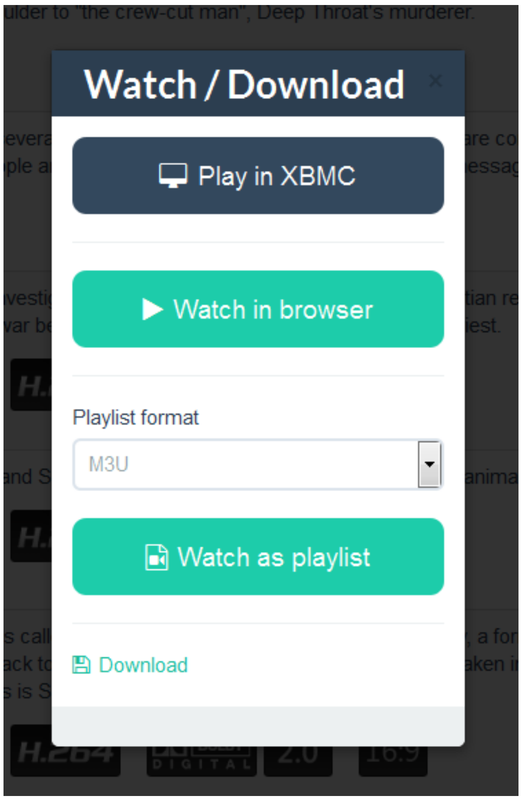


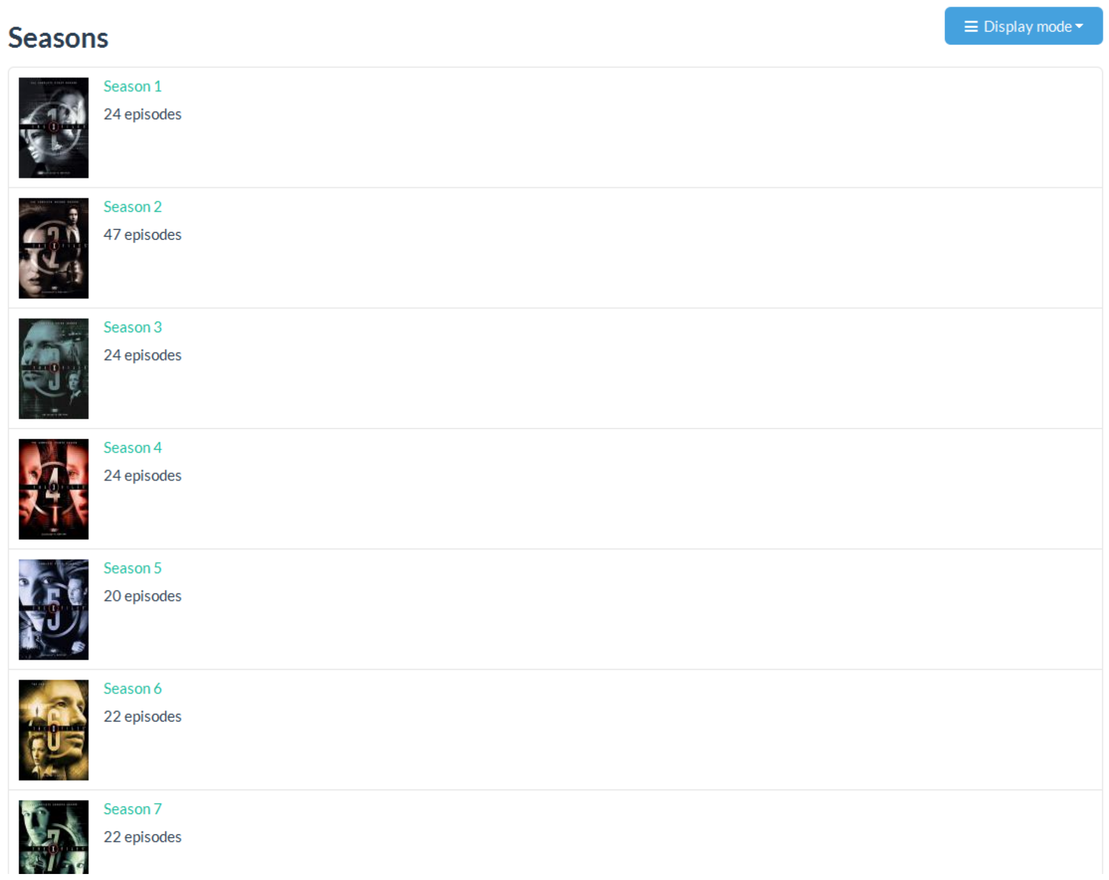
*Figure 20 Watch/Download dialog*

Currently the supported playlist formats are M3U, XSPF and PLS. The reason for supporting many different formats is that some players may only handle some of them. XSPF, being the most advanced format, also supports embedding URLs to artwork. Depending on the media player used to play the playlist, these images may or may not be displayed.

Playlist files are always served as attachments, which means that the browser always opens the “Save file as” dialog instead of attempting to open the file itself [42].

### *TV show details*

Like a movie details page, a TV show details page starts with some basic information about the show. After the list of actors there's a list of all the seasons available in the library (if there's only one season the list is not collapsed). The list of seasons can, like the browse pages, be displayed either as a list or as a grid. When displayed in grid mode, clicking a season simply takes you to a new page where the episodes are listed. When displayed in list mode, the seasons are shown in a collapsed list complete with season art and the number of episodes it contains. Clicking a season link fetches the list of episodes for that season asynchronously using AJAX and opens that part of the list to reveal the individual episodes. This is a necessity since some shows have a large amount of episodes (e.g. talk shows that often air on a daily basis) and rendering everything during the initial page load takes a long time. Figure 21 shows a typical season list in its collapsed form.



Season 1
24 episodes

Season 2
47 episodes

Season 3
24 episodes

Season 4
24 episodes

Season 5
20 episodes

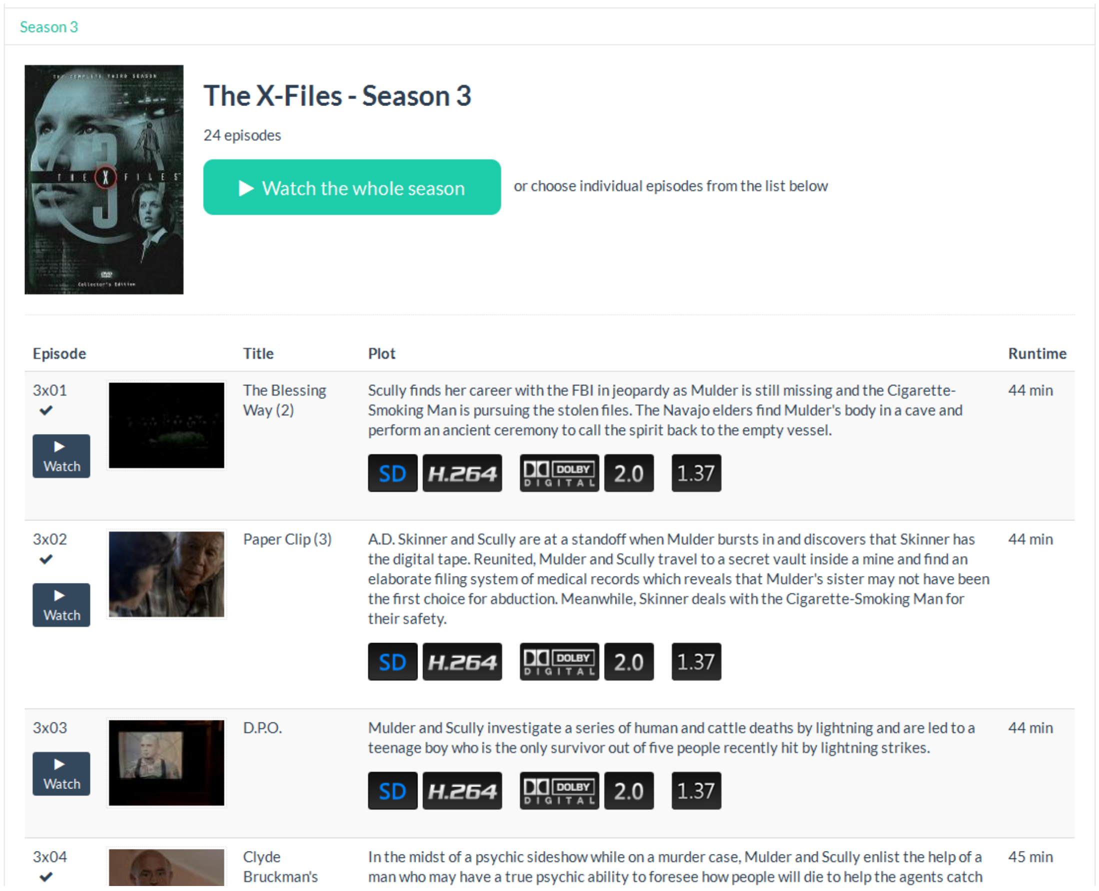
Season 6
22 episodes

Season 7
22 episodes

*Figure 21 Grid view of the available seasons for a TV show*

When a season is expanded, the artwork is slightly enlarged and the list of episodes is rendered. Each episode row contains a button for watching or downloading the episode in question (using the same dialog as for movies) as well as the title, plot and runtime of the episode.

In addition to the episode list there's a large “Watch the whole season” button which serves a playlist containing all the episodes of that season to the browser (as seen in Figure 22). This way the user can easily watch more than one episode and switch between them without having to reopen the browser. A playlist for the whole show is also available under the TV show poster further up on the page. Unlike when attempting to watch movies or episodes, with these buttons there is no way of selecting which playlist format should be used. Instead, the default playlist format specified by the application settings is used.



Season 3

The X-Files - Season 3

24 episodes

Watch the whole season — or choose individual episodes from the list below

| Episode | | Title | Plot | Runtime |
|---|---|---|---|---|
| 3x01 ✓ Watch | | The Blessing Way (2) | Scully finds her career with the FBI in jeopardy as Mulder is still missing and the Cigarette-Smoking Man is pursuing the stolen files. The Navajo elders find Mulder's body in a cave and perform an ancient ceremony to call the spirit back to the empty vessel. SD H.264 DOLBY DIGITAL 2.0 1.37 | 44 min |
| 3x02 ✓ Watch | | Paper Clip (3) | A.D. Skinner and Scully are at a standoff when Mulder bursts in and discovers that Skinner has the digital tape. Reunited, Mulder and Scully travel to a secret vault inside a mine and find an elaborate filing system of medical records which reveals that Mulder's sister may not have been the first choice for abduction. Meanwhile, Skinner deals with the Cigarette-Smoking Man for their safety. SD H.264 DOLBY DIGITAL 2.0 1.37 | 44 min |
| 3x03 Watch | | D.P.O. | Mulder and Scully investigate a series of human and cattle deaths by lightning and are led to a teenage boy who is the only survivor out of five people recently hit by lightning strikes. SD H.264 DOLBY DIGITAL 2.0 1.37 | 44 min |
| 3x04 ✓ | | Clyde Bruckman's | In the midst of a psychic sideshow while on a murder case, Mulder and Scully enlist the help of a man who may have a true psychic ability to foresee how people will die to help the agents catch | 45 min |

*Figure 22 Example of an expanded season listing all the episodes for that season*

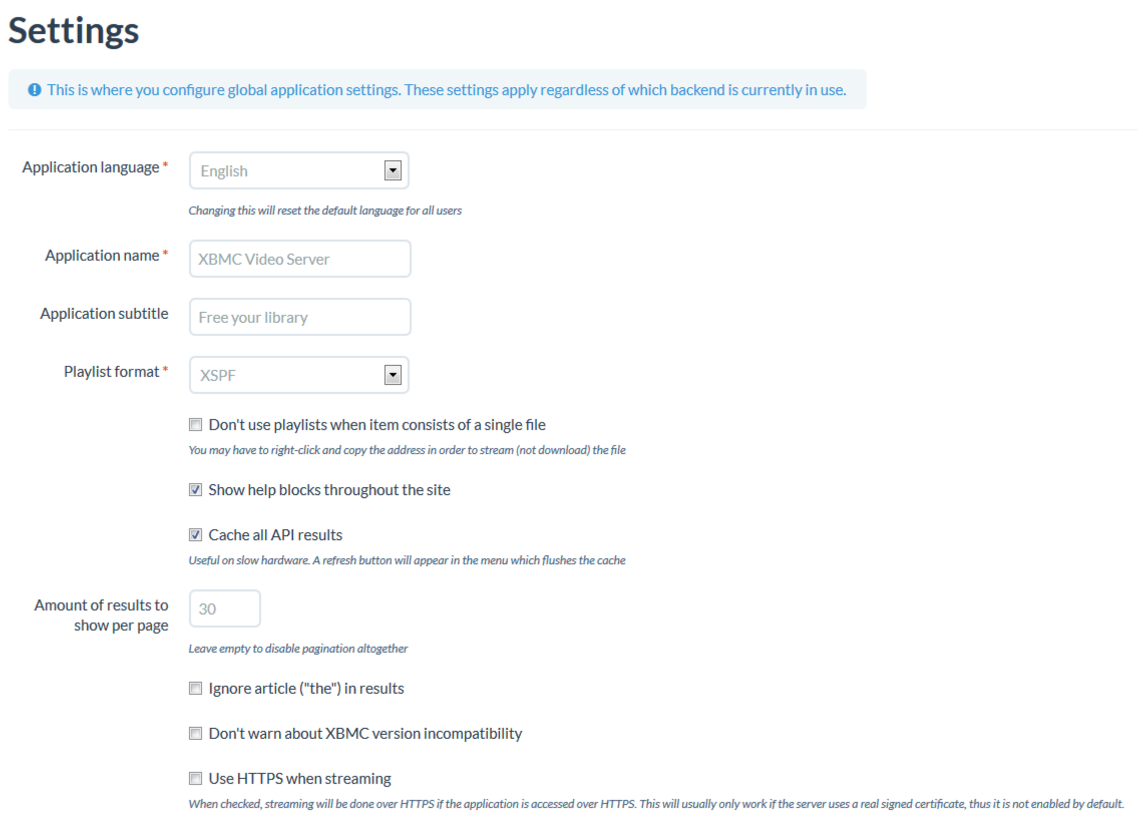
### 4.3.4 Settings and administration

The application is designed so that no manual editing of configuration files is necessary. While the application uses a configuration file per se (the main Yii configuration file), it is not supposed to be changed by the end-user. Instead it uses an *SQLite* database which holds information about users, backends and general settings as well as the application log.

The database is created when the application is installed using the installation instructions. A set of commands are run which creates the required tables and populates them with default values. In this step the first and only user is created (named “admin”). Generally the application stores user passwords in hashed form (using the *phpass* library), but in this case the password is specified as plain-text. Once the user logs in for the first time the password will be hashed. The reasoning for this is that phpass uses different hashing algorithms depending on what’s available on the system in question, so it’s a matter of guessing which one to use if the hash is not created using the phpass library directly.

When the user has logged in for the first time he is asked to configure a backend. Using pre-filters the application knows whether a backend has been configured or not and continues to redirect the user to this setup page if he hasn’t done so yet. Additionally, the validation for the form is relatively sophisticated in that it checks that the entered hostname or IP address is actually reachable, then it goes on to check if the server on the other end identifies itself as XBMC (by looking at the HTTP authentication realm string) and finally that the entered API credentials match. This ensures that people don’t accidentally enter the wrong port and end up with a broken installation. It is also fairly common to have some other piece of software involved on the same machine that uses the same default port as XBMC's web server (8080). These validation steps try to make it obvious to the user that it's eventually not XBMC listening on that port.

Once a backend has been created the user is directed to the main page, the “Movies - Browse” page, and is free to create additional users or change the application settings. The settings page (shown in Figure 23) is a simple page that lists all available settings in a form. Most of the settings are related to the user interface or performance related. The global application language can also be set here.



Settings

This is where you configure global application settings. These settings apply regardless of which backend is currently in use.

Application language * : English
*Changing this will reset the default language for all users*

Application name * : XBMC Video Server

Application subtitle : Free your library

Playlist format * : XSPF

☐ Don't use playlists when item consists of a single file
*You may have to right-click and copy the address in order to stream (not download) the file*

☑ Show help blocks throughout the site

☑ Cache all API results
*Useful on slow hardware. A refresh button will appear in the menu which flushes the cache*

Amount of results to show per page : 30
*Leave empty to disable pagination altogether*

☐ Ignore article ("the") in results

☐ Don't warn about XBMC version incompatibility

☐ Use HTTPS when streaming
*When checked, streaming will be done over HTTPS if the application is accessed over HTTPS. This will usually only work if the server uses a real signed certificate, thus it is not enabled by default.*

*Figure 23 Partial view of the application settings page*

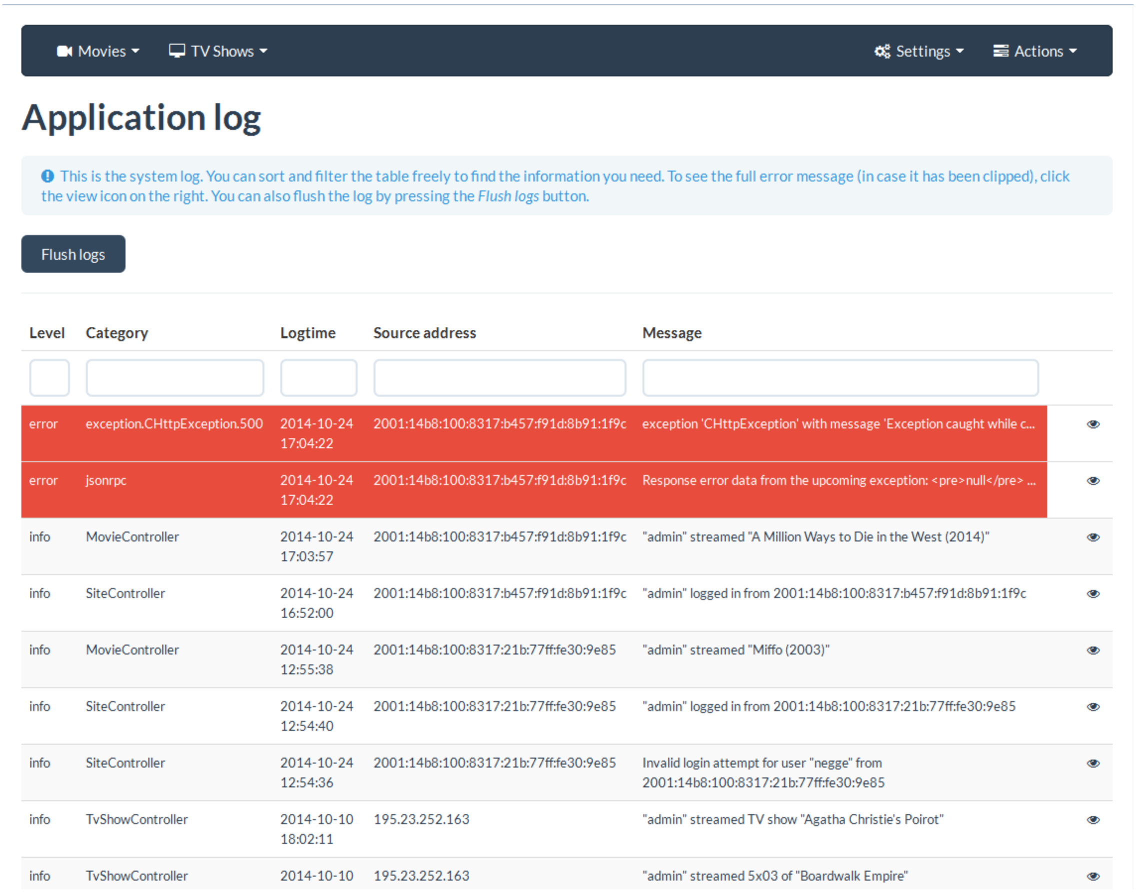
## *System log*

Last but not least, the application keeps a log that can be viewable directly from the interface. By default, Yii logs exceptions and other messages to a file (by default *protected/runtime/application.log*), but to make the log more discoverable I decided to additionally log everything to the SQLite database the application already uses. In Yii this is a simple matter of configuring an additional *log route* to the main log router component.

The system log page (Figure 24) displays all log items as a paginated list. Exceptions (which are logged by default by Yii) are marked red so as to distinguish them from less serious log messages. Each line has a small "eye" icon which when clicked will show more details about the log item. This is most useful for exceptions since the full stack trace will not be visible in the list itself.

Most user actions are logged, meaning that an administrator can keep tabs on successful and unsuccessful login attempts and who is watching what. The mechanism which determines whether an item has been streamed or downloaded isn't very smart;

an item is logged as watched as soon as the watch button or download link is clicked. In the case of the download link (which is a direct link to the file, i.e. it doesn't get handled by the application itself) a small piece of JavaScript sends a logging request to the server using AJAX. All other actions are logged directly on the server side.



Application log

This is the system log. You can sort and filter the table freely to find the information you need. To see the full error message (in case it has been clipped), click the view icon on the right. You can also flush the log by pressing the *Flush logs* button.

| Level | Category | Logtime | Source address | Message |
|---|---|---|---|---|
| error | exception.CHttpException.500 | 2014-10-24 17:04:22 | 2001:14b8:100:8317:b457:f91d:8b91:1f9c | exception 'CHttpException' with message 'Exception caught while c... |
| error | jsonrpc | 2014-10-24 17:04:22 | 2001:14b8:100:8317:b457:f91d:8b91:1f9c | Response error data from the upcoming exception: &lt;pre&gt;null&lt;/pre&gt; ... |
| info | MovieController | 2014-10-24 17:03:57 | 2001:14b8:100:8317:b457:f91d:8b91:1f9c | "admin" streamed "A Million Ways to Die in the West (2014)" |
| info | SiteController | 2014-10-24 16:52:00 | 2001:14b8:100:8317:b457:f91d:8b91:1f9c | "admin" logged in from 2001:14b8:100:8317:b457:f91d:8b91:1f9c |
| info | MovieController | 2014-10-24 12:55:38 | 2001:14b8:100:8317:21b:77ff:fe30:9e85 | "admin" streamed "Miffo (2003)" |
| info | SiteController | 2014-10-24 12:54:40 | 2001:14b8:100:8317:21b:77ff:fe30:9e85 | "admin" logged in from 2001:14b8:100:8317:21b:77ff:fe30:9e85 |
| info | SiteController | 2014-10-24 12:54:36 | 2001:14b8:100:8317:21b:77ff:fe30:9e85 | Invalid login attempt for user "negge" from 2001:14b8:100:8317:21b:77ff:fe30:9e85 |
| info | TvShowController | 2014-10-10 18:02:11 | 195.23.252.163 | "admin" streamed TV show "Agatha Christie's Poirot" |
| info | TvShowController | 2014-10-10 | 195.23.252.163 | "admin" streamed 5x03 of "Boardwalk Empire" |

*Figure 24 The system log*

# 5. CONCLUSIONS

The Yii framework is a serious alternative to similar frameworks. It is easy to use, even for beginners, and the documentation is excellent. If you're about to design something common like a blog or a web forum I wouldn't perhaps recommend using Yii in the first place since you'd be reinventing the wheel on quite a few levels, but for any application that is a little more unique it is a good choice.

Despite all that, Yii has some major short-comings, and while most of them are rectified in the upcoming Yii 2.0 version (currently in beta) the current stable version is likely going to be around for a while, so it's good to be aware of them. One drawback is the fact that the framework is largely monolithic. Unlike Zend and Symfony it is not possible to use only a small part of Yii; you either use it or you don't. This is perhaps not a big issue if you're developing a somewhat large web application since chances are you'll be using most of the framework's functionality anyway, but for smaller, more niche projects it can feel a bit wasteful. The monolithic design also means it's difficult, if not impossible, to replace certain parts of an application with custom implementations.

XBMC Video Server has come a long way since I started working on it in June 2013. Despite being a niche product it has gathered a lot of interest on the Internet, especially after I posted about it on the XBMC forums [43]. The fact that it is actually used by people, combined with the fact that the source code is free for everyone to look at and use has made me strive toward keeping the code as clean and bug free as possible.

It is hard to come up with exact usage numbers since there are multiple ways of acquiring the source code for the application. It is mainly done by cloning the git repository, but users can also download a tagged revision as a ZIP archive. Additionally, Github lets repository owners see statistics only from the last two weeks. Packagist on the other hand keeps a running counter of the number of times a published library has been downloaded using Composer. Here are some actual numbers as of November 26th 2014:

- *XBMC Video Server*: 54 clones (34 unique), 2471 views (535 unique visitors) during the last two weeks (12-26 November) [44]
- *simple-json-rpc-client*: 1967 installs through Packagist [45]
- *php-whitelist-check*: 2343 installs through Packagist [46]

Even though I'm happy with the current state of the application, there are still some areas that could be improved. I've gotten great feedback from a static analyzer service named Scrutinizer that has prompted me to refactor some parts of the application, mostly for the better [47].

For anyone who has used Yii in combination with SQL databases, the fact that XBMC Video Server bypasses the active record model completely means that the manipulation of movies and TV shows becomes a bit more complicated. The idea of mapping raw results to classes using jsonmapper makes it a bit more intuitive, but it's still not a replacement for being able to type something like `Movie::model()->findAll()`. I did consider creating an active record implementation of the API, but Yii's active record concept is very intimately tied to PHP *PDO*, so that approach was simply infeasible. This is yet another example of the drawbacks of a monolithic framework design.

The application is still missing one big feature – transcoding of video to arbitrary formats. This is something that is necessary if you want to be able to watch any video file directly in your browser or on limited platforms such as Apple devices. There is a proof-of-concept available which uses *FFmpeg* in a background process to pipe the raw transcoded data to the browser (which in turn is served to the client). While it works and I believe the foundation is solid, the feature set is lacking and the user interface isn't as good as it could be. Add to that the fact that there are still a lot of unresolved questions, like if seeking in a transcoded movie will be possible at all or if it's possible to perform a live transcode to MP4 at all.

Coding issues aside, one thing I would have liked for the project was to get more code contributions from the community. The majority of the code has been written by me, though there has been at least one large pull request from the community [48]. People have also contributed French and German translations. I haven't been able to determine why the number of contributions has been fairly low, though it could of course be a sign that the application is fairly bug-free and no one has had a reason to dig into the code. People have hinted that I should try to extend the range of platforms it is available on (notably to *OpenELEC* [49], a barebones HTPC operating system mainly designed to run XBMC) which could hopefully trigger the interest of new people, some of which may even contribute something code-wise.

# REFERENCES

1. Jalle19/xbmc-video-server. *github.com.* [Online] [Citat: den 11 12 2014.] https://github.com/Jalle19/xbmc-video-server.

2. Composer. *getcomposer.org.* [Online] [Citat: den 11 12 2014.] https://getcomposer.org/.

3. Amazon.com: yii. *amazon.com.* [Online] [Citat: den 24 11 2014.] http://www.amazon.com/s/ref=nb_sb_noss/188-0994387-5830256?url=search-alias%3Daps&field-keywords=yii.

4. Tutorials | Yii PHP Framework. *www.yiiframework.com.* [Online] [Citat: den 11 12 2014.] http://www.yiiframework.com/tutorials/.

5. BibWord Microsoft Word Citation and Bibliography styles - Download; ISO 690 - Numeric Reference with Square Brackets. *bibword.codeplex.com.* [Online] [Citat: den 24 11 2014.] https://bibword.codeplex.com/releases/view/14646.

6. Top 10 content management systems. *www.webdesignerdepot.com.* [Online] [Citat: den 12 11 2014.] http://www.webdesignerdepot.com/2011/10/top-10-content-management-systems/.

7. CMS comparision - WordPress vs Joomla vs Drupal. *websitesetup.org.* [Online] [Citat: den 11 12 2014.] http://websitesetup.org/cms-comparison-wordpress-vs-joomla-drupal/.

8. Sitepoint - Framework popularity, end of 2013. *www.sitepoint.com.* [Online] [Citat: den 27 5 2014.] http://www.sitepoint.com/best-php-frameworks-2014/.

9. Varnish Community | Varnish makes websites fly! *www.varnish-cache.org.* [Online] [Citat: den 27 10 2014.] https://www.varnish-cache.org/.

10. Testing: Overview, The Definitive Guide to Yii. *www.yiiframework.com.* [Online] [Citat: den 27 05 2014.] http://www.yiiframework.com/doc/guide/1.1/en/test.overview.

11. Yii 2.0 Beta is released. *www.yiiframework.com.* [Online] [Citat: den 22 05 2014.] http://www.yiiframework.com/news/77/yii-2-0-beta-is-released/.

12. The Definitive Guide to Yii | Yii PHP Framework. *yiiframework.com.* [Online] [Citat: den 24 11 2014.] http://www.yiiframework.com/doc/guide/.

13. About Yii. *www.yiiframework.com.* [Online] [Citat: den 27 5 2014.] http://www.yiiframework.com/about/.

14. PHP: New features. *www.php.net.* [Online] [Citat: den 27 5 2014.] http://www.php.net/manual/en/migration53.new-features.php.

15. Yii PHP Framework: Best for Web 2.0 Development. *www.yiiframework.com.* [Online] [Citat: den 24 10 2014.] http://www.yiiframework.com/.

16. yiisoft/yii. *packagist.org.* [Online] [Citat: den 11 12 2014.] https://packagist.org/packages/yiisoft/yii.

17. Fundamentals: Conventions, The Definitive Guide To Yii. *www.yiiframework.com.* [Online] [Citat: den 27 5 2014.] http://www.yiiframework.com/doc/guide/1.1/en/basics.convention#directory.

18. xbmc-video-server/Backend.php at master · Jalle19/xbmc-video-server. *github.com.* [Online] [Citat: den 11 12 2014.] https://github.com/Jalle19/xbmc-video-server/blob/master/src/protected/models/Backend.php.

19. Getting Started - Composer. *getcomposer.org.* [Online] [Citat: den 27 5 2014.] https://getcomposer.org/doc/00-intro.md.

20. PSR | Petermoulding.com. *petermoulding.com.* [Online] [Citat: den 27 5 2014.] http://petermoulding.com/php/psr#psr-0.

21. Package versions, Composer documentation. *getcomposer.org.* [Online] [Citat: den 27 5 2014.] https://getcomposer.org/doc/01-basic-usage.md#package-versions.

22. composer.lock – The Lock File, Basic Usage. *getcomposer.org.* [Online] [Citat: den 27 5 2014.] https://getcomposer.org/doc/01-basic-usage.md#composer-lock-the-lock-file.

23. Packagist. *packagist.com.* [Online] [Citat: den 24 10 2014.] http://packagist.org/.

24. Yii & composer modules autoload (1.1.14-RC), Issue #2642, yiisoft/yii, Github. *github.com.* [Online] [Citat: den 27 5 2014.] https://github.com/yiisoft/yii/issues/2642.

25. Internet Connection Speed Recommendations, Netflix. *help.netflix.com.* [Online] [Citat: den 27 5 2014.] https://help.netflix.com/en/node/306.

26. h5ai – a modern HTTP web server index for Apache httpd, lighttpd, nginx and Cherokee. *larsjung.de.* [Online] [Citat: den 28 5 2014.] http://larsjung.de/h5ai/.

27. 6 Subsonic Alternatives – TechSchout. *www.techshout.com.* [Online] [Citat: den 28 5 2014.] http://www.techshout.com/alternatives/2013/09/subsonic-alternatives/.

28. ID3.org – The MP3 Tag Standard. *id3.org.* [Online] [Citat: den 28 5 2014.] http://id3.org/.

29. WhyRAR.omfg.se! *whyrar.omfg.se.* [Online] [Citat: den 28 5 2014.] http://whyrar.omfg.se/index_eng.php.

30. Use VLC to Play Videos Inside [sic] an RAR File. *lifehacker.com.* [Online] [Citat: den 28 5 2014.] http://lifehacker.com/5922124/use-vlc-to-play-videos-inside-an-rar-file.

31. XBMC | Open Source Home Theatre Software. *xbmc.org.* [Online] [Citat: den 24 10 2014.] http://xbmc.org/.

32. XBMC Is Getting a New Name – Introducing Kodi 14 | XBMC. *xbmc.org.* [Online] [Citat: den 24 10 2014.] http://xbmc.org/introducing-kodi-14/.

33. About | XBMC. *xbmc.org.* [Online] [Citat: den 28 05 2014.] http://xbmc.org/about/.

34. 2.5 Virtual File System /vfs, Webserver – XBMC. *wiki.xbmc.org.* [Online] [Citat: den 28 5 2014.] http://wiki.xbmc.org/index.php?title=Webserver.

35. Transports & Functionalities – JSON-RPC API – XBMC. *wiki.xbmc.org.* [Online] [Citat: den 28 5 2014.] http://wiki.xbmc.org/index.php?title=JSON-RPC_API.

36. Read only list of TV and Movies · Issue #48 · Jalle19/xbmc-video-server · GitHub. *github.com.* [Online] [Citat: den 24 10 2014.] https://github.com/Jalle19/xbmc-video-server/issues/48.

37. JSON-RPC API – API versions. *kodi.wiki.* [Online] [Citat: den 24 10 2014.] http://kodi.wiki/view/JSON-RPC_API#API_versions.

38. Performance improvements by Jalle19 • Pull Request #8 • netresearch/jsonmapper • GitHub. *github.com.* [Online] [Citat: den 19 10 2014.] https://github.com/netresearch/jsonmapper/pull/8.

39. eventviva/php-image-resize. *github.com.* [Online] [Citat: den 28 5 2014.] https://github.com/eventviva/php-image-resize.

40. jQuery Unveil - A very lightweight plugin to lazy load images. *luis-almeida.github.io.* [Online] [Citat: den 24 10 2014.] https://luis-almeida.github.io/unveil/.

41. John Resig - HTML 5 data- Attributes. *ejohn.org.* [Online] [Citat: den 24 11 2014.] http://ejohn.org/blog/html-5-data-attributes/.

42. PHP: header - Manual. [Online] [Citat: den 28 05 2014.] http://php.net/manual/en/function.header.php.

43. XBMC Video Server: stream/download your library contents. *forum.xbmc.org.* [Online] [Citat: den 24 10 2014.] http://forum.xbmc.org/showthread.php?tid=168296.

44. Screenshot of Github traffic statistics for Jalle19/xbmc-video-server. *github.com.* [Online] [Citat: den 26 11 2014.] http://werket.tlk.fi/~negge/images/github_traffic_screenshot.PNG.

45. jalle19/simple-json-rpc-client - Packagist. *packagist.org.* [Online] [Citat: den 26 11 2014.] https://packagist.org/packages/jalle19/simple-json-rpc-client.

46. jalle19/php-whitelist-check - Packagist. *packagist.org.* [Online] [Citat: den 26 11 2014.] https://packagist.org/packages/jalle19/php-whitelist-check.

47. Code Quality Summary - Jalle19/xbmc-video-server - Measure and Improve Code Quality continuously with Scrutinizer. *scrutinizer-ci.com.* [Online] [Citat: den 24 11 2014.] https://scrutinizer-ci.com/g/Jalle19/xbmc-video-server/.

48. Option to power off the backend by pweisenburger • Pull Request #206 • Jalle19/xbmc-video-server • GitHub. *github.com.* [Online] [Citat: den 24 10 2014.] https://github.com/Jalle19/xbmc-video-server/pull/206.

49. OpenELEC Mediacenter - Home. *openelec.tv.* [Online] [Citat: den 24 10 2014.] http://openelec.tv/.

# APPENDICE 1 - SAMMANFATTNING PÅ SVENSKA

## 1.1 Introduktion

Webbutveckling är något som utvecklats med rasande takt under de senaste åren. Utvecklare förväntas få mera gjort på kortare tid, vilket innebär att det inte längre är allmän praxis skriva webbsidor helt ”för hand”, utan man använder ofta någon form av hjälpmedel i form av färdigutvecklad mjukvara eller ramverk. När man talar om ramverk i webbutvecklingssammanhang går det att göra några grova indelningar. Dels har vi kompletta CMS-system med hjälp av vilka man enkelt kan skapa relativt enkla webbsidor, såsom bloggar, företagssidor eller mindre webbutiker. Dessa ramverk är delvis riktade till personer som inte har en bakgrund som programmerare. På den andra ändan av skalan finns s.k. *programmeringsramverk*, t.ex. *Zend Framework*, *Symfony* och *Yii*. Det här arbetet handlar om Yii-ramverket samt programmet *XBMC Video Server* som baserar sig på ramverket. Syftet är att ge läsaren en inblick i hur Yii-ramverket används samt ge ett praktiskt exempel på hur pass avancerade program man kan åstadkomma med dess hjälp. Arbetet tar även upp beroendehanteraren *Composer* eftersom den spelar en central roll i hur man bygger upp avancerade webbapplikationer med PHP.

Den här sammanfattningen är strukturerad på samma sätt som själva arbetet, dock används inga källhänvisningar utan läsaren hänvisas till själva huvudtexten.

## 1.2 Yii-ramverket

Yii-ramverket är ett ramverk för PHP som ursprungligen skapats av *Qiang Xue* år 2008. Det här arbetet fokuserar på version 1.1 av Yii, fastän version 2.0 hållit på att utvecklats redan en tid. Yii är ett ramverk för programmerare, till skillnad från exempelvis Wordpress som även kan användas av personer som inte har en bakgrund inom programmering.

För att komma igång med Yii behöver man ladda ner en kopia av själva ramverket och skapa en *grundapplikation*, en s.k. ”skeleton”. Ramverket kan endera laddas ner som ett ZIP-paket från projektets officiella hemsida[1] eller installeras med Composer ge-

[1] http://www.yiiframework.com/

nom att definiera Yii som ett beroende till ens applikation. När man väl har installerat ramverket kan man skapa en grundapplikation med kommandot `yiic webapp`.

Den grundläggande byggstenen i Yii är en *komponent*, `CComponent`. Denna klass erbjuder funktioner såsom *getters* och *setters*, händelse- samt beteéndehantering (eng. *behaviors*). En hel del av den basfunktionalitet som Yii tillgängliggör baserar sig på något som kallas *applikationskomponenter*, en speciell typ av komponenter som är tillgängliga globalt genom `Yii::app()->component`, där ”component” är namnet på den komponent man vill använda.

Utöver komponenter läggs det i arbetet ganska mycket vikt på *modeller* (baserade på klassen `CModel`) och *kontrollers* (baserade på klassen `CController`). I kapitlet om modeller behandlas främst validering av data, såsom den data användaren skickat till servern via ett webbformulär. Då kontrollers diskuteras ligger fokus mycket på det som i Yii kallas för *filter*. Filter är bitar av kod som körs endera före eller efter en *aktion* (en aktion är en metod i kontrollern som anropas beroende på vilken sökväg som anges i webbläsaren) och kan påverka om aktionen ska köras eller inte. Andra helheter som tas upp i det här kapitlet är vyer och layouter, så kallade *widgets* (klassbaserade vyer för mer komplexa eller dynamiska delar av ett användargränssnitt) samt ett lite längre stycke om mellanlagring av data (eng. *caching*).

## 1.3 Composer

*Composer* är en s.k. beroendehanterare (eng. *dependency manager*) för PHP. En beroendehanterare används för att på ett enkelt och standardiserat sätt inkludera tredjepartsprogramvara (såsom ofta använda bibliotek) i sitt projekt utan att manuellt behöva ladda ner rätt versioner.

Composer baserar sig på två filer, *composer.json* och *composer.lock*. Den förstnämnda innehåller information om ens applikation, och kanske viktigast av allt de beroenden som applikationen har. Via *composer.json* kan man specificera exakt vilken version av ett visst paket man vill ha. Efter att man installerat de paket man definierat skapar Composer automatiskt *composer.lock*-filen. Denna fil innehåller information om exakt vilken version som installerats för att säkerhetsställa att framtida installationer får den version som utvecklaren hade tänkt sig.

För att Composer ska kunna hitta de paket man definierat är det enklast om paketen är publicerade på tjänsten *Packagist*. Man kan dock även använda egna paket, t.ex. internt utvecklade bibliotek som man inte vill publicera öppet på internet, genom att i *composer.json* definiera från vilken webbadress Composer kan hämta en kopia av paketet.

## 1.4 XBMC Video Server

Dagens internetuppkopplingar har blivit såpass snabba att det är möjligt att se på eller *strömma* videofiler direkt utan att behöva ladda ner filerna på förhand. Det traditionella sättet att se på strömmad video är att använda sig av en tredjeparts nättjänst såsom *Youtube* eller *Netflix*. Med dagens anslutningar är dock uppladdningshastigheten ofta tillräcklig för att strömma video från sin egen anslutning och på så vis inte behöva använda en tredje part över huvud taget.

*XBMC Video Server* är en applikation som möjliggör detta scenario på ett relativt unikt sätt. Applikationen är ett fristående webbaserat gränssnitt till det populära underhållningscentret XBMC. Både XBMC och XBMC Video Server är fri mjukvara. Via webbgränssnittet kan man bläddra bland de filmer och TV-serier som man har i sitt XBMC-bibliotek för att sedan strömma eller ladda ner dem för senare bruk.

Liknande lösningar har funnits för musik under en lång tid, men för strömning av video har situationen varit sämre. Detta beror framför allt på att man i ljudfiler oftast har s.k. ”metadata” inbäddad i filen, något man oftast inte har i videofiler. Detta är inte ett problem för XBMC Video Server eftersom den får all denna metadata från XBMC via dess API. XBMC i sig hämtar informationen från olika internetkällor, t.ex. IMDb.

XBMC Video Server kommunicerar med XBMC med hjälp av *JSON-RPC*-protokollet. JSON-RPC är ett transportagnostiskt protokoll, vilket innebär att specifikationen inte ställer några krav på hur själva kommunikationen ska ske. En av de vanligaste metoderna är att utföra kommunikationen över HTTP POST-anrop, och det är den metoden som XBMC Video Server använder sig av. För att använda JSON-RPC från PHP var jag tvungen att skriva ett eget bibliotek för det som jag döpte till *simple-json-rpc-client*. Även detta bibliotek är tillgängligt på internet som fri mjukvara.

XBMC Video Server har ett ganska enkelt användargränssnitt. Efter att man loggat in ombes man skapa en ny ”backend”, d.v.s. en instans av XBMC som applikationen ansluter till. Då det är gjort kan man bläddra och filtrera bland instansens filmer och TV-serier. Då man klickar på en film får man upp en sida där information om filmen visas samt en ”Watch”-knapp som ger olika alternativ för att se eller ladda ner filmen. Man kan välja att ladda ner en spellista som en fil (i *M3U*-format, men det går att ändra) eller ladda ner filen i sig för att kunna se den utan internetuppkoppling.

Vyerna för TV-serier påminner mycket om dess motsvarigheter för filmer. Man kan bläddra bland de tillgängliga TV-serierna, och då man klickat på en serie kommer man till en sida med information om serien samt en lista på de säsonger som finns tillgängliga. Genom att klicka på en säsong utvidgas en lista på alla avsnitt som finns tillgängliga för den valda säsongen. Här kan man välja att endera se på enskilda avsnitt eller ladda ner en spellista för hela säsongen så att man enkelt kan hoppa mellan olika avsnitt.

Förutom möjligheten att bläddra bland innehållet i biblioteket går det även att göra administrativa saker som att lägga till och ta bort användare och ändra inställningar för hela applikationen (språk, hur resultat visas o.s.v.). Det finns dessutom en logg varifrån man kan se detaljerad information om eventuella fel som uppstått i applikationen. Detta har varit till stor nytta eftersom man lätt kan få bra och relevant information från användare som rapporterat buggar.

## 1.5 Slutsatser

Yii-ramverket är ett seriöst alternativ bland de olika ramverk som finns idag. Det är ganska lätt att använda, även för nybörjare, och dokumentationen är utmärkt. Det är inte att rekommendera för enkla sidor som bloggar eller webbforum (där de ofta finns färdiga lösningar) men för lite mer unika tillämpningar lämpar det sig väl.

Allt detta till trots har ramverket en del nackdelar. Det är bl.a. monolitiskt, vilket innebär att det inte på ett enkelt sätt går att använda endast en del av ramverket (något som Zend och Symfony är kända för). De flesta av dess negativa sidor har åtgärdats på ett eller annat sätt i den kommande versionen (2.0), men det lär ännu ta en tid innan den nuvarande versionen börjar fasas ut helt.

XBMC Video Server har gått ganska långt sen utvecklingsarbetet påbörjades i juni 2013. Fastän det är en ganska nischad produkt har många på internet fått upp intresset för den, vilket även reflekteras i statistiken över nerladdningar. Bara under en två veckors period i november 2014 (över ett år sedan produkten lanserades officiellt) hade projektets sida 2471 visningar och 54 installationer (baserat på Githubs statistik). Jag skulle vilja påstå att programmet är relativt buggfritt, något som jag strävat efter eftersom vem som helst kan läsa källkoden.

Det finns fortfarande en del funktionalitet som jag gärna skulle se implementerad men inte har haft tid med, först och främst ”omkodning” av video till format som dagens webbläsare är kompatibla med och därmed kan spela upp direkt. På så vis skulle man inte behöva använda en extern mediaspelare på den enhet man spelar upp videon på.