# Neato SmartApps Backend – Overview

Raja Software

Last updated: 05/28/2013

|  |  |  |
| --- | --- | --- |
| Date | Version | Author/Comments |
| 12/11/2012 | 0.1 | Initial version |
| 12/17/2012 | 0.2 | Adding details on JavaScript, CSS and Testing |
| 12/18/2012 | 0.3 | Added server side component section, test console details |
| 12/20/2012 | 0.4 | Adding more details and reasons behind some of the decisions |
| 02/07/2013 | 0.5 | Minor tweaks, checked into Neato SVN |
| 05/28/2013 | 0.6 | Updated for RabbitMQ |

# Summary

This document gives an overview of the Neato SmartApps web application and it touches upon the various technical details of the implementation, and provides some context as to the decisions made in choosing these technologies.

# Technologies

This section briefly explains all the technologies that are used on the Neato SmartApps server.

### Programming Language

This web application is developed using PHP 5. PHP 5 is Object Oriented and complete server side coding is done in Object Oriented fashion. PHP is better suited for this application because:

* PHP has been proved on 3 key points Scalability, Flexibility and Speed.
* It can be further optimized, even "compiled" for higher performance gains.
* PHP gives maximum number of framework choices to pick from (Zend, Yii, CodeIgniter, Symfony, CakePHP etc).
* Because of its C like syntax and huge community support, speed of development is very fast as compared to other programing languages.

Keeping above points in mind, we opted for PHP for Neato Application development.

### Framework

We could have coded the PHP in plain scripting fashion but that would have made it a maintenance nightmare. Instead of plain scripting, we compared various PHP frameworks to decide upon the framework that should be used for Neato project. We compared 4 top PHP frameworks (Symfony, Zend, CodeIgniter and Yii). Eventually we picked Yii framework ([www.yiiframework.com](file:///C:\Temp\n1\Docs\www.yiiframework.com)) based on following salient features of Yii over other frameworks:

* Yii adopts the proven MVC architecture, which allows for clean separation of concerns and hence improves the maintainability of the code. This means, a UI developer can work on UI without knowing any implementation details of the controller/models.
* Yii allows developers to model database data in terms of objects and avoid the tedium and complexity of writing repetitive SQL statements.
* Yii supports automatic generation of complex WSDL service specifications and management of Web service request handling which is important while designing the web services for SmartApp
* Yii is equipped with many security measures to help prevent web applications from attacks such as SQL injection, cross-site scripting (XSS), cross-site request forgery (CSRF), and cookie tampering.
* Yii is the best performing PHP framework <http://www.yiiframework.com/performance/> out there in the market.

In order to be able to upgrade to newer versions of the Yii framework, we have kept the Yii framework’s core files inside **server/yii-1.1.12.b600af** folder. All the custom code is written in another folder and the Yii framework’s core is not touched at all.

### Database

MySQL 5.3 (<http://www.mysql.com/>) is used as database. Complete DB dump is provided inside **server/db\_schema** folder. The naming conventions that have been followed are:

* All the table names are in lower case and plural (**robots** instead of **robot**).
* All the column names are in lower case and singular (**serial\_number** instead of **serial\_numbers**).
* DB Schema is normalized to 3rd normal form to reduce redundancy.
* All the tables have a primary key.
* Tables have proper DB relationships among themselves.
* Searchable columns are indexed accordingly.

### Object Relational Mapper (ORM)

Instead of using direct PHP/MySQL connectivity, and running in to SQL injection security perils, Yii framework’s great ORM support is used. ORM ensures that:

* Instead of writing direct SQL queries, we are using “data objects”. For example instead of writing a query like
  + ***$dbConnectionHandle->execute(“insert into user (name) values (‘James’)”);***

We would be writing code like:

* + ***$user = new User(); // User is a Data object which is mapped to user table***
  + ***$user->setName(“James”);***
  + ***$user->save();***
* All the DB operations are DB agnostic. This means at later point of time, MySQL can be replaced with any other database without any code changes required.
* All these data objects can be cached with any standard caching solution (like Memcached) which can improve the performance drastically.

### XMPP/ Jabber Server

SmartApp requires Jabber server to communicate with the Robots. We are using Ejabberd (http://www.ejabberd.im/) as XMPP server. Ejabberd is used because it is:

* Cross-platform, as Ejabberd can run under Windows and Unix derived systems such as Linux and Mac OS X.
* Distributed, we can run Ejabberd on a cluster of machines and all of them will serve the same Jabber domain(s). In future to support growing number of users and robots would mean simply adding another node to Ejabberd cluster and everything else would work as it is.
* Fault-tolerant, we can deploy an Ejabberd cluster so that all the information required for a working service will be replicated permanently on all nodes. This also means that if one of the nodes crashes, the others will continue working without disruption.
* Administrator Friendly, as Ejabberd is built on top of the Open Source Erlang, we do not need to install an external database or an external web server as everything is already included. It also provides a web based admin interface, with which we can set permissions and monitor the usage of the Ejabberd server(s).

### RabbitMQ

As SmartApp supports few long running tasks, it is imperative that we perform these tasks asynchronously so that Apache performance is not affected. This application uses RabbitMQ to handle long running tasks (like push notification to GCM/iPhone). RabbitMQ (<http://www.rabbitmq.com/>) is a producer/consumer messaging framework. We opted for RabbitMQ because:

* It is open source and robust messaging solution for all type of applications
* It is very easy to integrate and runs on all major operating systems
* There are various wrappers available to encapsulate the actual functioning of the producer/consumers.
* We are using PHPAMQPLib as a PHP based wrapper.

### HTML/CSS

To enable a web 2.0 user experience, a rich UI is developed. All the styling is done using external Style Sheets to avoid code duplication and enhance the maintainability of these files. All the HTML/CSS is validated for W3C ([www.w3c.org](file:///C:\Temp\n1\Docs\www.w3c.org)) compliance.

### JQuery

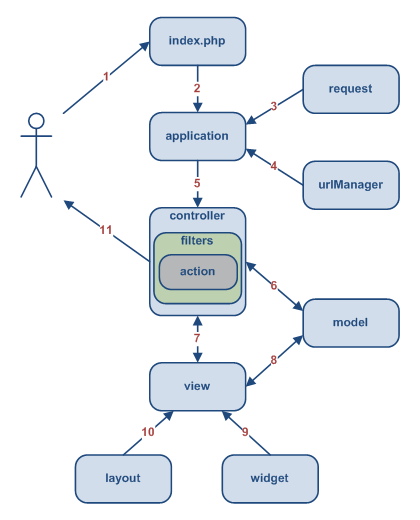
For all the client side scripting, instead of plain JavaScript, JQuery (<http://jquery.com/>) library is used. Few advantages of JQuery over simple JavaScript are :

* JQuery eliminates a lot of cross browser JavaScript issues and normalizes the event model across the browsers. This ensures that the application behavior does not change with the client’s browser.
* JQuery can perform complicated JavaScript operations in very little code. This results in lesser code. For example, if we were reading value from a text box.
  + Using JavaScript it would be
    - ***alert(document.getElementById(‘text\_id’).val);***
  + Using JQuery it would be
    - ***alert($(‘text\_id’).val());***
* It has inbuilt support for Ajax
* Has great support for UI and effects libraries (for example, users and robots listing grids)

Instead of keeping the JQuery code in all the pages, it is kept inside one JavaScript file so that we can improve the performance by enabling the caching of it or moving it to a different server which would be serving only static content.

# Sample HTTP Call

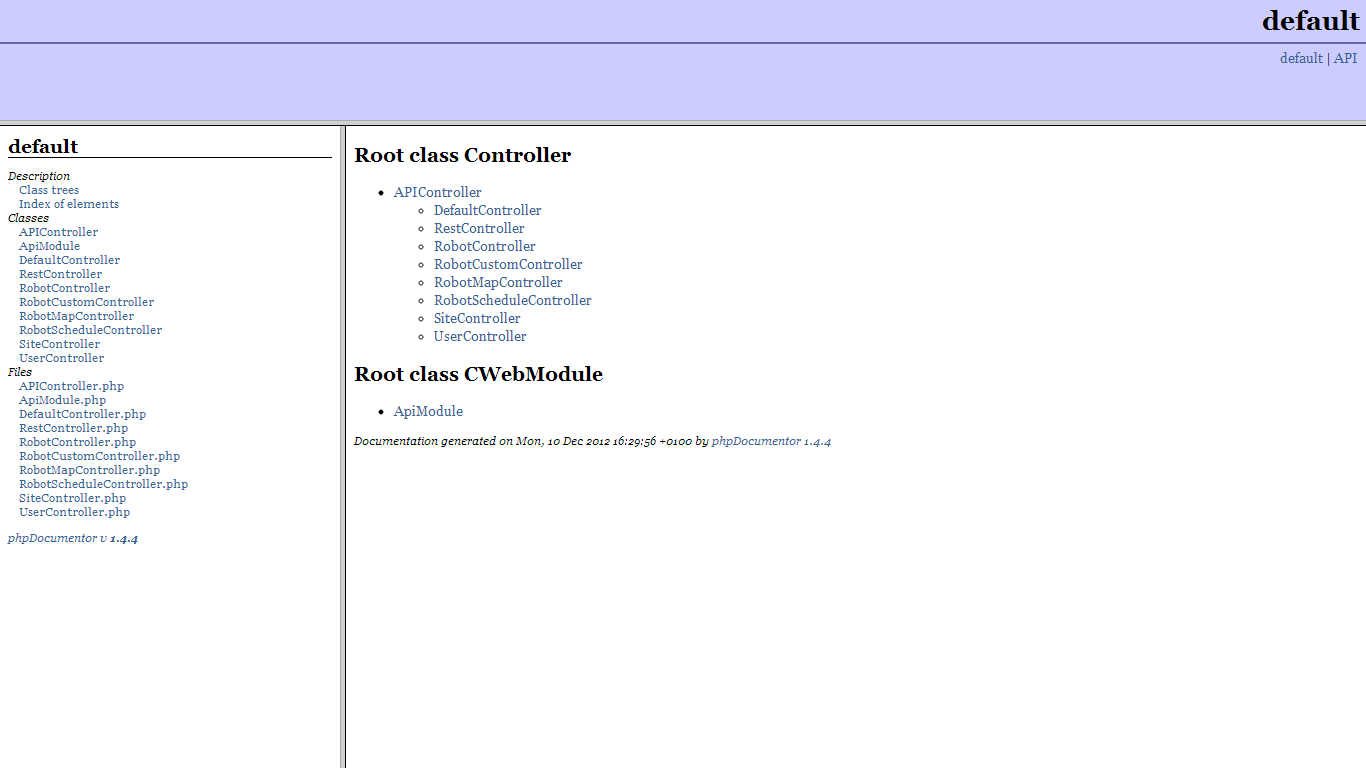
The diagram below depicts a web user making a simple HTTP request to the server. Description below this diagram lists each interaction.



1. Assuming that application is host on [http://myhostname.com](http://neatoroboticswebapp.com) and a user makes a request to URL [http://myhostname.com/server/Neato](http://neatoroboticswebapp.com/server/Neato) . This results in the Web server directing the request by executing the bootstrap script’s index.php.
2. The bootstrap script creates an Application instance and runs it.
3. The Application obtains detailed user request information from an application component named request.
4. The application determines the requested controller and action with the help of an application component named urlManager. For this example, the controller is user, which refers to the UserController class; and the action is login, whose actual meaning is determined by the controller.
5. The application creates an instance of the requested controller to further handle the user request. The controller determines that the action show refers to a method named actionLogin in the controller class. It then creates and executes filters (e.g. access controls, benchmarking) associated with this action. The action is executed if it is allowed by the filters.
6. The action reads a Login model.
7. The action renders a view named login with the Login model.
8. The view reads and displays the attributes of the Login model.
9. The view executes some widgets (If Needed).
10. The view rendering result is embedded in a layout.
11. The action completes the view rendering and displays the result to the user (The login page).

# Documentation

All the classes and methods are documented using PHPDoc (<http://www.phpdoc.org/>), which resembles standard JavaDoc or DotNet documentation output. Using this documentation, a technical reader can browse through all the classes and methods that are implemented. This documentation can be found inside (**neato/doc**) directory. Below is a screenshot of an actual PHPDoc for APIs. A reader can browse through classes using the left pan. He can also drill down and view methods/variables inside a class by clicking on “methods” link in a class’s PHPDoc.



# Testing

### Unit Testing

PHP Unit (<http://www.phpunit.de/manual/current/en/>) is used to provide the required unit test coverage around the web services. All the test cases are kept inside (**neato/protected/tests**) folder.

### Functional Testing

Selenium (<http://seleniumhq.org/>) is used to do some basic level of functional testing. We opted for Selenium as:

* Selenium gives ability to run the same test case across the browsers
* Decent support for Ajax calls
* Easy to maintain as lesser turn-around time to support frequently changing UIs

There is a functional test case that covers end to end flow for an end user. It is kept inside (**neato/protected/tests/functional**) folder.

# Web Service Test Console

The backend exposes a lot of web services that are used extensively by the robot and the smartphone apps. Use the web service test console (<http://neatostaging.rajatogo.com/wstest/>) to learn about the various APIs (their parameters and return values) as well as test them.

End.