OpenEyes - RBAC

Editors: G W Aylward
Version: 0.9:
Date issued: 20 February 2012

Target Audience

General Interest	✔
Healthcare managers	✔
Ophthalmologists	✔
Developers	✔

Amendment Record

Issue	Description	Author	Date
0.9	Draft	G W Aylward	20 Feb 2012

Table of Contents

Introduction

OpenEyes uses Role Based Access Control (RBAC) to handle permissions. With this approach, permission to view or modify patient records is based on a role, rather than on an individual. Users are not assigned permissions directly, but rather acquire them through their role (or roles). This means that management of individual user rights becomes a matter of assigning appropriate roles to the user, which simplifies common adjustments, such as adding a user, or changing a user's department.

OpenEyes is now written using the Yii framework, and can therefore make use of the RBAC system built into Yii, which supports the functionality required for OpenEyes as described in a separate document.[1]

This document describes the OpenEyes implementation using the built in RBAC system within the Yii framework.

Terminology

The implementation of RBAC in Yii, and the terminology used in the Yii documentation differ somewhat from the ANSI standard, last updated in 2004.[2] The following table describes the differences in the terminology.

Yii concept	ANSI standard	Notes
Operation	Permission	An atomic permission
Task	No equivalent	A task is a group of operations which allows similar operations to be conveniently associated
Role	Role	Defined as a set of tasks (and therefore operations)
User	User	A user of the system

All of the above concepts are treated as ‘authorization items’ in Yii, and each is uniquely identified by its name. This means that it is not possible for a task and a role (for example) to share the same name. A simple naming scheme in which the type of authorization item is the prefix in the name has therefore been adopted;

Yii concept	Prefix
Operation	Oprn
Task	Task
Role	Role
User	User

An authorization item may also be associated with a business rule, which is a piece of PHP code that will be executed when performing access checking with respect to the item. If the code returns true, the user be considered to have the permission represented by the item.

An authorization hierarchy is constructed as a partial-order graph, rather than a tree. Assignation of operations to tasks, and of tasks to roles is carried out by adding child nodes to the respective item. For example adding the item 'TaskClinical' as a child of the role 'RoleDoctor' assigns that task to the role of doctor.

Operations

Elements

Every OpenEyes element can potentially be added, viewed, edited, or deleted. This therefore results in a maximum of four operations for each element, and given that there is a large total number of elements, the number of operations could become impractically large. However, it is difficult to imagine a scenario where permissions would be different for adding and editing an element, so these functions can conveniently be combined into one editing operation. Hence for each elements there are three possible operations which are listed in the following table;

Operation	Description
OprnEdit<Element name>	Adding or editing an <Element name> element
OprnView<Element name>	Viewing an <Element name> element
OprnDelete<Element name>	Deleting and <Element name> element

Events

Each OpenEyes event consists of at least one element, though some may contain many more. For example the booking event contains an element with data about the operation, and another with information about the date booked. Events can be classified into a number of logical groups, some of which are listed in the following table;

Event group	Description
Demographic	Information identifying the patient, including name, date of birth, and address
Clinical	Clinical information relevant to any ophthalmic clinician, e.g. visual acuity
Diagnoses	Diagnostic information
Treatment	Information about treatment, including lasers and operations
Correspondence	Letters about the patient
Prescribing	Prescriptions for drugs and eyedrops
Admin	Administration information

Information that is highly sensitive (eg HIV status, child protection etc) within a group might also be subject to additional layers of permissions. This may be accomplished using the 'bizrule' functionality of the authitem table (see example below).

Deleting

There are very few circumstances in which data should legitimately be deleted from a clinical record. Indeed, there is clear advice from the GMC about the need to preserve clinical records in the form in which they were added. There is however a time period within which editing and deleting of information is allowable, and this is called the 'Allowable Editing Time ' (AET). This can be set at site level, and is recommended to be at least one hour, but no more than 24 hours. The aim is to allow a record to be edited as new information becomes available, but to disallow editing of records at a later date.The following principles are applied within OpenEyes to the deleting of information.

- An event or element may be deleted within the AET by the user who created the record.
- If the addition of an event resulted in the creation of a new episode to contain it, and no other events have been added by other users, then the episode should also be deleted
- If information is later found to be incorrect then it should be left, unless there is a risk that the incorrect information could compromise the care of the patient in the future. In this case the information can be deleted by an administrator user.

Tasks

A task is a group of logically related operations, usually on similar types of element or event. Using the naming convention, examples of tasks would be 'TaskCorrespondence', or 'TaskDiagnoses'. Tasks therefore cover both editing and viewing of a collection of events or elements. This is based on the principle that if a user is allowed to see one test result (for example), then it is likely that they will also be allowed to see any test result.

Roles

Tasks are then assigned to roles, so that for example, a trainee doctor might have the following tasks assigned;

- TaskBooking
- TaskCorrespondence
- TaskClinical
- TaskDiagnoses
- TaskPrescribing

A full list of roles is given in Appendix 1.

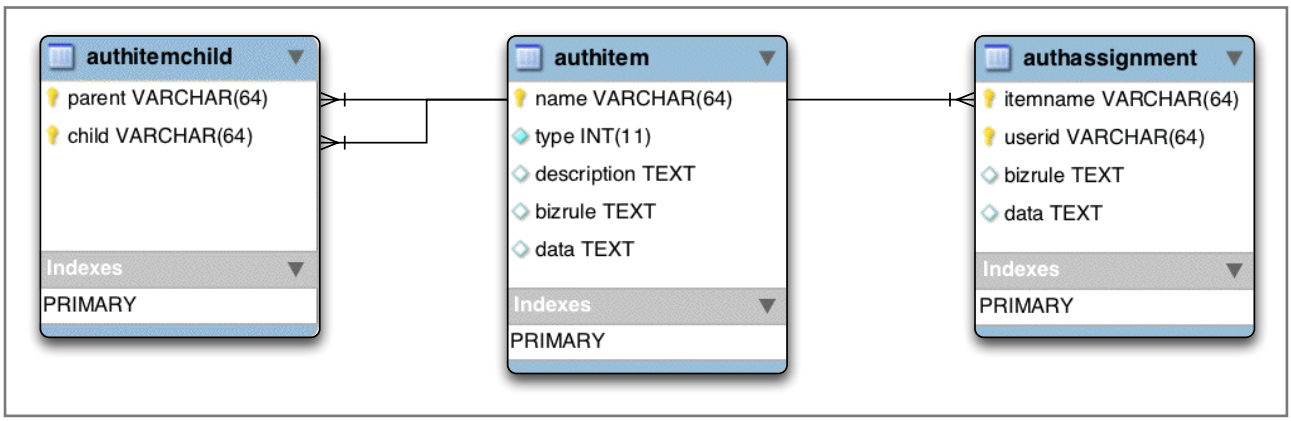
Configuration

Yii provides two types of authorization managers: CPhpAuthManager and CDbAuthManager. The former uses a PHP script file to store authorization data, while the latter stores authorization data in database. OpenEyes makes use of the latter type, and this needs to be set in the configuration file as follows;

```
'authManager'=>array(
        'class'=>'CDbAuthManager',
        'connectionID'=>'db',
    ),
```

The tables need to be created, and the SQL to create them is found at framework/web/auth/schema.sql. Items can be added to the RBAC either using Yii commands, or by directly editing the authentication tables in the database.

The database scheme is illustrated in the following entity diagram;



Adding Operations

The first step is to add a series of operations to the RBAC. The following commands add two operations for editing and viewing diagnoses;

```
$auth=Yii::app()->authManager;
$auth->createOperation('OprnEditDiagnoses', 'Operation to add or edit diagnoses');
$auth->createOperation('OprnViewDiagnoses', 'Operation to view diagnoses');
```

The effect of these commands is to add to entries to the authitem table as follows;

name	type	description	bizrule	data
OprnEditDiagnoses	0	Operation to add or edit diagnoses	NULL	N;
OprnViewDiagnoses	0	Operation to view diagnoses	NULL	N;

The type column defines what the authitem is (0: operation, 1: task, 2: role).

Creating a task

The next step is to create a task called 'Diagnoses' and add to it both the edit and view operations created in the above step.

```
$taskDiagnoses = $auth->createTask('TaskDiagnoses', 'Adds, views and edits diagnoses');
$taskDiagnoses->addChild('OprnEditDiagnoses');
```

```
$taskDiagnoses->addChild('OprnViewDiagnoses');
```

The effect of these commands is firstly to add a new entry to the authitem table, this time of type 1 (task);

name	type	description	bizrule	data
TaskDiagnoses	1	Adds, views and edits diagnoses	NULL	N;

Secondly, the command adds to entries to the authitemchild table indicating the parent-child relationship;

parent	child
TaskDiagnoses	OprnEditDiagnoses
TaskDiagnoses	OprnViewDiagnoses

Creating a role

The next step is to create a role to which the above task can be assigned. The following commands will create the role of doctor, and assign the above task to that role;

```
$roleDoctor = $auth->createRole('RoleDoctor', 'A medical doctor');
$roleDoctor->addChild('TaskDiagnoses');
```

The effect of these commands is firstly to add a new entry to the authitem table, this time of type 2 (role);

name	type	description	bizrule	data
RoleDoctor	2	A medical doctor	NULL	N;

Secondly, the command adds to entries to the authitemchild table indicating the parent-child relationship;

parent	child
RoleDoctor	TaskDiagnoses

Greater granularity can also be achieved by assigning individual operations to a role rather than a task.

Assigning a role

The final step is assign this role to a user, in this case the ‘demo’ user;

```
$auth->assign('RoleDoctor','demo');
```

The effect of this commands is to add a new entry to the authassignment table);

itemname	userid	bizrule	data
RoleDoctor	demo	NULL	N;

Using the RBAC

Once the configuration is complete, the RBAC can be used within Yii using constructs like the following;

```
// Check if the user can do a task
if (Yii::app()->user->checkAccess('TaskDiagnoses'))
{
    echo '<p>Diagnoses available for editing and viewing</p>';
}
// Check if the user can do an operation
if (Yii::app()->user->checkAccess('OprnViewDiagnoses'))
{
    echo '<p>Diagnoses available for viewing</p>';
}
```

It is also possible to put snippets of code within the bizrule field of the authitem table. For example, if we wanted the user not to be able to view sensitive diagnoses, a code snippet returning a bool could be written and put into the bizrule field of the OprnViewDiagnoses table such as this;

```
$isSensitive = function($diagnosis);            // Returns true if diagnosis is 'sensitive'
$isConsultant = function(Yii::app()->user->name)   // Returns true if user is consultant
return $isSensitive && $isConsultant;
```

SImilar code snippets can be used in order to check that users are members of a firm, specialty, or service. For example a rule which allows a user who has the role of 'Consultant' to edit letter phrases belonging to his/her firm only would have an entry called OprnEditLetterPhrases with a bizrule such as this;

```
return isMemberOfFirm(Yii::app()->user->name);
```

References

1. Aylward GW. OpenEyes Access Control.
2. Role Based Access Control. (ANSI ® INCITS 359-2004) American National Standards Institute, Inc. 2004.

Appendix 1

The following is a list of the likely roles required in a large installation of OpenEyes. Hierarchy is indicated by the arrows;

- Root
- Head → System → Admin → System → Admin → Local Admin
- Medical Director → Clinical Director → Service Director → Consultant → Doctor
- Head Nurse → Nursing → Sister → Staff Nurse → Nurse
- Head Optometrist → Senior Optometrist → Optometrist
- Head Orthoptist → Orthoptist
- Head Pharmacist → Pharmacist
- Electrocardiographer
- Phlebotomist
- Radiologist
- Ultrasonographer
- Head Technician → Technician
- Head Secretary → Secretary
- Head Clinic Clerk → Clinic Clerk
- Head Booking Clerk → Booking Clerk
- Chief Executive → Director of Operations → SDU Manager → Manager
- Director of Clinical Governance → Head of Clinical Governance
- Auditor
- Research Director → Head of Research → Researcher
- General Practitioner
- Hospital Consultant
- Community Optometrist
- Patient