HAIFA UNIVERSITY

Department of Computer Science

Quizzy Developer Guide

Table of Contents

Chapter 1

Background

How this project originated and a few notes.

During the search for a year project in our Computer Science BSc we found ourselves looking for a topic that would be beneficial to actual daily work and enable other to build upon it.

We ended up at the Department of Occupational Therapy at Haifa University. To our amazement we discovered a department that is a major part of this academic institute which is still using ancient and cumbersome ways of organizing and executing daily research related tasks.

The goal we were presented with was to provide the researchers with a simple, computerized system that would enable them to:

- Track patients
- Manage patient related "files"
- Build questionnaires and allow patients to digitally fill them in.

The reason we chose CakePHP as a framework to build upon:

We were looking for an existing CMS[1] that would provide us with enough functionality to implement the required components while allowing others to build upon our solution and to further extend it without having to dive deeply into our code.

Using an existing framework provides many benefits such as: built-in functionality and modules, a strong supporting community, easy to adapt and build upon for others and so on.

An important component used in this project is jQuery[2], jQuery is a "a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and

[1] Content management system

manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers. With a combination of versatility and extensibility, jQuery has changed the way that millions of people write JavaScript." (from the jQuery project site). jQuery enables many enhancements and new functionalities on the client-side.

Quizzy was developed and thoroughly tested on XAMPP[3] 1.8.0 (using PHP 5.4.4), running on Windows 7. We recommend using PHP 5.4 or above because of some changes to handling and access to named-arrays containing named-arrays (dereferencing).

Quizzy sources and all documentation can be found on the project page @ https://github.com/rsack/quizzy .

[2] http://jquery.com/

[3] http://www.apachefriends.org/en/xampp.html

Chapter 2

Important implementation topics

Some interesting and important implementation topics and discussion.

As with any project in general and specifically in programming, bringing the textual requirements into a real-life working system involve many road-blocks, development decision points and implementation decisions.

In this chapter we will discuss several of these topics and areas in which we had either faced problems or have some interesting and complex decision making.

The topics are listed by random order.

File uploads

A global variable wasn't used in case anyone decides to implement our solution over a different version of CakePHP and we would like to allow such implementations to involve the minimum amount of changes to the "core" of CakePHP. This would enable more novice programmers or developers to move faster with less potential failure points and changes to be made on the Cake part of things.

We did decide to separate between quiz related uploads (i.e. question images) and user related uploads (i.e. "patient files" that the research personal decide to store in the system). The decision to do so was twofold:

1. Privacy: separating between personal data which is more "delicate" than question images. Some people may decide apply different types of permission and access control to the different classifications.

2. Organizational: separation for organizing and managing purposes, some people may want to back up part of the data.

jQuery

We decided to use version 1.10 due to the fact it is the last official version to support IE6/7/8, while we do believe that newer browsers should be preferred mostly because

of better compliance with standards, we understand that most of the applicable users of our project might not be fully up-to-date or "tech savvy" and thus our choice.

Language support

An extremely important aspect we saw in the project was supporting multiple languages include RTL (Right-to-left) languages. We wanted to allow simple addition of new languages even for non-programmers while enabling a broad variety of potential languages.

CakePHP has built-in support for "Internationalization & Localization"[4] which evolve around the i18n concept[5], while this concept allows extremely versatile and robust translations and localizations it does require a certain level of programming understanding and adds a fair amount of complexity to the mix of creating a translation.

We decided to use a simple concept for the translation of the actual views and the i18n concept for the controller related messages. This decision was due to the fact we presumed someone wanting translated views might be a researcher or someone trying to use our implementation who might not be too tech-savvy, while someone editing controllers would require some level of computer programming expertise. The only drawback a translator might face in this setup is having translated views (and thus pages), but controller sourced messages will be displayed in English.

We strongly believe this is a fair compromise that allows users to quickly implement and translate pages without too much hassle.

"Out-of-the-box" translations are English and Hebrew. Accepted languages are defined in the .\app\Config\routes.php file. To add another language simply browse to the "Quizzy: adding support for the languages" block (line 36 onwards) and add another or clause in the 3 relevant lines.

Decision on what language should be displayed are performed in the AppController,php file (which all controllers inherit from). The selected language is stored in a cookie for convenience sake. The "_setLanguage" function will check whether a cookie is already defined, whether a language was specified in the request URL and whether the currently stored cookie fits the specified language and update the cookie accordingly. Next, the "beforeFilter"[6] function which is executed before every action in the controller will check for a view file existence in the app\View\[Controller]\[language] folder (i.e. for a Hebrew translation of

[4] http://book.cakephp.org/2.0/en/core-libraries/internationalization-and-localization.html

[5] http://en.wikipedia.org/wiki/I18n

[6] http://book.cakephp.org/2.0/en/controllers.html#Controller::beforeFilter

Admin\Something, it will check for a file named app\View\Admin\heb\Something.ctp). This way we make sure that not only a translation folder for the controller exists but that the exact requested view is available in the specific requested language. This manner would enable a user to translate only specific parts of the project while "falling back" to the English basic views for the rest. For example: translating all the views users might use while leaving the administration views in English.

To add the support for the language we had to create the MyHtmlHelper helper (which extends HtmlHelper[7]) and would add the new 'language' parameter (see routes.html mentioned previously) to URLs generated using it. If you decide to use the Router::url[8] function and will require support for language in the URLs it generates, please remember to implement a similar solution as we used for the Html url creation. We didn't require it because the Html url was sufficient and Router was only used for obtaining paths to several non-language specific paths (i.e. actions, images).

The actual language switching is done using links provided in the default layout file (default.ctp) that use the current arguments and update the language.

Supporting RTL for Hebrew required defining several HTML related attributes such as dir, we decided to add to the default layout a check to see if 'heb' was included in the viewPath (which would indicate that Hebrew was specified as the requested language **and** that a corresponding view was found, otherwise we might set RTL when an RTL language was specified but not actual view exists for it), if it was, we referred to a different CSS file which includes the direction[9] attribute set to RTL.

Localization of messages

For localizing message we decided to use the i18n standard and Cake's built-in translation capabilities because of our "opinion" on translating. To translate views you need not have a deep understanding of Cake/PHP/programming or any significant programming skills. You simply copy the page, create a new one with the required language and modify the text as text. On the other hand, to change messages you need some sort of programming knowledge and PHP understanding as you'll be modifying actual lines of code and thus we decided, "if you're going to change code, might as well use this great functionality".

Why not use the 18n support for everything you must be asking yourselves? For this exact same reason would be our answer ☺. Not only is changing and updating of i18n translations not the simplest of tasks, it would also require to re-do the process of updating for any change a developer would like to test out (including us while developing).

---

[7] http://book.cakephp.org/2.0/en/core-libraries/helpers/html.html

[8] http://book.cakephp.org/2.0/en/development/routing.html#Router::url

[9] http://www.w3schools.com/cssref/pr_text_direction.asp

Summing this up: if translating views -> copy and modify, if translating internal messages -> i18n.

Authentication

As we decided to use a known and proven framework, whenever the option of using an existing concept, component or functionality was presented, we did ☺. The authentication used in our project is simply based on the CakePHP Authentication component[10]. To adapt the component to our exact use case we did the following:

- All types of users are entered in the "users" table.
- A patient will have 2 entries: one in the "patient" table which would include all required profile information and another in the "users" table which will contain the user name (which is the patient's ID), password and role ("patient").
- An administrator or research assistant will only be listed in the "users" table with their appropriate role.
- Administrators can change a patient's password via the "Edit Patient" view.
- Administrators can add assistants who are basically administrators on single research.
- For security purposes we have included one administrator account (ID=1, username=admin password=123456). Upon installation the password can be modified (or done manually using some PHP code though we don't recommend doing so) using the default home page we adapted to our needs.

Password

Passwords are hashed using Cake's internal hash[11] function as detailed in the relevant methods. As we have mentioned the initial administrator password is defined as "123456" and should be changed using the function we added to the default home page.

For further information regarding setting initial passwords and more please refer to the "Quizzy Installation Guide" (available on the project page @ https://github.com/rsack/quizzy).

[10] http://book.cakephp.org/2.0/en/core-libraries/components/authentication.html

[11] http://book.cakephp.org/2.0/en/core-utility-libraries/security.html#Security::hash

Common functionality between controllers

There are several common functionalities between the Admin section and the Assistant section, for the simple reason that an assistant is basically an administrator on a specific research albeit some capabilities. Cake suggests using components for common functionality between controllers but they don't support models and thus we will end up with tiny components that provide little improvements over the duplication.

Another option is using admin prefixing but seeing as we are already using a sort of prefixing for the matter of the multiple language capabilities we decided to avoid the additional complexity and cumbersomeness of adding another level of partial prefixing and stick with some duplication between the controllers.

The options of "Import patient" and "Manage patients" do have a fair amount of functionality in common but we decided to leave the import option for a quick and simple import task. If an administrator would just like to import a patient quickly, they might as well use the "Import patient". While using the "Manage patients" is slightly more delicate and more places for errors exist, additionally it has more capabilities but would require being a bit more careful when using it as it can also remove patients and not only add.

Mobile support

In this day and age the matter of mobile support and access from mobile devices is gaining more and more momentum and importance. It's hard to say which functionality was more important in the project, that of mobile support or that of multiple language support.

Our mobile device support revolves around Cake's RequestHandler mobile device detection[12] and a custom layout (mobile.ctp). The layout has been adapted to support the required mobile meta tags with the appropriate definitions.

One of the main requirements we were given for the project was to enable patients and researchers to perform actions on mobile devices, specifically tablets. Allowing researchers to sit with patients and fill out quizzes, enabling patients to perform quizzes at their leisure at home using a computer or mobile device, and adapting to the modern era of mobile device access..

An interesting note regarding debugging mobile access, there are two simple ways to debug and develop mobile sites from the comfort of your computer:

1. Changing your browser's user-agent, either manually or using a plugin/extension.

[12] http://book.cakephp.org/2.0/en/core-libraries/components/request-handling.html#RequestHandlerComponent::isMobile

2. Moving the 2 lines that define the mobile rendering out of the "if" clause in the afterFilter method in AppController.php

Chapter 3

Classes

The naming convention CakePHP imposes makes most classes fairly self-explanatory. We will detail each controller, model and view and detail their included functionality if it isn't self-explanatory by the name of the relevant function.

Controllers

AdminController

Intention: handling all administrative tasks (highest level of access).

Included Methods:

- addAssistant - add or edit assistant.
- addPatient - add or edit patient.
- addPatientFile - adding patient file (from the "Edit Patient" page).
- addQuiz - quiz builder.
- addResearch - add new research.
- deleteAssistant - delete an assistant, and all relevant data (research associations).
- deleteFile - delete a patient file, the actual file and DB entry.
- deletePatient - delete a patient, and all relevant data (research associations, patient files, quiz allocation and the actual patient entry).
- deleteQuiz - delete a quiz, and all relevant data (patient-quiz allocations for this quiz, questions from the quiz and the actual quiz entry).
- importPatient - import existing patients from the system (will not display patients already associated with the current research).
- index - index page, will handle research choice and display the relevant data.
- isAuthorized - uses default isAuthorized method for validating user role (administrator).

- manageAssistants - manage assistant allocation page, will handle the submit of changes as well.

- managePatients - manage patient allocation page, will handle the submit of changes as well.

- patientQuiz – manage a specific patient quiz allocation page, will handle the submit of changes as well.

- [private] handleUpload - handle all uploads (both patient files and question images), validate against a list of defined extensions, if a file with an identical name exists it will automatically rename it.

- [private] savePatientQ - handles saving a patient's quiz allocation (including removal of no longer required quizzes and addition of new ones).

- [private] saveQuestion - handles saving one question with all its fields (text, type, data and image).

- [private] saveQuiz - handles saving the actual quiz entry (includes only the title).

- [private] saveResearchAssistant - handles saving a research's assistant allocation (including removal of assistants and addition of new ones).

- [private] saveResearchPatients - handles saving a research's patient allocation (including removal of patients and addition of new ones).

AnalysisController

Intention: handling all data analysis and exporting tasks.

Included Methods:

- index - index page, will handle the submit of filter requests (to display data) as well.

- isAuthorized - validates user role, either administrator or assistant.

AppController

Intention: built-in basic controller that all others extend.

Included Methods:

- afterFilter - built-in Cake method[13], the last controller method executed. We use it to change the rendering layout to the mobile one only after all other

[13] http://book.cakephp.org/2.0/en/controllers.html#Controller::afterFilter

methods have been run, thus providing full functionality to the mobile version while only changing the actual layout used.

- beforeFilter - built-in Cake method[14], executed before any controller action. We use it to update the Auth messages to translations, to set the requested language (see "_setLanguage"), to decide on the appropriate view (language defined AND view exists (see "Language support") and to detect mobile device access.

- isAuthorized - default isAuthorized method for validating user role, administrators have access everywhere, if a different type of validation is required, the method will be overloaded.

- redirect - overloading/extending the redirect function to support languages (and calling the original function at the end).

- [private] _setLanguage - handles all language related configurations (cookies and session).

AssistantController

Intention: handling all research assistant related tasks.

Included Methods:

- index - index page, will handle research choice and display the relevant data.

- isAuthorized - validates user role (assistant).

- [private] isAssistantOnResearch - validates whether or not an assistant is defined as an assistant on a given research.

- **Note**: the AssistantController includes all the methods (and naturally views) from the AdminController except the research and assistant related ones. We didn't detail them again to avoid being redundant.

PagesController

Intention: built-in controller for handling certain pages.

Included Methods:

- beforeFilter - built-in function as described above, used to allow viewing of the initial home page we setup.

[14] http://book.cakephp.org/2.0/en/controllers.html#Controller::beforeFilter

- display - built-in function but updated with several pieces of data for our initial home page (admin password changing).

PatientController

Intention: handling all patient tasks, basically the patient's window to the system.

Included Methods:

- index - index page, displays the relevant data and links to pending quizzes.
- isAuthorized - validates user role (patient).

QuizController

Intention: handling quiz taking tasks.

Included Methods:

- index - index page, will handle displaying a defined quiz for a patient (and naturally validate the choice/availability) and handle saving of answers on quiz submit.
- isAuthorized - validates user role (patient).

UsersController

Intention: handling all authentication tasks.

Included Methods:

- beforeFilter - built-in function as described above, used to allow viewing of login/logout pages without authentication.
- login - handles the login page and actual login process, if successful it will redirect the user based upon their defined role.
- logout - handles the logout page and the actual logout action, destroying the session and redirecting back to the login page.

Models

Answers

Intention: reflecting the answers table, each answer filled in by a user on a specific question.

Included Methods:

- loadAllData
- loadListByQuestion

- loadListByQuiz
- loadListByUser
- loadMultiQA
- loadPatRes
- loadQuizRes

Assistant

Intention: reflecting the assistant table, each assistant will have a unique entry.

Included Methods:

- loadAllData

Patient

Intention: reflecting the patient table, each patient will have a unique entry.

Included Methods:

- loadAllData
- loadPatient
- loadPatientList
- loadPatientsAsList

PatientFiles

Intention: reflecting the patientfiles table, each uploaded patient file will have an entry describing and linking it.

Included Methods:

- loadAllData
- loadFileID
- loadPatientFiles

PatientQuiz

Intention: reflecting the patientquiz table, each quiz a patient associated with will have an entry

Included Methods:

- getPatientQuizID
- hasPatientTaken
- loadAllData
- loadPatientQuizzes

Questions

Intention: reflecting the questions table, each question will have a unique entry.

Included Methods:

- loadAllData
- loadSpecific

Quiz

Intention: reflecting the quiz table, each quiz will have a unique entry.

Included Methods:

- loadAllData
- loadQuiz
- loadQuizList
- loadQuizListFiltered
- loadQuizResearchList
- loadResearchQuizzes

Research

Intention: reflecting the research table, each research will have a unique entry.

Included Methods:

- loadAllData
- loadResearch

ResearchAssistant

Intention: reflecting the researchassistant table, each association of an assistant to a specific research will have an entry.

Included Methods:

- loadAllData
- loadAssistantResearches
- loadResearcheAssistants

ResearchPatients

Intention: reflecting the researchpatients table, each association of a patient to a specific research will have an entry.

Included Methods:

- isPatientOfResearch
- loadAllData
- loadResearchPatients

Users

Intention: reflecting the users table, handling all user authentication and role data.

Included Methods:

- beforeSave - hash (encrypt) the user password before saving, we want to save it encrypted and not in plain text.

Chapter

4

Database Tables

Table "answers"
Intention: contains all question answers. Fields: a unique ID, the date of the answer, patient ID that answered it, the quiz ID, the question ID, the type of the question and naturally the answer to it.

Table "assistant"
Intention: contains all research assistants. Fields: a unique assistant ID, and name.

Notes:

- The assistant password information is stored in the "users" table (see below).

Table "patient"
Intention: contains all patients and their profile information. Fields: a unique patient ID, first name, last name, age, gender, marital status address and phone number.

Notes:

- The patient password information is stored in the "users" table (see below).

Table "patientfiles"
Intention: contains information regarding all uploaded patient files. Fields: a unique ID, patient ID, date & time of addition, the file path and a description.

Table "patientquiz"
Intention: contains all patient quiz allocation. Fields: a unique ID, patient ID, quiz ID and a flag whether or not the quiz has been taken.

Table "questions"
Intention: contains all questions and their relevant information. Fields: unique question ID, quiz ID, the question text, type of the question, image if relevant and the data which is the actual information we use to construct the question.

Table "quiz"
Intention: contains all quiz information. Fields: unique quiz ID, title and the research with which it is associated.

Table "research"

Intention: contains all research information. Fields: unique research ID and the name of the research.

Table "researchassistants"

Intention: contains all assistant -> research allocations. Fields: unique research ID, unique assistant ID.

Table "researchpatients"

Intention: contains all patient -> research allocations. Fields: unique research ID, unique patient ID.

Table "users"

Intention: holds all system users (administrators, assistants and patients). Fields: unique ID, username, password, role, date/time of creation and date/time of modification.

Notes:

- Passwords are hashed by CakePHP[15].
- The "role" column is an enum (enumerated data type) to only allow the relevant roles.
- For security purposes we have included one administrator account (ID=1, username=admin password=123456). Upon installation the password can be modified (or done manually using some PHP code though we don't recommend doing so).

[15] http://book.cakephp.org/2.0/en/core-libraries/components/authentication.html