Fall 12-13-2010

Web-Based IDE to Create Model and Controller Components for MVC-based Web Applications on CakePHP

Sugiharto Widjaja
San Jose State University

Recommended Citation
Widjaja, Sugiharto, "Web-Based IDE to Create Model and Controller Components for MVC-based Web Applications on CakePHP" (2010). *Master's Projects*. 1.
DOI: https://doi.org/10.31979/etd.mq7g-jkjc
https://scholarworks.sjsu.edu/etd_projects/1

Web-based IDE to create Model and Controller Components for MVC-based Web Applications on CakePHP

A Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Computer Science

by Sugiharto Widjaja
Dec 2010

Abstract

Web-based IDE to create Model and Controller Components for MVC-based Web Applications on CakePHP

by Sugiharto Widjaja

A Web-based IDE that allows users to easily manage Model and Controller components of a CakePHP web application was developed. With this IDE, users are able to manage the model and controller components without having to write very much PHP code. They are able to create new model components without having to worry about creating the database tables for the models as the IDE creates them automatically. The IDE offers a simple interface for users to edit the schema of their models. Users can add or delete data from their models without dealing with the SQL insert/delete statements. The IDE also supports adding new controllers to their projects and adding models to be used by the controllers. Constructing find methods (equivalent to SQL statements) is easy because the IDE provides users with a simple interface. Users do not need to deal with the complex SQL select queries. Users can associate the models by using a simple form. The IDE also provides a code-completion feature for users when they edit the controller files. If users want to run their applications on other machines, then they can simply export their applications to a SQL file that they can run on other machines. Finally, the IDE also provides small interactive tutorials on each feature when the users use those features.

Acknowledgements

I would like to thank my advisor, Dr. Chris Pollett for his constant encouragement, technical advice, and support during this project. I would also like to thank my committee members Dr. Sami Khuri and Dr. Mark Stamp for taking time to offer me great suggestions and supports. Special thanks also go to my classmates at San Jose State University (in particular Swathi, Deepti ,and Tejasvi) for their constant feedback and input during this project.

Thanks also goes to John Resig who has built such a great JavaScript framework, jQuery, which has made my JavaScript codings very enjoyable.

Finally, my love and gratitude goes to my fiancée Anne Hansen for her never ending support and encouragement.

Contents

List of Figures

1. Introduction

With the emergence of cloud computing, more services are now available online. For example, Google provides Google Docs service that allows users to create and edit their document, presentation, and spreadsheet files online. Online services offer several advantages over traditional desktop applications. Users do not need to install the software on their computers. Instead, they can just use their Web browser to access the online services. Users do not have to worry about updating the software as the service provider will usually perform the update task. Users can use the service anywhere as long as they have access to computers and an Internet connection. Online services will usually back up all of the users' files. Therefore, users will always have access to their files as long as they have access to computers and Internet connection.

There are many developers who write or edit their source codes by using a software application called IDE (Integrated Development Environment). Most IDEs are usually consists of a source code editor and a compiler or interpreter [1]. In the area of Web development, there are several free online services that offer an IDE sub-component. By using these IDE services, users are able to edit their files (PHP, HTML, etc.) online. Some examples of such IDE services are Mozilla Skywriter by Mozilla, PHPanywhere.net, cPanel, and Kodingen. However, most of the online IDE services have one crucial weakness: they are not user-friendly enough. Users have to manually edit their files, set up the database and database tables. All of these tasks can be troublesome for less advanced users.

The goal of our project is to develop a user-friendly CakePHP IDE that users can use to develop their Web applications. We designed our IDE to be compatible with CakePHP framework. CakePHP is a PHP framework that is based on MVC (Model, View, and Controller) design pattern [2]. While MVC was initially a design pattern mainly used by traditional desktop applications, there are many Web frameworks that are based on MVC design pattern currently. CakePHP is one of the MVC-based Web frameworks, and it is heavily influenced by Ruby on Rails. Every Web application in CakePHP is broken into three components: model, view, and controller [2]. The model component will represent the data used in the Web applications. The controller component will manage the logic part of the Web application. The view component will generate specific output in formats such as HTML, XML, or JSON. While such an approach will allow users to have more structured applications, it will also present challenges since users have to make sure all the components are set up properly. Our CakePHP IDE will attempt to mitigate such challenges by automating the Web applications creation processes as much as possible. CakePHP also uses the “coding by convention” paradigm [3]. For example, CakePHP will automatically associate the model Book with the database table “books”. By adhering to CakePHP naming conventions, users will not need to create configuration files. Therefore, our CakePHP IDE will always adhere to the CakePHP naming conventions.

For this project, I am responsible for implementing the model and controller components of our CakePHP IDE. I am also responsible for implementing some common features of our IDE such as: Registration system, login system, and confirmation dialog boxes. Swathi Vegesna, another Computer Science graduate student, is responsible for implementing the view components of our CakePHP IDE.

We organized our reports into ten chapters. Chapter one discusses about the major limitation of the current Web-based IDE services and how our CakePHP IDE can overcome this limitation. We discuss the technologies that we used in creating our CakePHP IDE in chapter two. Chapters three and four cover the design and architecture of our IDE. Chapter five discusses about the implementations of the major functionalities of our CakePHP IDE. We discuss the issues that we encountered and how we solved them in chapter six. We present the results of our CakePHP IDE's performance testing in chapter seven. Chapter eight discusses about the results of the usability testings of our IDE. Finally, we present the conclusion of our project in chapter nine.

2. Technologies Used

There are several technologies that we use in this project:

CakePHP Framework

CakePHP is an MVC-based PHP framework that is heavily influenced by Ruby on Rails. CakePHP is available for free from http://cakephp.org/. Our IDE can create new CakePHP projects along with the required model, view, and controller components.

JavaScript

We used jQuery 1.4.3 as our main JavaScript framework. jQuery is a JavaScript framework created by John Resig. The main strength of jQuery is the retrieval of DOM elements from HTML pages. We also used jQuery UI 1.8.5 to take advantage of its drag-and-drop, tabs, and modal dialog box features. jQuery UI also provides themes and styling for the modal dialog box and tabs. jQuery is available for free from http://jquery.com/. jQuery UI is also available for free from http://jqueryui.com/.

We also used a jQuery plugin called jQuery contextMenu for the right-click options of the models, views, and controllers on the left panel of the IDE. Right-clicking on any of these elements will bring up a context menu that has all the possible operations for those elements.

Since our GUI will allow users to manually edit the model, view, and controller files; our IDE also uses a rich-text editor framework called CKEditor (version 3.4). CKEditor allows developers to easily create personal plugins to be used with it. For the purpose of this project, we developed two plugins: A plugin to save the edited files and a plugin for auto-suggest on controller files editing.

We used JSON (JavaScript Object Notation) heavily for data transfer between the client-side and server-side. We used Douglas Crockford's JSON JavaScript script to parse JSON strings and to convert JSON objects to strings because it is safer than using JavaScript eval() function [5].

We also had to use another jQuery plugin called liveQuery because jQuery live() function does not work for events that do not bubble up [4]. In our project, jQuery failed to bind the onChange event to the callback function for newly created select element in Internet Explorer browser. We are able to bind the onChange event to the callback function for newly created select element by using liveQuery.

Firebug and Safari / Chrome Developer Tool

Firebug is an add-on for Mozilla Firefox Web browser. Firebug will let users debug the HTML, CSS, and JavaScript of HTML pages. We used this tool a lot when we coded our GUI. This tool also provides a tool to debug the JavaScript codes. Firebug is also very helpful in debugging the Ajax calls. Both Safari and Chrome browsers have a built-in developer tool that has many features to debug the HTML, CSS, and JavaScript of HTML pages. We use this developer tool to help us tracing the Ajax calls and debugging our JavaScript codes.

Parallels Desktop 5 For Mac

Since we developed our IDE in a Mac environment, we needed to find a way to test our IDE on Internet Explorer browsers. We used Parallels Desktop 5 for Mac and we downloaded the VPC (Virtual PC) hard disk images from Microsoft site. After importing the VPC hard disk images to Parallel virtual machine file, we were able to test our IDE on IE browsers.

3. Design

3.1 Goal and Requirements

The goal of our CakePHP IDE is to allow users to create sophisticated CakePHP Web applications easily. Users should not need to worry about the files setup and database requirements as the IDE takes care of them. Beside the automatic setup of files and database, the IDE should also support these critical features:

1. Model Schema Editing

Without using our IDE, users need to issue an SQL alter query to alter the model database table. This can be challenging for users who are not well-versed in SQL. The IDE provides a simple interface to let users edit the model schema easily. Our IDE automatically generates the required SQL alter query to alter the model database table and executes that SQL query.

2. Adding and Deleting Model Data

If users want to add new data to the model, they will need to execute SQL insert queries on the database. If they want to delete data from the model, they will need to execute SQL delete queries on the database. Our IDE lets users to add new data or delete existing data from a model without having to write SQL insert/delete queries.

3. Constructing the Model find() Method

If users want to retrieve data from a model, they can use CakePHP model find() method instead of performing select SQL queries on the database. Writing the PHP code to perform find() method can be tricky because find() parameter can be very complex [6]. Our IDE provides an interface to allow users to construct the parameter of the find() method. After our IDE has constructed the parameter, it will also add that function to the model PHP file.

4. Associating Models

One of the CakePHP features is to allow the linking of two or more models. These linkings are handled through associations. CakePHP currently supports four type of associations:

- One-to-one

- One-to-many
- Many-to-one
- Many-to-many

Associating two models (model 1 to model 2) requires users to create a foreign key that will link the database tables of the two models. This can be challenging for users who are not well-versed in SQL foreign key concept. Our IDE lets users to create associations between models without having to worry about the database. Our IDE currently supports one-to-one, one-to-many, and many-to-one associations.

5. Creating a New Model or Controller for a Project

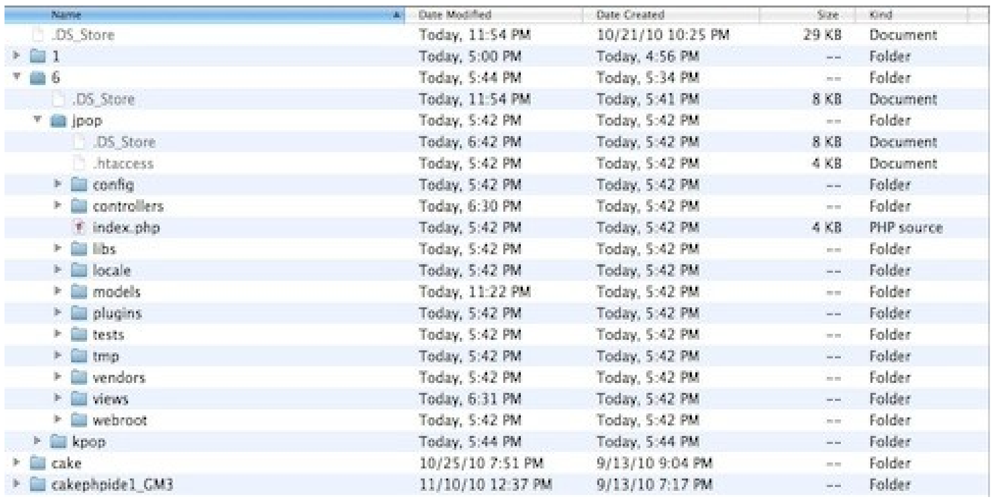
When users develop their Web application projects with our IDE, they will want to add new models or controllers for those projects. Our IDE provides users with simple interface to add the new models or controllers to their project. When users create a new model or controller for their project, our IDE automatically creates a basic model or controller file and associate it with the project.

3.2 Users Registration and Login Pages

Our CakePHP IDE has a registration page that lets users to register to use our CakePHP IDE. After the users have registered, they will receive an email that asks them to confirm their account. When the users confirm their account, they will be able to login to our IDE and start to create CakePHP Web applications.

3.3 Directory Structure

We put the CakePHP core library, our CakePHP IDE, and other user-created projects in the web server www directory. Figure 1 illustrates the typical directory structure of our CakePHP IDE. The cake directory is the CakePHP core library and the the cakephpide_1_GM3 directory contains our IDE source codes. The rest of the directories (1 and 6) are users' directories. Each user directory will have several directories for the user's projects. User's directory will have the same name with the user's user id. For example, directory 6 is the directory for user with id 6. We see that this user has created two projects: jpop and kpop. While it is possible to put these folders in any directory, we suggest putting them in the web server www directory so that users do not need to edit the web server virtual host configurations.



Name	Date Modified	Date Created	Size	Kind
.DS_Store	Today, 11:54 PM	10/21/10 10:25 PM	29 KB	Document
1	Today, 5:00 PM	Today, 4:56 PM	--	Folder
6	Today, 5:44 PM	Today, 5:34 PM	--	Folder
.DS_Store	Today, 11:54 PM	Today, 5:41 PM	8 KB	Document
jpop	Today, 5:42 PM	Today, 5:42 PM	--	Folder
.DS_Store	Today, 6:42 PM	Today, 5:42 PM	8 KB	Document
.htaccess	Today, 5:42 PM	Today, 5:42 PM	4 KB	Document
config	Today, 5:42 PM	Today, 5:42 PM	--	Folder
controllers	Today, 6:30 PM	Today, 5:42 PM	--	Folder
index.php	Today, 5:42 PM	Today, 5:42 PM	4 KB	PHP source
libs	Today, 5:42 PM	Today, 5:42 PM	--	Folder
locale	Today, 5:42 PM	Today, 5:42 PM	--	Folder
models	Today, 11:22 PM	Today, 5:42 PM	--	Folder
plugins	Today, 5:42 PM	Today, 5:42 PM	--	Folder
tests	Today, 5:42 PM	Today, 5:42 PM	--	Folder
tmp	Today, 5:42 PM	Today, 5:42 PM	--	Folder
vendors	Today, 5:42 PM	Today, 5:42 PM	--	Folder
views	Today, 6:31 PM	Today, 5:42 PM	--	Folder
webroot	Today, 5:42 PM	Today, 5:42 PM	--	Folder
kpop	Today, 5:44 PM	Today, 5:44 PM	--	Folder
cake	10/25/10 7:51 PM	9/13/10 9:04 PM	--	Folder
cakephpide1_GM3	11/10/10 12:37 PM	9/13/10 7:17 PM	--	Folder

Figure 1: CakePHP IDE Directory Structure

3.3 Naming Conventions

When users create a new project, they are only required to enter the project name and their database system credentials (db server name, db username, and db password). In order to automate most of the backend processes, we use several naming conventions:

- Project Name: Provided by Users
- Model Class Name: camelize(Project Name)
- Model filename: underscore(Project Name).php
- Controller Class Name: camelize(pluralize(Project Name))
- Controller Filename: underscore(pluralize(Project Name))_controller.php
- View Folder: pluralize(Project Name)

Camelize is a CakePHP utility function that converts a string to its camelized form. For example, camelize('my_bookstore') will return string 'MyBookstore'. Underscore is a CakePHP utility function that converts a camelized string to its lower-cased and underscored string. For example, underscore('MyBookstore') will return string 'my_bookstore'. Pluralize is another CakePHP utility function that will convert a string to its plural form. For example, pluralize('book') will return string 'books'.

If a user creates a project called 'Bookstore', then we will have:

- Model Class Name: Bookstore
- Model filename: bookstore.php

- Controller Class Name: Bookstores
- Controller Filename: bookstores_controller.php
- View Folder: Bookstore

4. Architecture

4.1 Database

Our IDE will use one database called ides. Inside the ides database, we will have six database tables:

users(<u>id</u>, username, password, confirm_hash, has_confirmed)

The users table has the login information of users who have registered to our CakePHP IDE. When users register to our IDE, it will automatically create a confirmation code for the users. The confirmation code is calculated by performing md5 hash on the concatenation of users' email address, current time, and the pre-defined salt.

ides (<u>id</u>, project_name, project_path, project_db_config, user_id)

The ides table will have the information about the projects created by users. The project_name field is the name of the project created by the users. The project_path field is the directory of the projects. The project_db_config field is the DB configuration of the projects. Each project will require this DB configuration to function properly. The DB configuration will have the database driver type (for example: MySQL, MySQLi, Postgres), database server host, database name, database login, and password. Our IDE will use the project name as the default value for the database name. For example, project "Bookstore" will have database name "Bookstore". The user_id field is the user id of the user who creates this project.

model_components (<u>id</u>, model_name, model_filename, ide_id)

The model_components table will have the information about the models for every project. The model_name will be the name of the model. The model_filename is the path of the php filename that has the class definition of the model. Ide_id is the foreign key that points to id in ides table.

controller_components (<u>id</u>, controller_name, controller_filename, ide_id)

The controller_components table will have information about the controllers for every projects. The controller_name will be the name of the controller. The controller_filename is the path of the php filename that has the class definition of the controller. Ide_id is the foreign key that points to id in ides table.

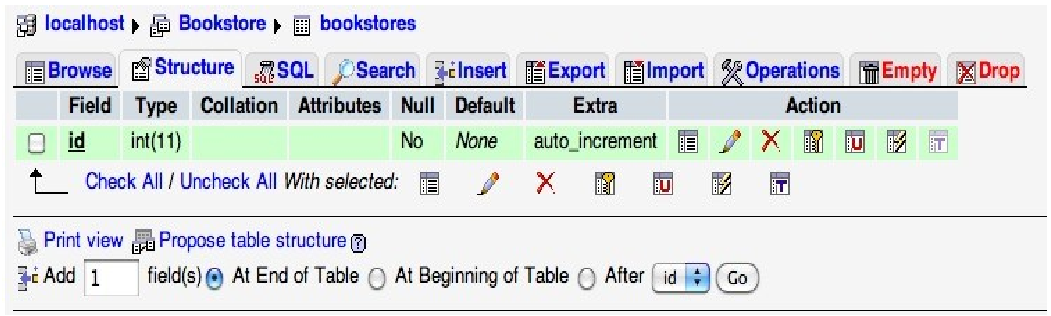
view_components (<u>id</u>, view_filename, ide_id)

The view_components table will have the view filenames that are being used in every project. The view_filename is the path of the .ctp (template) file that is being used in a project. Ide_id is the foreign key that points to id in ides table.

models_associations (<u>model1_id</u>, <u>model2_id</u>, association_type)

The models_associations table will store the information of the associations that are created between models. Model1_id is the id of the first model, while the model2_id is the id of the second model. The association type will be an enum with values of 'hasOne,' 'hasMany,' and 'belongsTo'.

When users create a new project, the IDE will create a new database for that particular project. By default, the IDE will automatically create one database table. The name of the database table will be underscore(pluralize(project name)). This default database table will only have one field ('id'). Users will be able to add more fields to this table by using the edit schema feature of the IDE.



	Field	Type	Collation	Attributes	Null	Default	Extra	Action
☐	id	int(11)			No	None	auto_increment	

Check All / Uncheck All *With selected:*

Print view Propose table structure

Add 1 field(s) [✓ At End of Table] At Beginning of Table After id Go

Figure 2: The default table and attribute created for project Bookstore

Whenever users create a new model for their project, the IDE will automatically create a new database table for that model. The database table will be created in the project database. For example, if a user creates a new model Book for project Bookstore, the IDE will create a new database table 'books' in the 'Bookstore' database.

4.2 Critical PHP/JavaScript Scripts

In order for our CakePHP IDE to work, it will need the interactions between the client-side (JavaScript) and server-side (PHP). We will discuss several PHP and JavaScript scripts that are critical for our CakePHP IDE.

4.2.1 PHP Scripts

controllers/ides_controller.php

This is the controller script for our CakePHP IDE. Most of the time, this script will receive requests from the JavaScript scripts via Ajax call. After processing the requests, the controller script will return the result back to the JavaScript scripts in either text or JSON format.

controllers/filedls_controller.php

This controller script will process the requests from clients to download the result of project export in SQL file.

models/ide.php

This is the model script for our CakePHP IDE. The script will mostly handle the requests from the IDE controller script for every database-related operations. The model script is also responsible for creating the models, views, or controller components from other projects.

vendors/cakephpmysqlconverter.php

This PHP script is responsible for converting the CakePHP data types to MySQL data types. The conversions will be based on these mappings:

CakePHP	**MySQL**
string	VARCHAR
text	TEXT
integer	INT
float	FLOAT
Datetime	DATETIME
timestamp	TIMESTAMP
date	DATE
Binary	BLOB
Boolean	TINYINT(1)

vendors/cakephpcontrollermodelusesadder.php

This PHP script is responsible for adding a model to a controller. It performs this by adding or editing the $uses variable in the controller php file. For example, if a user adds model 'Book' to controller 'Bookstore', the script will attempt to add 'Book' to the $uses variable. If the attempt is successful, the controller $uses variable will now contain 'Book'.

vendors/cakephpcontrollermodelfindadder.php

This PHP script is responsible for adding a new find method to a model. It performs this by adding a new function in the model php file.

vendors/cakephpcontrollermodelassocadder.php

This PHP script is responsible for adding an association between two models. It performs this by adding to either the $hasMany, $hasOne, or belongsTo variable. For example: if a user creates a hasMany association between model “Bookstore” and “Author”, the script will attempt to add model “Author” to $hasMany variable in “Bookstore” model php file.

4.2.2 JavaScript Scripts

webroot/js/jquery-1.4.3.min.js

This is the main js file for jQuery.

webroot/js/jquery-ui.js

This is the main js file for jQuery UI.

webroot/js/livequery.js

Livequery is a plugin for jQuery. We used Livequery because the jQuery live() operation does not work well with onchange events on IE Web browsers. By using livequery, we are able to bind the onchange events successfully on all types of Web browsers.

webroot/js/json2.js

Json2.js is a script written by Douglas Crockford. This script serves two important functions: to convert a JSON string to JSON object and to convert a JSON object to its string representation. We used this script because it provides a safer way to parse a JSON string than using JavaScript built-in eval() function.

webroot/js/treeview.js

Treeview.js is a plugin for jQuery. The purpose of this script is to display the projects and their MVC components in a tree hierarchy.

webroot/js/jquery.contextMenu.js

ContextMenu is a plugin for jQuery. The purpose of this script is to enable right-click interactions for our IDE.

webroot/js/ckeditor

This folder contains the required JavaScript files for CKEditor. The CKEditor is a Web-based rich text editor that our IDE will use for manual editing of any model, controller, or view files. We have also created a custom plugin for CKEditor to handle the auto-completion for controller files editing.

webroot/js/project.js

The project.js script contains implementations of some project-related functions. We implemented the functions to create new projects and to export existing projects to SQL files. We also implemented functions to read and save the php and ctp (CakePHP template) files.

webroot/js/model.js

The model.js scripts contains implementations of model-related functions. Some important functions that we implemented here are: creating and saving model schema, creating new models, adding or deleting data from models, and associating models.

webroot/js/controller.js

The controller.js scripts contains implementations of two controller-related functions. The functions that we implemented here are: creating new controller and adding models to controller.

webroot/js/init.js

This script is responsible for the initialization of the GUI. The initializations that this script performs are

- Instantiate the CKEditor editor that users will use to manually edit the model, controller, or view files.
- Initialize all the dialog boxes that will be used by the GUI.
- Bind all the buttons click events with their corresponding callback functions.
- Bind all the right-click events with their corresponding callback functions.

5. Implementation

5.1 General Implementation Flow

Our CakePHP IDE uses Ajax technology heavily to prevent excessive numbers of page reloads. For the most part, our JavaScript code will issue Ajax calls to particular controller actions. The controller will process the Ajax call and return the results to the JavaScript in either text or JSON format. The controller will also access any required models to retrieve data from the models or to perform certain functionalities.

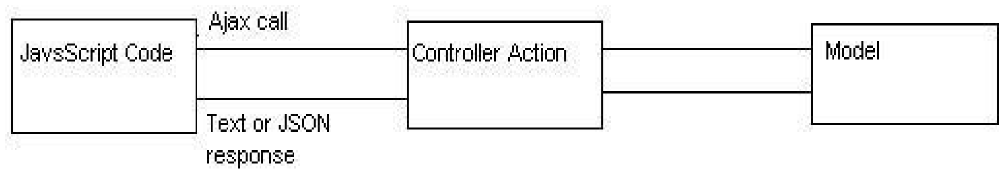


Figure 3: The Application Flow

5.2 Critical Back-end Features

Our CakePHP IDE requires many back-end features to ensure the success of the users' requests from the front-end. We will discuss the following critical features: the MVC components instantiation, CakePHP – MySQL data types conversion, adding model to controller uses, constructing new find methods for models, and associating different models.

5.2.1. MVC Components Instantiation

Our CakePHP IDE requires many instantiations of the model, view, and controller components from different projects. The successful instantiations of the components are critical because the IDE will need the component objects to obtain the attributes from the components (such as obtaining schema from model) or to use the methods from the components (such as saving model entries to database table). The IDE model is responsible for instantiating the MVC components. The following PHP code will show how the MVC components from different projects are instantiated.

```
/**
  component_type: Model, View, or Controller
  component_path: the path of the component php file
  component_name: the name of the component
  db_config_name: the name of the database config used by the project that has this component
  Note that App::import will only include the required file to instantiate the component.
/*
App::import($component_type, $component_name ,array('file' => $component_path));
return new $component_name(false,null,$db_config_name);
```

Interestingly, we discovered a bug in CakePHP that will prevent the successful instantiation of a model component if that model is associated with other models. We fixed this bug by patching the core CakePHP model.php script. The following code will show the patch.

```
/**
  Line 656 of CakePHP model.php script
  We need to pass the data source for the model so that CakePHP can get the correct database config to
construct the associated models.
*/
$model = array('class' => $className, 'alias' => $assoc, 'ds' => $this->useDbConfig);
```

5.2.2 CakePHP – MySQL Data Types Conversion

We did not use MySQL data types in our front-end IDE GUI as this will confuse the users who are not well-versed in SQL data types. Instead, we used the data types that are supported by CakePHP. However, the IDE will still need to convert the CakePHP data types to MySQL data types in some cases. For example, when a user edits a model schema, the IDE will need to convert the model data type to MySQL data type before performing the SQL alter statement. We will discuss more about the model schema editing in Section 5.3.1.1.

5.2.3 Adding Model to Controller Uses

CakePHP controllers will always have access to their primary models. If users want to access different models for certain controllers, then they can do so by using the $uses variable in the controller class definition. For example, If we want to access model “Book” from controller “BookstoreController”, then we can do so by adding “Book” to $uses variable.

```
$uses = array('Book', 'Bookstore');
```

Our IDE will automatically add or update the $uses variable whenever users add models to controller uses. Advanced users can still do this manually by editing the controller files. We will discuss more about adding model to controller uses in Section 5.3.1.6.

5.2.4 Constructing New Find Methods for Models

The find method is used to retrieve data from models. Our IDE is capable of automatically generating the required find methods based on the users' requests. The following example shows the automatically generated find method when users are trying to find all books authored by Jane Austen. The find method will be generated in the corresponding model php file.

```
function find_jane_austen_books() {
  return $this->find('all', array('recursive' => -1,'joins' => array(array('table' => 'authors', 'alias' =>
'Author', 'type' => 'INNER', 'conditions' => 'Book.id = Author.book_id')),'fields' => array('Book.id',
'Book.title', ),'conditions' => array('Author.fname' => 'Jane', 'Author.lname' => 'Austen', )));
}
```

Advanced users can still add their own find methods by manually adding them in the model php file. We will discuss more about adding new model find methods in Section 5.3.1.4.

5.2.5 Associating Different Models

CakePHP allows users to associate different models in four different types of associations: hasOne, hasMany, belongsTo, and hasAndBelongsToMany. Users can associate the models by including a variable named after the association type in the controller class definition. The following code will associate “Bookstore” and “Book” models in hasMany association.

```
var $hasMany = array('Book' => array('className' => 'Book', 'foreignKey' => 'bookstore_id',
'dependent' => true));
```

Advanced users can add associations between models manually by editing the model php file. We will discuss more the associating of different models in Section 5.3.1.5.

5.3 Front-end Components

The IDE GUI is composed of three major panels: left panel, center panel, and right panel. The left panel will display all the projects and their corresponding MVC components. The center panel will either display “edit mode” panel or “design mode” panel depending on the component that the user is working on. The right panel will either display the interactive help or instruction (for model and controller components) or the HTML elements tool (for view components).

The left panel will show all the projects and the corresponding model, view, and controller components in a tree view. When the IDE loads, the left panel will only show the projects. Users can click on the project name to see its model, view, and controller components. If the user clicks on the “Models” text, the IDE will display all the models created for that project. Clicking on the “Views” text will display all view files (.ctp files) created for that project, while clicking on the “Controllers” text will display all controllers created for that project.

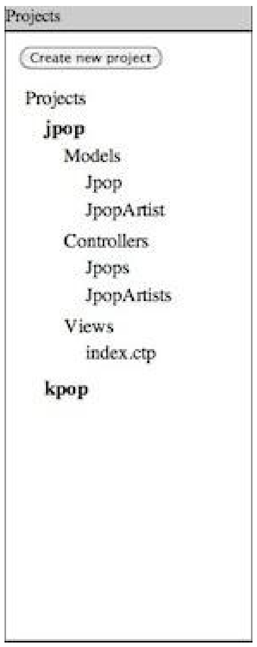


Figure 4: The Projects

The IDE will first query the database to get all the users' projects along with their models, views, and controllers components. We do this by issuing the find method in our IDE controller file:

```
$all_projects = $this->Ide->find('all',
  array('fields' => array('Ide.id', 'Ide.project_name', 'Ide.project_path'),
        'conditions' => array('Ide.user_id' => $this->Session->read('User.myuid')),
        'contain' => array('ModelComponent' => array('fields' => array('ModelComponent.id',
'ModelComponent.model_name', 'ModelComponent.model_filename')),
                          'ViewComponent' => array('fields' => array('ViewComponent.id',
'ViewComponent.view_filename')),
                          'ControllerComponent' => array('fields' => array('ControllerComponent.id',
'ControllerComponent.controller_name', 'ControllerComponent.controller_filename')))));
```

If the find method above returns any result, then the our IDE will display all of the users' projects and their components in the left panel. Besides displaying these components, it will also automatically generate JavaScript codes to store the information of each component to its respective DOM element. We store the information to each DOM element by using the jQuery data() function. The following code will show how the IDE store the project information to its respective DOM element.

```
$('span.projectid).data('cfg', {'project_id': project_id,
'project_name':project_name,'project_path':project_path});
```

The above JavaScript code will store the information of each project as an object to its span element. The information that it stores will be the project id, project name, and the directory of the project.

We use treeview jQuery plugin to display the list of the projects and their MVC components in tree structure.

```
$("#browser").treeview({ animated: 'fast', toggle: function() {
        console.log("%s was toggled.", $(this).find(">span").text());
});
```

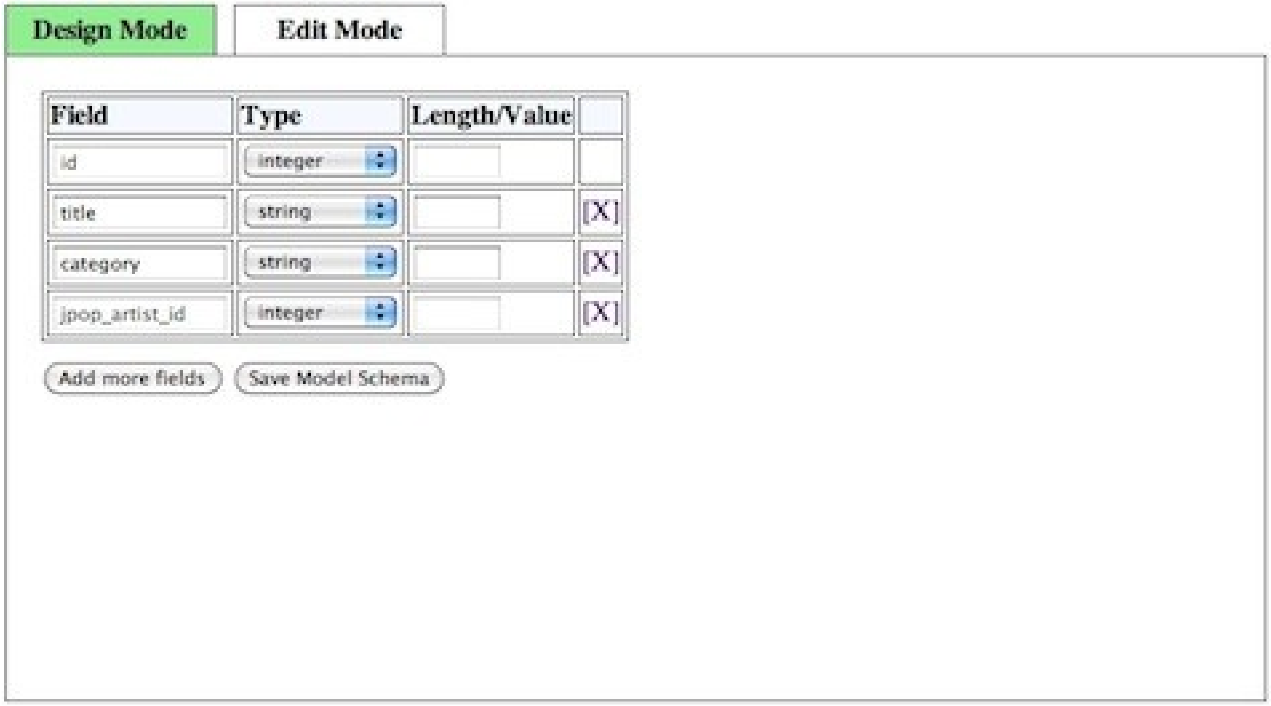
Upon the successful loading of the IDE, users will be able to right-click at the project names and their component names to perform several supported operations. We will discuss the supported features for the model and controller components. We will also discuss several other important features such as adding new projects/models/controllers, exporting projects to SQL files, and an interactive help system.

5.3.1 Model Components Supported Features

There are currently six supported features for model components. They are editing model schema, adding entries to model, deleting entries from model, constructing find methods for models, associating model with another models, and including model to be used by controller.

5.3.1.1 Editing Model Schema

When a user creates a new model, the IDE will automatically create a database table with one field ('id') for that model. User will be able to edit the model schema by right-clicking on the model name and select "Edit Model Schema". User will then be presented with the current schema in the center panel on design mode.



Field	Type	Length/Value	
id	integer		
title	string		[X]
category	string		[X]
jpop_artist_id	integer		[X]

Illustration 5: Edit Model Schema [Design Mode]

The field columns contain the attributes of the model. The id attribute is created automatically for the model, and it cannot be edited or deleted. Users can add more attributes, change the attribute names, change the attribute type, specify the length of the attribute value (for string), and delete attributes. The attribute types that are available here are the types supported by CakePHP. The IDE will automatically convert the types of attributes from CakePHP type to MySQL type. For the string type, users can limit the length of the strings by entering an integer value in the length/value column. If users want to add extra attributes, they can do so by clicking on the "Add more fields" button. Finally, when users are satisfied with their model schema, they can save it by clicking on the "Save Model Schema" button. The IDE will then send an AJAX call to the controller with the following parameters:

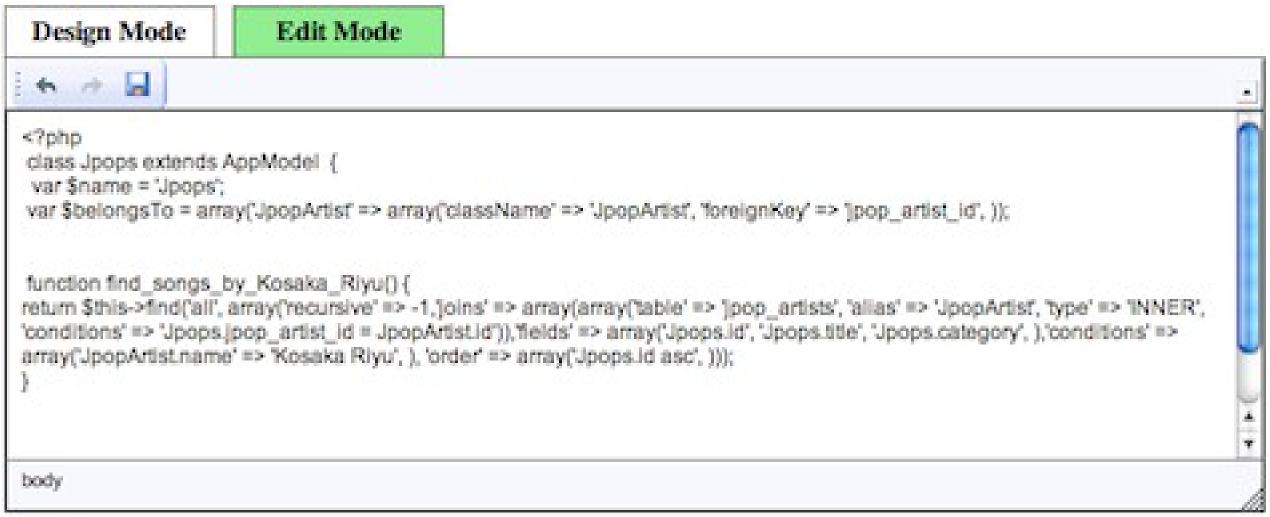
- project_name: the name of the project
- model_name: the name of the model
- fc : the number of attributes for this model.
- f<index>o : the old name of the attribute
- f<index>n : the new name of the attribute. This will be the same as f<index>o if this attribute is not changed.
- f<index>a : the action taken on this attribute. This parameter has three possible values: add, change, and drop. Add indicates that this is a new attribute. Change indicates that the user has changed the attribute. Drop indicates that the user has decided to drop this attribute.
- f<index>t : the data type of this attribute.
- f<index>v: the length/value of this attribute.

The controller will go through all of these parameters, set up a list of alterations required by the model, and send them to our IDE model. The model will create a SQL alter statement based on the list, and will execute it. If the SQL alter statement is executed successfully, the controller will return an empty string to the client-side JavaScript. If the SQL statement fails to execute, then the controller will return the error message to the client-side JavaScript.

More advanced users can also edit the model php file directly by clicking on the "Edit mode" tab. The IDE will display the CKEditor panel on the center panel along with the contents of the model php file. Users can edit the php file directly and save it. The IDE will get the contents of the php file from the CKEditor, and issue an AJAX call to the controller with these parameters:

- component_filename : the path of the file that users edit
- content: the new content of the file

If the file cannot be saved successfully, then the controller will return an error message to the JavaScript.

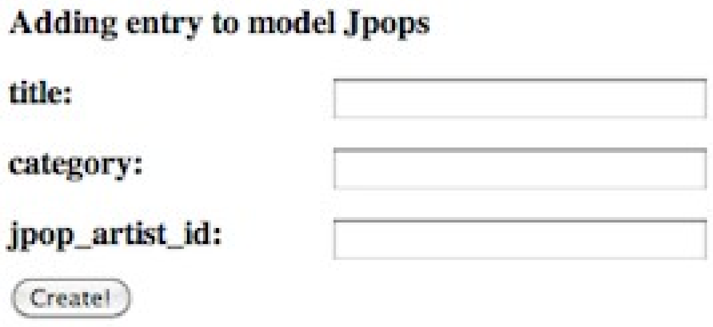


```
<?php
 class Jpops extends AppModel  {
  var $name = 'Jpops';
 var $belongsTo = array('JpopArtist' => array('className' => 'JpopArtist', 'foreignKey' => 'jpop_artist_id', ));


 function find_songs_by_Kosaka_Riyu() {
return $this->find('all', array('recursive' => -1,'joins' => array(array('table' => 'jpop_artists', 'alias' => 'JpopArtist', 'type' => 'INNER',
'conditions' => 'Jpops.jpop_artist_id = JpopArtist.id')),'fields' => array('Jpops.id', 'Jpops.title', 'Jpops.category', ),'conditions' =>
array('JpopArtist.name' => 'Kosaka Riyu', ), 'order' => array('Jpops.id asc', )));
}
```

Illustration 6: Editing Model Schema [Edit Mode]

5.3.1.2 Adding Entries to Model

Users can add more entries (data) to their models by right-clicking on the model name and choose "Add Entry". The center panel will display a form where users can enter the values for the attributes of the model. Once the user clicks on the "Create" button, the IDE will save the entries to the database table of that model. This task is equivalent to issuing an INSERT statement in SQL.

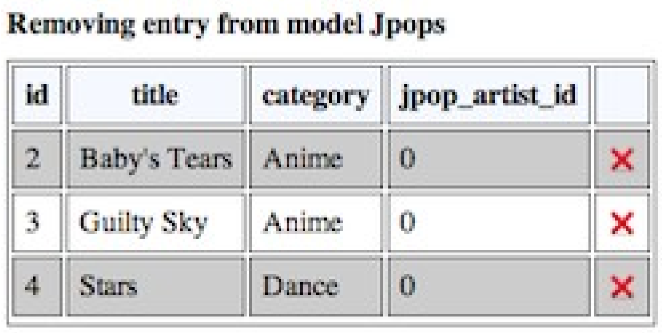

Adding entry to model Jpops

title:	
category:	
jpop_artist_id:	

Illustration 7: Adding Entry (Data) to Model

5.3.1.3 Deleting Entries from Model

Users can delete entries from their models by right-clicking on the model name and choose "Delete Entry". The center panel will display a form where users can delete any entries they want. This task is equivalent to issuing a DELETE statement in SQL.



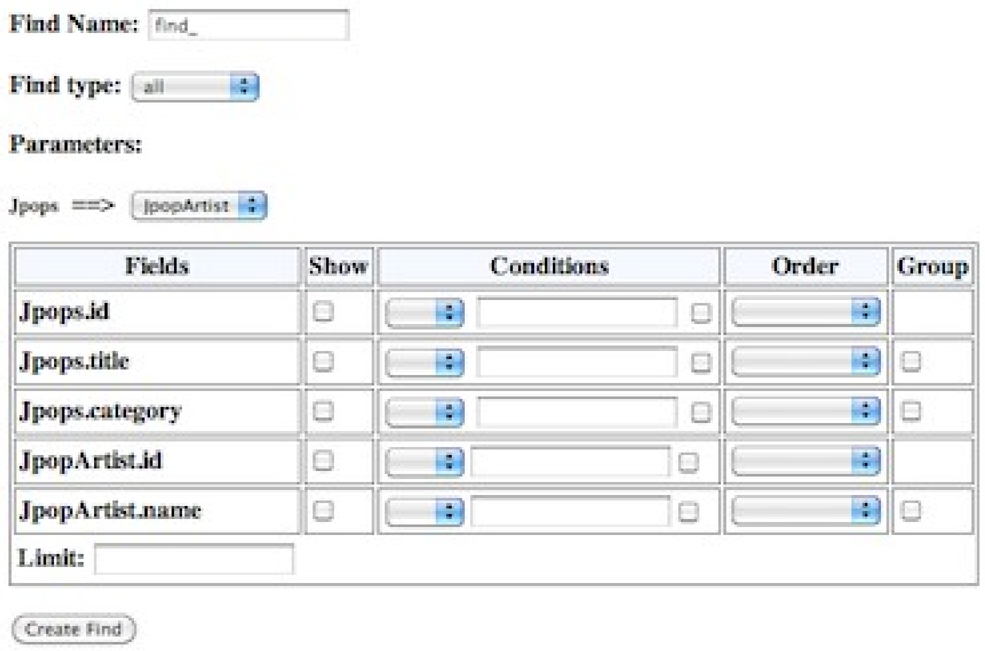
Removing entry from model Jpops

id	title	category	jpop_artist_id	
2	Baby's Tears	Anime	0	✗
3	Guilty Sky	Anime	0	✗
4	Stars	Dance	0	✗

Illustration 8: Removing Entry from Model

5.3.1.4 Constructing Find Methods for Models

Users can construct find methods for a model by right-clicking on the model name and select "Create Find". The center panel will display a form that users can use to create find methods. The find method is equivalent to issuing a SELECT statement in SQL and returning the results of the SELECT statement. Figure 9 shows the construct find interface.


Find Name: find_

Find type: all

Parameters:

Jpops ==> JpopArtist

Fields	Show	Conditions	Order	Group
Jpops.id	☐	☐		
Jpops.title	☐	☐		☐
Jpops.category	☐	☐		☐
JpopArtist.id	☐	☐		
JpopArtist.name	☐	☐		☐

Limit:

Create Find

Illustration 9: Constructing New Find method for Model

Users will need to give a name for the find method that they create. The name of the find method must start with word "find_".

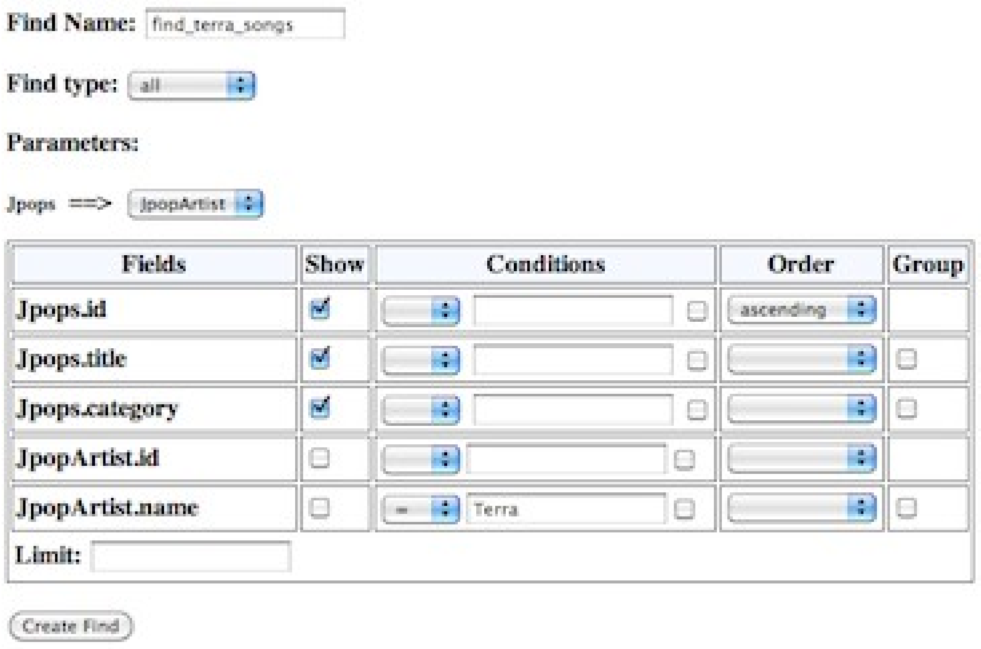
The form will allow users to choose from six different types of find that are supported by CakePHP:

- First : The find method will only return one result. Users can use this type if they expect only one result.
- Count: The find method will return the number of results.
- All: The find method will return an array of all results
- List: The find method will return the results in the form of an indexed array.
- Threaded: The find method will return the results as a nested array.
- Neighbors: The find method will return the rows before and after the row that is requested.

The navigation link above the table displays all models that are involved in creating the find method. The IDE will allow users to select the associated models to be included if the parent model has associations with other models. For example, if "Bookstore" model has hasMany associations with both "Book" and "Movie" models, then users will be able to choose either "Book" or "Movie" model to be included. When users choose associated models to be included, the table below the navigation link will refresh to include all attributes from the chosen associated models. Similarly, if users decide to cancel including the associated models, the table will refresh to remove all attributes from the associated models.

The create find table below the navigation link has five columns:

- Fields : The attribute names of the models. The attributes names will be displayed as <model name>.<attribute name>.
- Show : This column will indicate whether the particular attribute will be included in the find results.
- Conditions : This column will indicate the condition that is imposed on the attribute. Figure 10 illustrates the conditions that are required on the attribute name of JpopArtist model if we want to find all songs performed by Terra.



Find Name: find_terra_songs

Find type: all

Parameters:

Jpops ==> JpopArtist

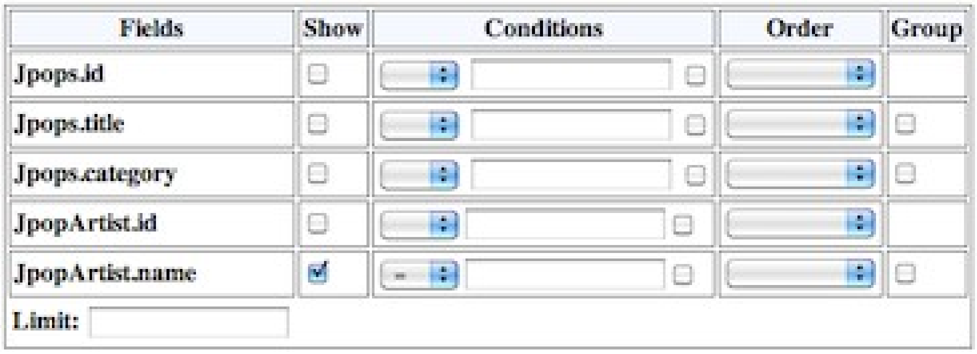
Fields	Show	Conditions			Order	Group
Jpops.id	✓			☐	ascending	
Jpops.title	✓			☐		☐
Jpops.category	✓			☐		☐
JpopArtist.id	☐			☐		
JpopArtist.name	☐	=	Terra	☐		☐
Limit:						

Create Find

Figure 10: Conditions on name attribute

Users can also create 'parameterized' conditions. For example, users might want to create a find method to get all books authored by author X (where X is the value supplied by users). To create parameterized conditions for attributes, users will click on the checkbox to the right of the input text in the conditions column. Figure 11 illustrates the parameterized conditions on the name attributes of

JpopArtist model.



Fields	Show	Conditions	Order	Group
Jpops.id	☐	☐		
Jpops.title	☐	☐		☐
Jpops.category	☐	☐		☐
JpopArtist.id	☐	☐		
JpopArtist.name	✓	= ☐		☐
Limit:				

Figure 11: Creating Parameterized Conditions

- Order : Users can control the ordering behavior of the results by specifying the ordering behavior of the attributes. For example, if users want to sort the results by the id attribute of Jpops model in descending mode, then users can select descending in the Order column.
- Group : Users can use the group column to group the results by particular attributes.

Users can limit the number of results returned by the find method by specifying the limit. If users do not specify the limit, the IDE will not include the limit in the find method.

When users submit their create find method form, the JavaScript will issue an Ajax call to the controller with these parameters:

- project_id : the id of the current project
- project_name : the name of the current project
- model_id : the id of the main model
- model_name : the name of the main model
- model_filename : the path of the main model php file
- find_name : the name for the find method
- find_params : a JavaScript object that has the following properties
  - find_type : the type of the find ('first', 'all', 'count', 'threaded', 'list', 'neighbors')
  - parameterized : an array that has all the parameterized attributes
  - fields_show : an array that has the attributes of the models that will be included in the

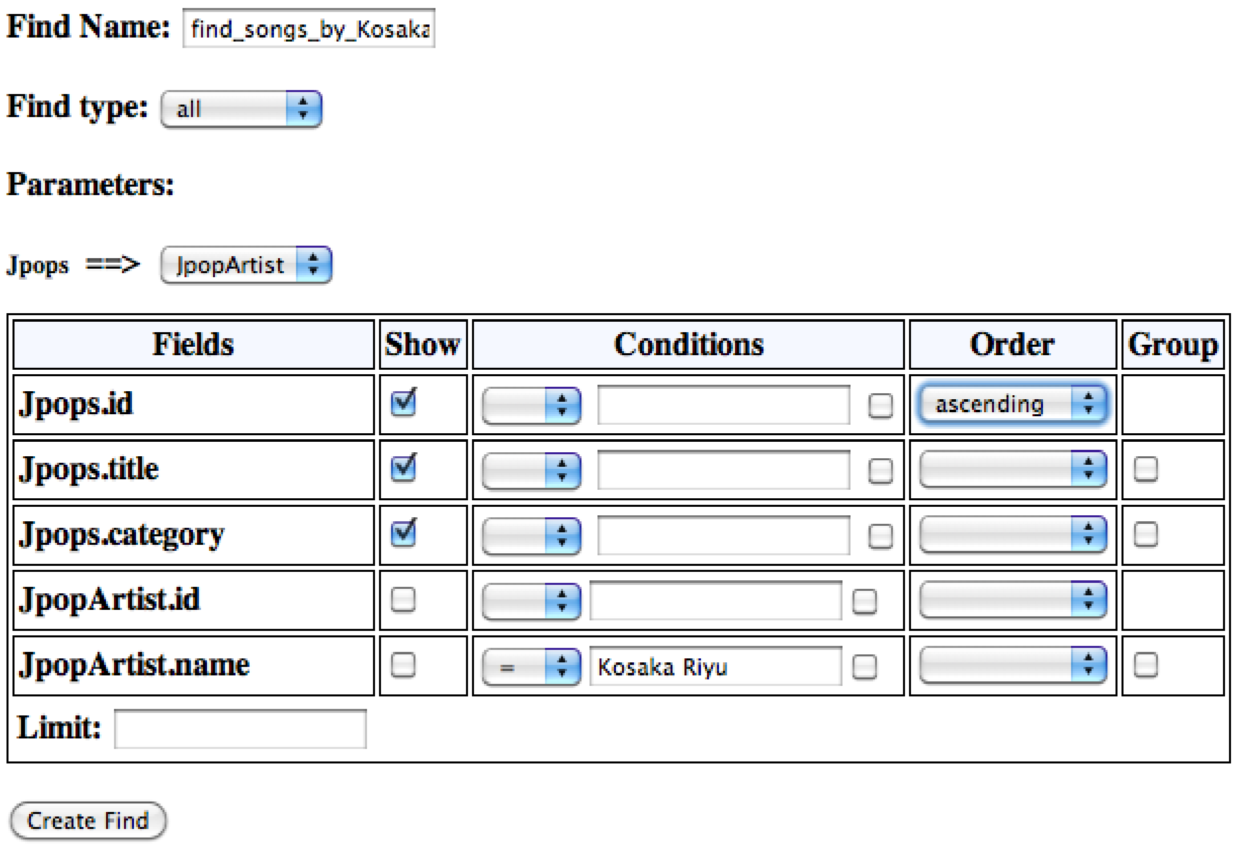
results

  - fields_cond_val : an array of the conditions imposed on the attributes
  - fields_order : an array of attributes ordering behaviors.
  - fields_group : an array of grouped attributes

- limit : the maximum number of results that the find method will return
- model_ids : the id of all models involved in the find method

The Controller will pass all the parameters to the model, and the model will generate the find method and include it in the model php file.

Figures 12 and 13 show an example of the create find method form and the modified find method code in the model php file.



Find Name: find_songs_by_Kosaka

Find type: all

Parameters:

Jpops ==> JpopArtist

Fields	Show	Conditions	Order	Group
Jpops.id	✓		ascending	
Jpops.title	✓			☐
Jpops.category	✓			☐
JpopArtist.id	☐			
JpopArtist.name	☐	= Kosaka Riyu		☐

Limit:

Create Find

Figure 12: Find all songs performed by Kosaka Riyu

```
function find_songs_by_Kosaka_Riyu() {
return $this->find('all', array('recursive' => -1,'joins' => array(array('table' => 'jpop_artists',
'alias' => 'JpopArtist', 'type' => 'INNER', 'conditions' => 'Jpops.jpop_artist_id =
JpopArtist.id')),'fields' => array('Jpops.id', 'Jpops.title', 'Jpops.category', ),'conditions' =>
array('JpopArtist.name' => 'Kosaka Riyu', ), 'order' => array('Jpops.id asc', )));
}
```
Figure 13: The generated PHP code in the model php file

5.3.1.5 Associating Models with Other Models

Users can link different models by creating associations between them. The IDE will allow users to associate models in either hasMany, hasOne, or belongsTo association types. Users can create the association by right-clicking on the model name and choose “Associate w/ Another Model” option. The center panel will display a form that lets users create associations between models.

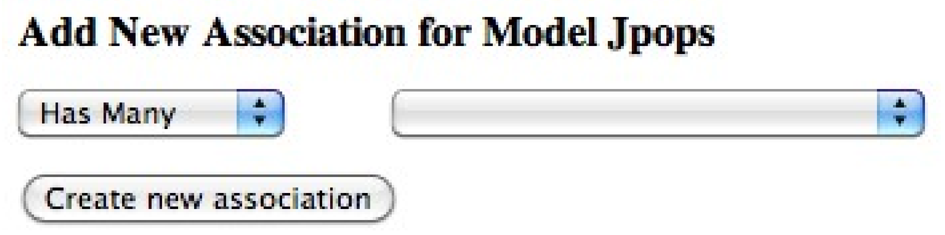

Add New Association for Model Jpops

Figure 14: Creating Associations between Models

The form will only show the models that are not yet associated with the main model. This will prevent users from accidentally associating two models twice.

When a user submits this form, the JavaScript will issue an Ajax call to the controller along with these parameters:

- project_id : the id of the current project
- project_name : the name of the current project
- model1_id : the id of the main model
- model1_name : the name of the main model
- model1_filename : the path of the main model php file
- model2_id : the id of the second model. This is the model that the main model will be associated with

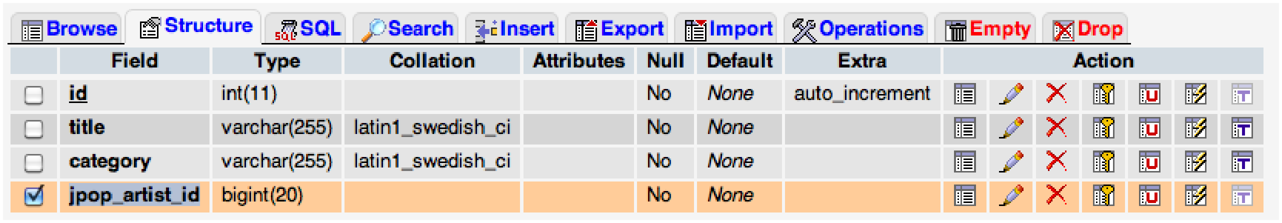
- model2_name : the name of the second model
- assoc_type : the type of the association

The controller will automatically generate the PHP code in the model php file to form the associations between the models. The following code is an example of the PHP code that is automatically generated when we form a hasMany association between “JpopArtist” and “Jpop” models. The following code will be generated in the Bookstore model php file.

```
var $hasMany = array('Jpops' => array('className' => 'Jpops', 'foreignKey' => 'jpop_artist_id',
'dependent' => true));
```

The foreignKey parameter indicates the name of the foreign key that will link the tables bookstores and books. The foreign key must be added to the books table to ensure the successful association between JpopArtist and Jpop models. If the dependent parameter is set to true and the model delete() method is called with cascade parameter set to true, the associated model records will also be deleted. By default, our IDE will always set the dependent parameter to true.

The IDE will also create the required foreign keys to link the bookstores and books database tables together. Users will not need to do this manually. Figure 14 illustrates the schema of the jpops table after we associate the JpopArtist and Jpop models. We observe that the jpop table has jpop_artist_id as the foreign key.



	Field	Type	Collation	Attributes	Null	Default	Extra	Action
☐	id	int(11)			No	None	auto_increment	
☐	title	varchar(255)	latin1_swedish_ci		No	None		
☐	category	varchar(255)	latin1_swedish_ci		No	None		
☑	jpop_artist_id	bigint(20)			No	None		

Figure 15: The schema of books table after the association

5.3.1.6 Including Models to be Used by controller

Our IDE allows users to enable controllers to access other models in the same project (other than the primary model). Users can set this up by dragging a model and drop it in a controller. The only restriction is users cannot enable controllers to access models from different projects.

When users drop the model in the controller, the JavaScript will issue an Ajax call to the IDE controller file with the following parameters:

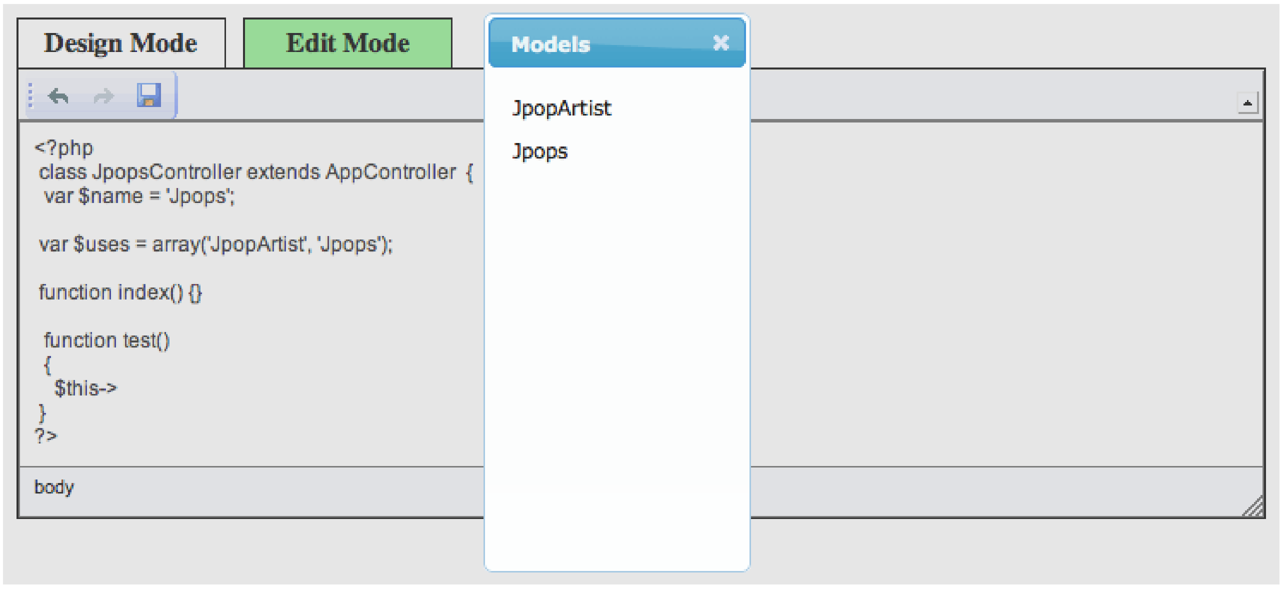
- project_name : the name of the current project
- model_id : the id of the main model
- model_name : the name of the main model
- controller_id : the id of the controller
- controller_name : the name of the controller
- controller_filename : the path of the controller php file

The IDE controller will invoke its model to automatically generate the required PHP code and insert it to the controller php file.

5.3.2 Controller Components Auto-suggest Feature

Users can edit the controller file by right-clicking on the controller name and select "Edit controller file". The Center panel will display the content of the controller php file inside the CKEditor panel. The CKEditor panel will have an auto-completion feature. When users are editing a controller file, the CKEditor panel will automatically display the possible model and model methods that the controller has access to. The CKEditor panel achieve this feature by issuing an Ajax call to the IDE controller file whenever users type $this-> or $this->model_name->.

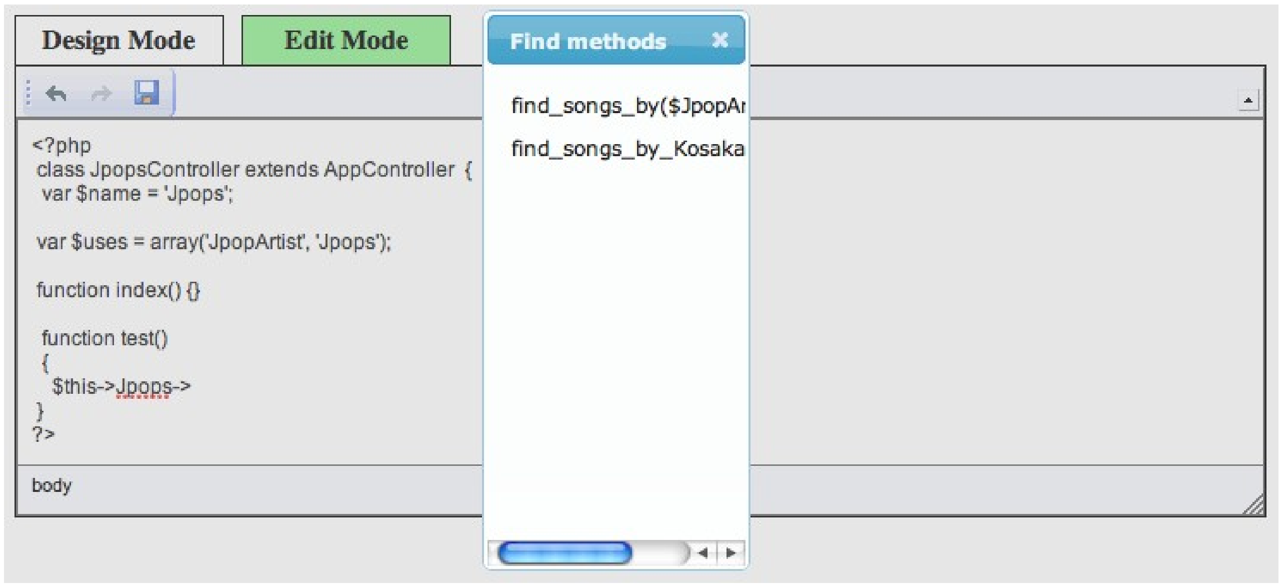
When users type $this->, the JavaScript will assume that users are trying to access models and bring up a dialog box. The dialog box will have all the models that can be accessed by the controller. Figure 15 illustrates the dialog box that has all the models that are accessible by the controller.



```
<?php
 class JpopsController extends AppController {
  var $name = 'Jpops';

 var $uses = array('JpopArtist', 'Jpops');

 function index() {}

  function test()
  {
    $this->
 }
?>
```

Figure 16: Dialog box of accessible models

When users type $this->model_name->, the JavaScript will assume that the users are trying to access the methods of the model, and bring up a dialog box. The dialog box will have all the methods of that model. Figure 16 illustrates the dialog box that has all the methods of that model.



```
<?php
 class JpopsController extends AppController {
  var $name = 'Jpops';

 var $uses = array('JpopArtist', 'Jpops');

 function index() {}

  function test()
  {
    $this->Jpops->
 }
?>
```

Figure 17: Dialog box of all methods of model

5.3.3 Other Features

We will discuss several other features that are critical for the IDE.

5.3.3.1 Adding New Projects

Users can create new projects by clicking on the "Create New Project" button or select "File – New Project" from the menu bar. Users will then be presented a dialog box that they need to fill out to create a new project. Users will have to enter the project name, the database system used, the database server host, and the username/password to access the database. When users click on the create button, several events will happen on the back-end side.

- The IDE will create a new directory for the project and copy all the required files needed for that project to that directory. All of these required files are located in directory controllers/new_project folder under our IDE directory.
- The IDE will create basic model, view, and controller files for the project.
- The IDE will create a database and a base table for the project. The base table will have only one field "id".
- The IDE will create a database configuration for the project.
- Finally, the IDE will save the information of the project and the MVC components to the database. The project information will be stored in the ides table. The model information will be stored in the model_components table, controller information will be stored in the controller_components table, and view information will be stored in the view_components table.

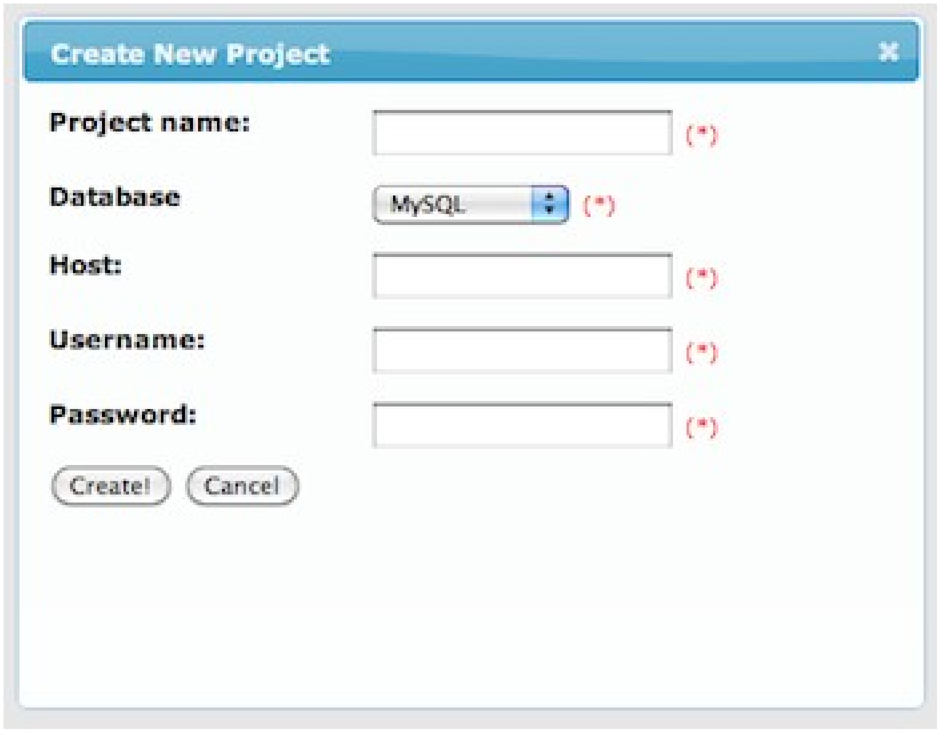


Figure 18: Create New Project Dialog Box

5.3.3.2 Adding New Models or Controllers

Users can create new models by right-clicking on the 'Models' text under each project and choose 'Create New Model'. Users will only need to enter the new model name. The IDE will create a basic model php file for that new model. Users can create new controllers by right-clicking on the 'Controllers' text under the project and choose 'Create new Controller'. Once the users enter the controller name, the IDE will create a basic controller php file for that new controller. Besides creating the new model or controller php files, the IDE will also store the new model or controller component information in the model_components or controller_components database table.

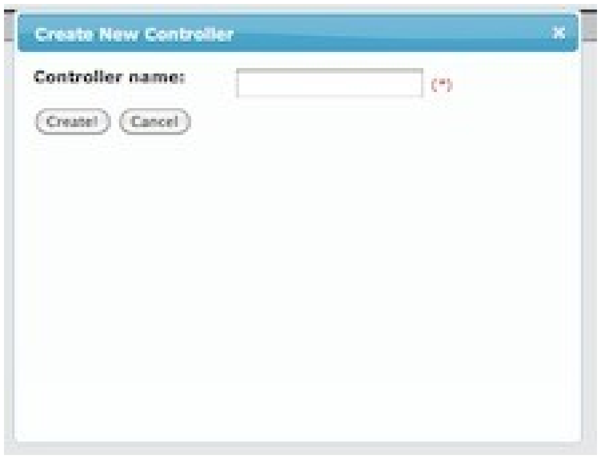


Figure 19: Create New Controller Dialog Box

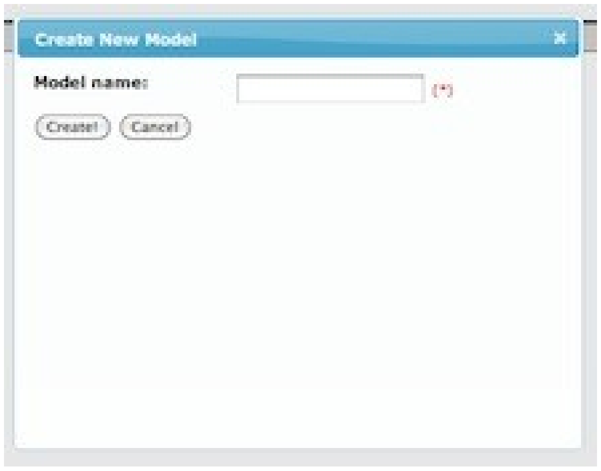
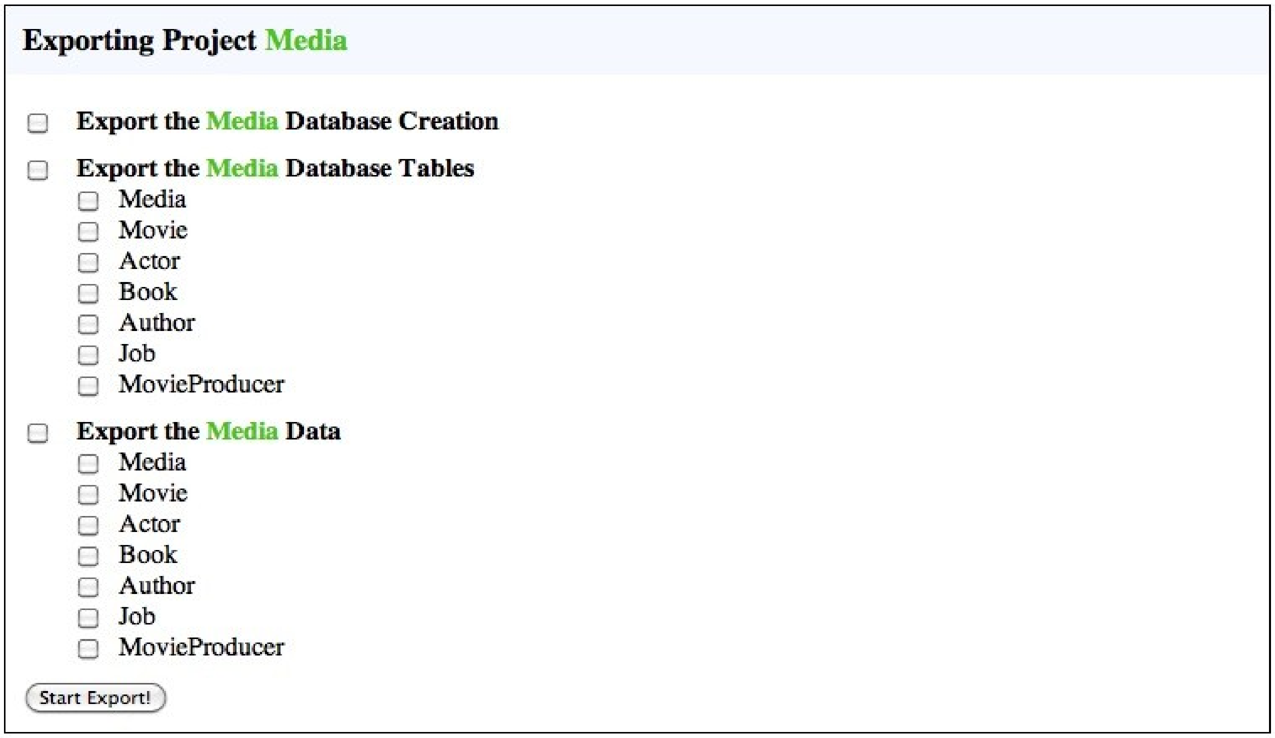


Figure 20: Create New Model Dialog Box

5.3.3.3 Exporting Projects to SQL Files

If users want to host their project on a different server, then they will need to copy the project files to that server. They will also need to export the project database if the database server is also hosted on a different server. The IDE offers a simple interface that will allow users to export the project database to an SQL file. Users can access this interface by right-clicking on the project name and choose “Export Project”.


Exporting Project Media

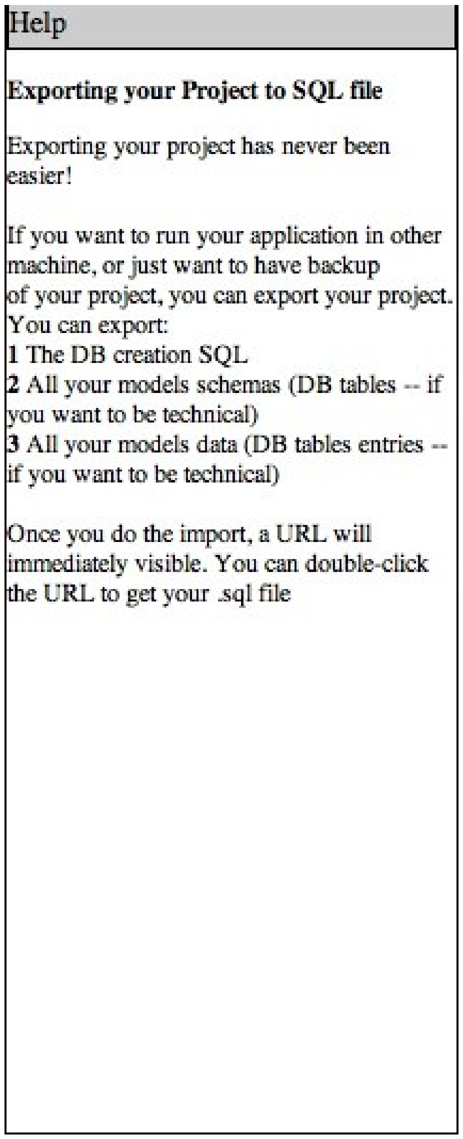
- [ ] **Export the Media Database Creation**
- [ ] **Export the Media Database Tables**
  - [ ] Media
  - [ ] Movie
  - [ ] Actor
  - [ ] Book
  - [ ] Author
  - [ ] Job
  - [ ] MovieProducer
- [ ] **Export the Media Data**
  - [ ] Media
  - [ ] Movie
  - [ ] Actor
  - [ ] Book
  - [ ] Author
  - [ ] Job
  - [ ] MovieProducer

Figure 21: Exporting Project Interface

Users can choose to export the SQL statement to create the database, the tables in the database, and the data in the tables. The IDE will automatically generate all the SQL statements and write them to a SQL file. Once the SQL file is ready, a download link will be visible on the interface. Users can click on that link to download the SQL file. Once users have downloaded the SQL file, they can run the SQL script on the new database server.

5.3.3.4 Interactive Help

When users perform certain models or controller operations, an interactive help will appear on the right panel of the GUI. The interactive help will offer brief tutorials or hints of the operations performed by users. The following figure shows an example of the interactive help that will appear when users try to export their project to an SQL file.


Help

Exporting your Project to SQL file

Exporting your project has never been easier!

If you want to run your application in other machine, or just want to have backup of your project, you can export your project.
You can export:
1 The DB creation SQL
2 All your models schemas (DB tables -- if you want to be technical)
3 All your models data (DB tables entries -- if you want to be technical)

Once you do the import, a URL will immediately visible. You can double-click the URL to get your .sql file

Figure 22: Interactive Help

The interactive help will be very useful since users will be able to learn more about the operations that they choose to perform. The interactive help also offers examples on constructing new find method for a model.

6. Issues Encountered and Solutions

During the development phase of our CakePHP IDE, we encountered several issues. We will now discuss the issues that we encountered and how we solved them.

6.1. Instantiating the Model or Controller Components from Users' Created Projects

One of the critical core functionalities of our CakePHP IDE is the ability to instantiate the model or controller components from the users' created projects. The following code shows how we can instantiate a model or controller component.

```
/**
  $comp_name – the name of the model/controller component
  $db_config_name – the name of the database config to be used
*/
$comp_obj = new $comp_name(false, null, $db_config_name);
```

The issue with the above code is that CakePHP will attempt to find the DB configuration from the file config/database.php in our IDE directory. This will fail because the DB configuration is stored in database.php file located in users' project directory.

Our solution is to get all the users' projects and their DB configurations during the instantiation of the DATABASE_CONFIG object. For each project, we will instantiate an array variable that holds the DB configuration of that project. The following code shows how this is accomplished.

```
function DATABASE_CONFIG()
{
 if(@mysql_connect($this->default['host'], $this->default['login'], $this->default['password']))
 {
  if(@mysql_select_db($this->default['database']))
  {
   $result = @mysql_query("SELECT project_name, project_db_config FROM ides");
   if($result)
   {
    while($row = @mysql_fetch_object($result))
    {
     $project_name = $row->project_name;
     $project_db_conf = unserialize($row->project_db_config);
     $this->$project_name = array(
      'driver' => $project_db_conf['driver'],
      'persistent' => $project_db_conf['persistent'],
      'host' => $project_db_conf['host'],
      'login' => $project_db_conf['login'],
      'password' => $project_db_conf['password'],
      'database' => $project_db_conf['database'],
      'prefix' => $project_db_conf['prefix']);
    }
   }
  }
  @mysql_close();
 }
}
```

The above PHP code will create many instance variables that hold the DB configuration for the projects. Since we use the name of the project as the instance variable name, we can instantiate any model or controller components by using a valid DB configuration.

One side effect of our approach is that we have to refresh our IDE page whenever we create a new project. We need to do this because we have to load the DB configuration of the newly created project.

6.2 Instantiating the Model Components that Have Associations

The core CakePHP libraries currently do not provide good support for the instantiation of the models that have associations. The instantiation will fail if both the main model and its associated models are from different project. This is a blocker issue for us because the IDE must be able to instantiate the models from the projects that the users create.

Our first attempt to solve this issue is by including the php files of the main model and its associated models. We will then instantiate the main model.

```
App::import('Model', $child_model1, array('file' => $child_model1_path));
App::import('Model', $child_model2, array('file' => $child_model2_path));
App::import('Model', $child_modelN, array('file' => $child_modelN_path));
App::import('Model', $main_model, array('file' => $main_model1_path));
$model_obj = $main_model(false, null, $db_config);
```

We later discovered that our first solution would not work if both the main model and its associated models had associations with each other. For example, model "Bookstore" has hasMany association with model "Book" and model "Book" has belongsTo association with model "Bookstore". Our first solution will result in an infinite loop.

After investigating the core CakePHP model.php file, we found out that the source of our issue is the __constructLinkedModel function. This function attempts to instantiate the associated models with incorrect database config. We fix this issue by adding the DB config parameter to the array in line 656.

```
$model = array('class' => $className, 'alias' => $assoc, 'ds' => $this->useDbConfig);
```

6.3 jQuery live() function does not work with onChange events on Internet Explorer Web browsers (Browser Compatibility Issue)

We encountered this issue when we implemented the construct find interface. When users select the associated models, the find table should refresh to display all attributes of the selected models. When we tested this on Internet Explorer 8 browser, the table did not refresh. This happens because the jQuery live() function sets all the event handlers on the document level instead of on elements directly. This means that live() function needs the event bubbling to work. However, onChange events do not bubble up in Internet Explorer Web browsers. We fixed this issue by using livequery plugin. Livequery does not require event bubbling because it attaches the event handlers on elements directly [4].

6.4 Interactivity Issue

It is always a challenge to make a Web-based application to behave interactively as in desktop-based applications [7]. We had tried to make our CakePHP IDE to be as interactive as possible by using some of the jQuery UI features such as dialog box and drag-and-drop. We have also added the auto-completion feature for the CKEditor to make our IDE more interactive.

7. Performance

We measured the time our IDE took to complete an operation. Since our CakePHP IDE is Web-based, the time will include both the latency and the download time. We got the average times of the ten critical operations from our IDE by using the developer tool in Safari Web browser. Since we were unable to find an existing CakePHP IDE, we decided to compare the performance of our CakePHP IDE against the performance of ModelBaker. ModelBaker is a desktop-based application that lets the users create CakePHP-based Web applications. ModelBaker is currently only available in Mac platform and it can be downloaded from http://www.widgetpress.com/modelbaker/download/. Below is the list of the ten critical operations that we tested and their average times for our CakePHP IDE and ModelBaker.

Operations	**CakePHP IDE Average Time [Latency and Download]**	**ModelBaker Average Time**
Creating new project	343 ms	219 ms
Reading model or controller files	191 ms	N/A. ModelBaker does not support manual editing of php files.
Saving model or controller files	157 ms	N/A. ModelBaker does not support manual editing of php files.
Saving Model Schema	200 ms	212 ms
Adding New Entry to Model	195 ms	N/A. Users need to manually add entries into the database.
Deleting Entry from Model	120 ms	N/A. Users need to manually delete the entries from the database.

Creating New Model	201 ms	218 ms
Creating New Controller	118 ms	219 ms
Associating Two Models	206 ms	223 ms
Creating New Find Method	182 ms	N/A. ModelBaker does not support creating find methods for model.

The performance of our CakePHP IDE is slightly worse than the performance of the ModelBaker for creating new project operation. However, we observed that the performance of ModelBaker was very similar for all other operations. This happened because ModelBaker would always rebuild the application regardless of which operation that we performed.

We had also measured the time it takes for our CakePHP IDE page to load. It is very important that our IDE loads as fast as possible because most users will lose interest in using it if the loading time is too long. By using the developer tool in Safari Web browser, we observed that our IDE loading time is one second on average.

8. Usability Testing

We conducted a usability testing with five users. All of ours users are experienced PHP developers and they have experiences in developing Web applications by using CakePHP. We would like to find out whether our users were able to perform certain tasks with our IDE without any help or guidance from us. We asked our users to perform these tasks sequentially.

1. Create a new account and login with that account
2. Create a new project titled "Bookstore"
3. Edit the schema of Bookstore model by adding attributes bookstoreName (string) and bookStoreOwner (string)
4. Add a new model titled "Book"
5. Edit the schema of Book model by adding attribute bookTitle (string)
6. Add a hasMany association between models Bookstore and Book
7. Add a new model titled "Author"
8. Edit the schema of Author model by adding attributes authorFirstName (string) and authorLastName (string)
9. Add a belongsTo association between models Book and Author
10. Add some entries (data) for the models Bookstore, Book, and Author
11. Delete some entries (data) from the models Bookstore, Book, and Author
12. Add a model to a controller
13. Create a find method to get all books authored by "Jane Austen"

We also asked our users to rate the difficulty in figuring out how to perform the following tasks on the scale from 1 to 5. (1 = cannot figure out, 5 = very easy).

1. Create a new account and login to that account
2. Create a new project
3. Edit the schema of a model
4. Add new model
5. Associate models

6. Add new entries for a model
7. Delete entries from a model
8. Add a model to a controller
9. Create a new find method for a model

Three of our users had reported difficulty in figuring out the right-click mechanism of our CakePHP IDE because they did not pay close attention to the interactive help system. Once they had figured out the right-click mechanism, they were able to perform most of the other tasks. Three of our users had also reported difficulty in figuring out how to add a model to a controller because they thought that this operation was performed by right-clicking. Finally, they ended up adding a model to a controller by manually editing the controller file. Our users also said that the create find interface is quite useful but the interactive help did not provide enough examples. Overall, our users reported that the IDE simplify the process of creating a CakePHP project with its model and controller components. All of our users had suggested that we enhance our help system by adding more tutorials and examples to our IDE. Figure 23 shows the result of our usability testing.

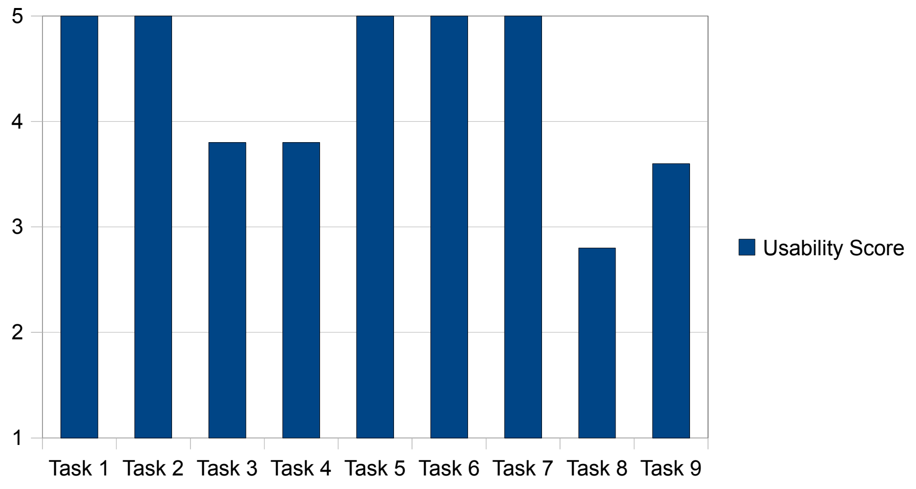


Figure 23: The Usability Testing Result

We conducted another usability test where we asked users to use their favorite IDE to perform the 13 tasks above. We also asked them to rate the difficulty of performing the eight tasks (without the registration/login task) above on the scale from 1 to 5. (1 = cannot figure out, 5 = very easy). Most of users used desktop-based IDE like Notepad++, EditPlus, or . Most of our users reported that low rating for those tasks except the adding model to controller task. Most of them thought that adding a new project task is the hardest task to perform since it requires so many steps. They also found the other tasks hard because they need to interact with the database directly. Every users reported that adding model to controller was the easiest task to perform. Figure 24 shows the result of our second usability testing

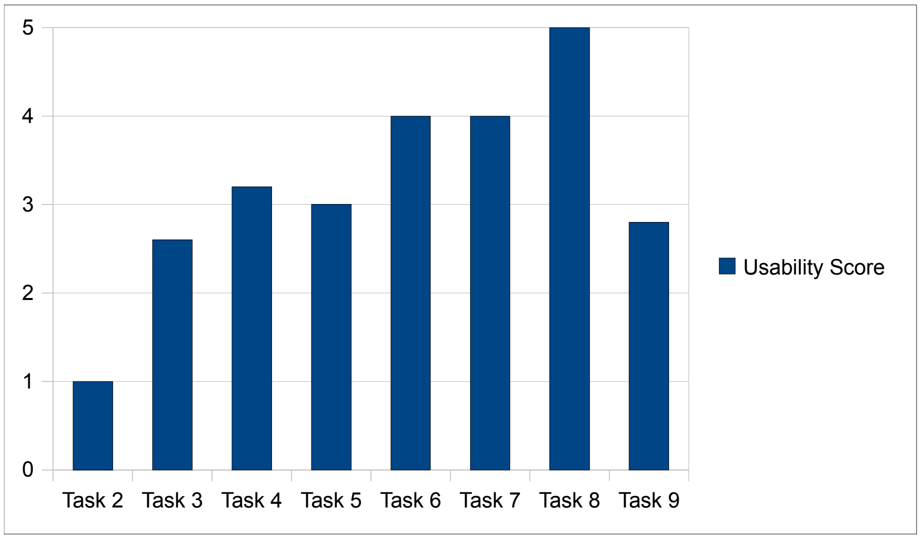


Figure 24: The Second Usability Testing Result

From the two graphs above, we can see that our CakePHP IDE excels in almost all of the tasks. In our second usability test, users report very high usability score for adding model to controller task. We expect this result because users only need to add the model name to the $uses variable in the controller PHP file.

9. Conclusion

Our ultimate goal in this project is to create a user-friendly CakePHP IDE that will automate most of the processes involved in creating CakePHP Web applications. We have achieved this goal by implementing many features that will let users easily create CakePHP Web applications. With our IDE, users do not have to deal with creating required model, controller, and view files for the project. Users also do not need to deal explicitly with the database as our IDE will handle most of the interactions with the database.

Since CakePHP uses the “coding by convention” paradigm, users have to be extremely careful if they want to create CakePHP Web applications manually. Users have to make sure that they adhere to CakePHP naming conventions when they create their applications. Any single failure in adhering to the CakePHP naming conventions can lead to long and frustrating debugging sessions. By adhering to CakePHP naming conventions, our CakePHP IDE will relieve users from these long and frustrating debugging sessions.

During the implementation phase of our project, we faced some challenging tasks. One of these tasks was to enable our IDE to instantiate any model or controller components from users' created projects. We had to figure out how to use the DB configurations of users' projects to create the model or controller components. We also had to deal with the case in which a model has associations with other models. Finally, we managed to solve this task by fixing a bug in core CakePHP library script.

We had tried to make our CakePHP IDE to be as interactive as possible. We had used jQuery UI to create interactive dialog boxes and to enable drag-and-drop interactions. We also used jQuery ContextMenu plugin to enable right-click.

During the performance testing, we observed that our IDE took roughly the same time as the time taken by ModelBaker for most operations except creating the project. ModelBaker takes less time in creating the projects because it creates all the required files in client's machine where as our IDE creates the required files on the Web server.

We conducted a usability testing and the result shows that our users found our CakePHP IDE to be relatively easy to use. The majority of our users had problems in adding a new model to controller uses because they got confused with the drag-and-drop mechanism of the operation. They also thought that the create find interface was very useful but the help system did not provide enough examples about the interface. Overall, our users found our CakePHP IDE easy to use and interactive, but they suggested to have a better help system with proper examples. We also conducted another usability test where we asked users to use their favorite IDE. All of the users reported that it was harder to accomplish almost of the tasks with their IDE. They reported that our IDE excelled in most of the tasks. They preferred our IDE because our IDE automates most of the processes involved in creating a CakePHP Web application.

By successfully completing our project, we had learned a lot about the mechanisms of MVC-based Web frameworks. We had also learned a lot in building interactive Web applications by using jQuery, jQuery UI, and other JavaScript scripts. We hope that our IDE will help users to have better understanding of the MVC Web framework and to create sophisticated CakePHP Web applications with ease.

10. References

[1] *Integrated Development Environment. (n.d.).* Retrieved from http://en.wikipedia.org/wiki/Integrated_development_environment

[2] *Understanding Model-View-Controller. (n.d.)*. Retrieved from http://book.cakephp.org/view/10/Understanding-Model-View-Controller

[3] *Convention over Configuration. (n.d.)*. Retrieved from http://en.wikipedia.org/wiki/Convention_over_configuration

[4] Neeraj Singh (2009, Oct 14). *How live method works in jQuery. Why it does not work in some cases. When to use livequery.* Retrieved from http://www.neeraj.name/2009/10/14/how-live-method-works-in-jquery-why-it-does-not-work-in-some-cases-when-to-use-livequery.html

[5] Stefanov, S. (2010). *JavaScript Patterns*. Sebastopol, CA : O'Reilly

[6] *Find :: Retrieving Your Data :: Models :: Developing with CakePHP :: The Manual :: 1.2 Collection :: The Cookbook. (n.d.)*. Retrieved from http://book.cakephp.org/view/449/find

[7] Spool, Jared M (2007, Dec 4). *Five Usability Challenges of Web-Based Applications*. Retrieved from http://www.uie.com/articles/usability_challenges_of_web_apps/