

Dokumentacja projektu pt. „ Serwis kulinarny ”
zrealizowanego w ramach przedmiotu Aplikacje
Internetowe.

Grupa:
F1C-DU L5
Autorzy:
Flis Mariusz
Głodzik Krzysztof

1. Cel Projektu

Celem projektu było stworzenie internetowego serwisu kulinarnego wraz z bazą danych. Serwis miał realizować szereg funkcjonalności: rejestrowanie użytkownika, logowanie, wyszukiwarka przepisów, dodawanie/usuwanie komentarzy i przepisów, dodawanie/usuwanie zdjęć do umieszczanych na serwisie przepisów. Gotowy serwis miał zostać zaimplementowany na serwerze WWW.

2. Specyfikacja techniczna

Projekt został zrealizowany w CakePHP ver. 2.6.3. CakePHP jest frameworkiem języka PHP, który powstał na bazie inspiracji Ruby on Rails. CakePHP opiera się na idei wykorzystania konwencji nazewnictwa i miejsca przechowywania elementów aplikacji w celu zminimalizowania konfiguracji. Struktura katalogów aplikacji jest ustalona. Katalog „app” reprezentuje samą aplikację, „cake” kod frameworka, a „vendors” wspólne dodatki (klasy obce). W ujęciu CakePHP elementy MVC reprezentują:

- Model - odzwierciedla konkretną tabelę bazy danych i jej relację z innymi tabelami. Zawiera reguły dotyczące typu przechowywanych danych stosowane podczas ich wstawiania i aktualizacji (wzorzec ActiveRecord). Model rozszerza klasę AppModel. Dla każdej tabeli bazy danych używanej w aplikacji musi istnieć jej model. Modele znajdują się w katalogu „app/models”. W pliku konfiguracyjnym znajdują się informacje o bazie. Dostępne bazy to: MySQL, PostgreSQL, SQLite, drivery do PEAR oraz ADO.
- Kontroler - obsługuje żądania serwera, pobiera dane wejściowe od użytkownika (w formie adresu URL i danych POST), stosuje reguły logiki biznesowej. Następnie używa modelu, aby czytać i zapisywać informacje z bazy danych lub innych źródeł. Ostatecznie kontroler wysyła dane wyjściowe do widoku w celu ich wyświetlenia. Kontroler rozszerza klasę AppController. Kontrolery znajdują się w katalogu „app/controllers”, według konwencji nazwa controller.php.
- Widok - szablon strony HTML ze zintegrowanym kodem PHP odpowiedzialnym za wyświetlenie danych wyjściowych aplikacji. Widoki przechowywane w podkatalogach „app/views” nazwanych odpowiednio do nazw obsługujących kontrolerów. Pliki widoków są nazwane tak, jak akcje kontrolera z rozszerzeniem .ctp. Istnieje możliwość stworzenia layoutu, który będzie wielokrotnie wykorzystany.

W czasie powstawania projektu nie była jeszcze dostępna stabilna wersja 3.1. Wszystkie wersje dostępne są do pobrania pod adresem: www.cakephp.org lub bezpośrednio za pośrednictwem linku: <http://book.cakephp.org/2.0/en/installation.html> . Do poprawnego

działania aplikacji, którą stworzymy niezbędnych jest jeszcze kilka dodatków, o których więcej informacji znajduje się poniżej.

3. Użyte oprogramowanie

Do działania aplikacji niezbędny jest serwer HTTP oraz baza danych MySQL. Oba te wymagania załatwia zainstalowanie jednego programu:

- XAMPP- darmowy wieloplatformowy, zintegrowany pakiet, składający się głównie z serwera Apache, bazy danych MySQL i interpreterów dla skryptów napisanych w PHP i Perlu. Nazwa XAMPP jest akronimem od X (ang. *cross-platform*), Apache, MySQL, PHP, Perl. Najnowsza wersja programu do pobrania pod adresem: www.apachefriends.org/index.html

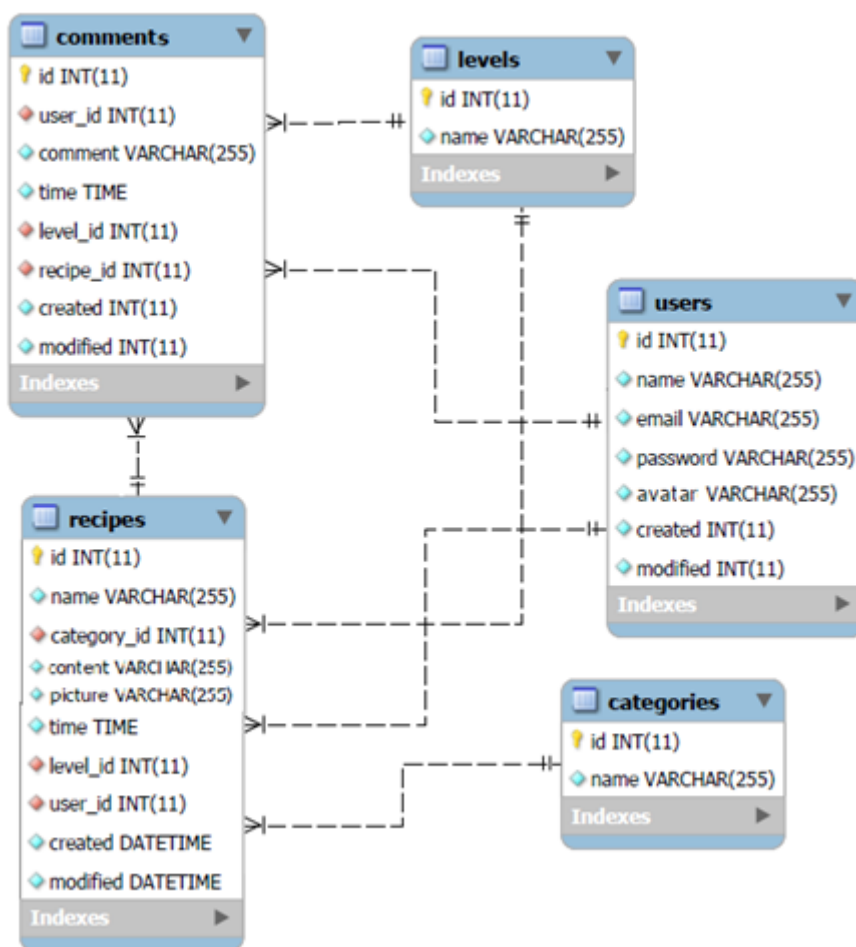
W celu łatwiejszego pisania i edycji kodu można skorzystać z różnych środowisk programistycznych. My korzystaliśmy z:

- phpDesigner 8, który jest potężnym środowiskiem programistycznym PHP, a także edytor HTML, CSS i JavaScript. Program dostarcza szereg ciekawych funkcji w tym również przydatnych programistom stawiającym pierwsze kroki w PHP - podpowiadanie kodu, auto-uzupełnienie, inteligentne podświetlanie i kolorowanie składni czy też zintegrowany manual. Oferuje wsparcie dla niemalże wszystkich frameworków PHP i najpopularniejszych JS (jQuery, Ext JS, YUI, Dojo, MooTools, Prototype), umożliwiając analizę i debugowanie tworzonych projektów (Xdebug), ich podgląd w popularnych przeglądarkach (m.in. IE, Firefox, Opera, Safari i Netscape), sprawdzanie składni kodu w czasie rzeczywistym i znacznie więcej. phpDesigner pozwala na pracę z plikami poprzez FTP/SFTP, obsługując TortoiseSVN.

Narzędzie to jest polecane na wielu forach internetowych oraz w wielu tutorialach. Jedynym minusem jest ograniczona czasowo licencja (21 dni darmowego użytkowania). Do pobrania pod adresem: <http://www.mpsoftware.dk/downloads.php>

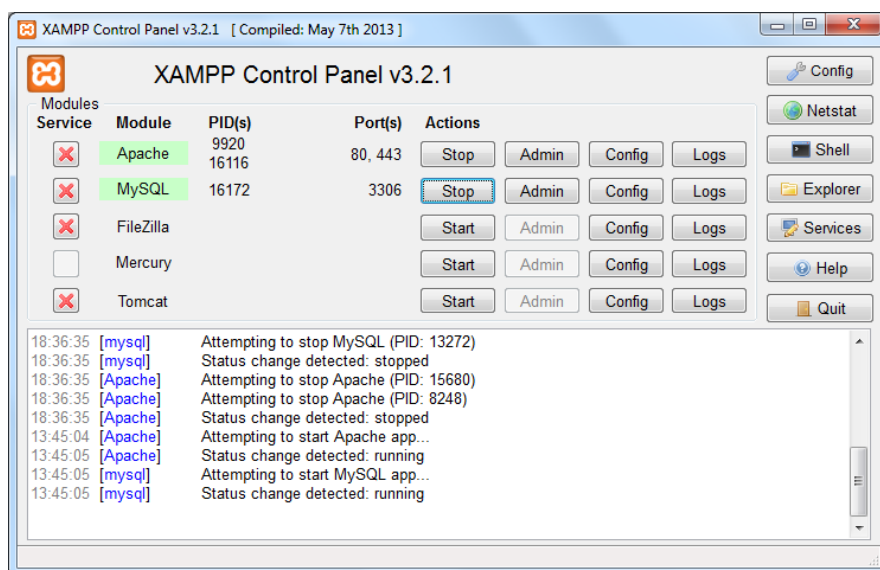
4. Rozpoczęcie pracy

Po zainstalowaniu programu XAMPP instalujemy (kopiujemy) pliki Cake do folder *xampp/htdocs* i przystępujemy do pracy, którą należy rozpocząć od stworzenia bazy danych. Należy zastanowić się jakie pola wystąpią w naszej bazie oraz jakie połączenia wystąpią między tabelami. Dobrą praktyką jest narysowanie diagramu ERD bazy danych. Ułatwi to stworzenie bazy na serwerze i pozwoli uniknąć pomyłek. Diagram ERD naszej bazy danych wygląda następująco:



Rys. 1 Diagram ERD gotowej bazy danych

Aby stworzyć bazę danych należy włączyć program XAMPP, a następnie kliknąć przyciski „Start” przy modułach Apache oraz MySQL. Pokazuje to rysunek drugi.



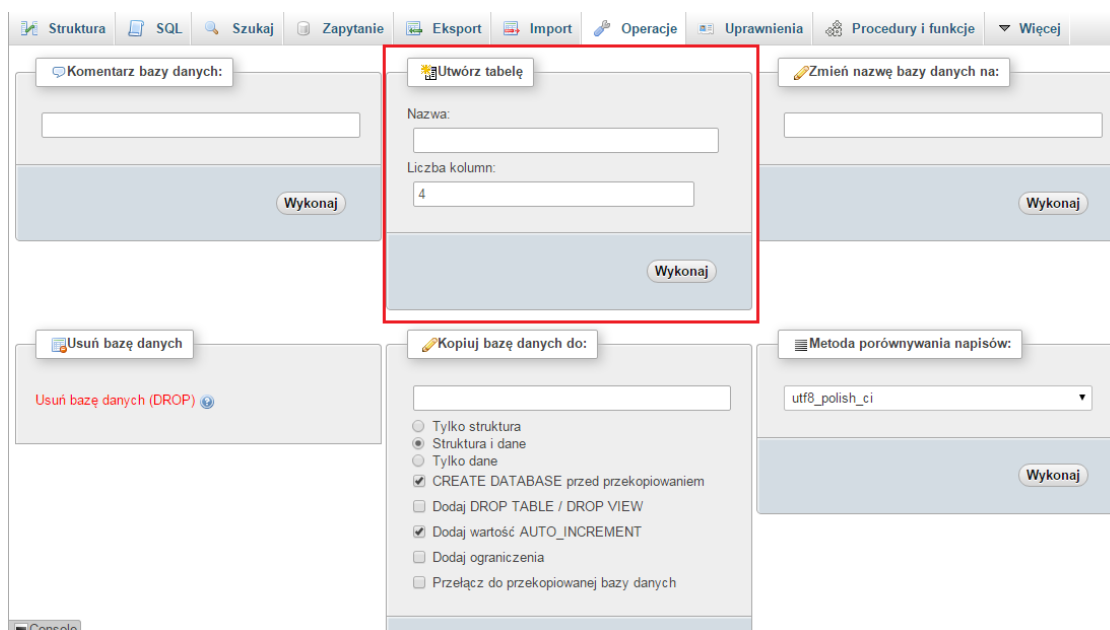
Rys. 2 Widok okna programu z włączonymi modułami

Po włączeniu modułów otwieramy okno przeglądarki i wpisujemy adres: **localhost/phpmyadmin** . Pojawi się okno służące do łatwego zarządzania bazą danych MySQL. W lewej części okna mamy widok w formie drzewa stworzonych baz danych oraz przycisk „Nowa” do utworzenia nowej bazy.



Rys. 3 Fragment ekranu prezentujący tworzenie nowej bazy danych

Klikamy na przycisk „Nowa” i wpisujemy nazwę nowej bazy danych (my nazwaliśmy bazę „recipes” oraz wybieramy z listy kodowanie „utf8_polish_ci”. Na końcu klikamy „Utwórz” i mamy stworzoną bazę danych, która jest pusta i należy w niej stworzyć odpowiednie tabele.



Rys. 4 Widok okna nowo utworzonej bazy danych

Nas najbardziej interesuje zaznaczona czerwonym prostokątem opcja „Utwórz tabelę”. Na podstawie diagramu ERD tworzymy pierwszą tabelę. I tutaj mamy pierwszą styczność z konwencjami nazewnictwa w CakePHP. Jeśli kontroler ma się nazywać Users, tabela w bazie danych powinna się nazywać tak samo, oczywiście nie musi, lecz zaoszczędzi nam to pisania zbędnych w tym momencie linii kodu. Natomiast model w tym wypadku musi nazywać się User, czyli liczba pojedyncza. Wpisujemy w pole nazwy „Users” i wybieramy liczbę kolumn 7. Uzupełniamy tabelę tak jak pokazano to na rysunku piątym.

Nazwa tabeli: Users Dodaj 1 kolumnę(y) Wykonaj

Nazwa	Typ	Długość/Wartości	Ustawienia domyślne	Metoda porównywania napisów	Atrybuty	Null	Indeks	A_I
id	INT	11	Brak			<input checked="" type="checkbox"/>	PRIMARY	<input checked="" type="checkbox"/>
name	VARCHAR	255	Brak			<input type="checkbox"/>	---	<input type="checkbox"/>
email	VARCHAR	255	Brak			<input type="checkbox"/>	---	<input type="checkbox"/>
password	VARCHAR	255	Brak			<input type="checkbox"/>	---	<input type="checkbox"/>
avatar	VARCHAR	255	Brak			<input type="checkbox"/>	---	<input type="checkbox"/>
created	DATETIME		Brak			<input type="checkbox"/>	---	<input type="checkbox"/>
modified	DATETIME		Brak			<input type="checkbox"/>	---	<input type="checkbox"/>

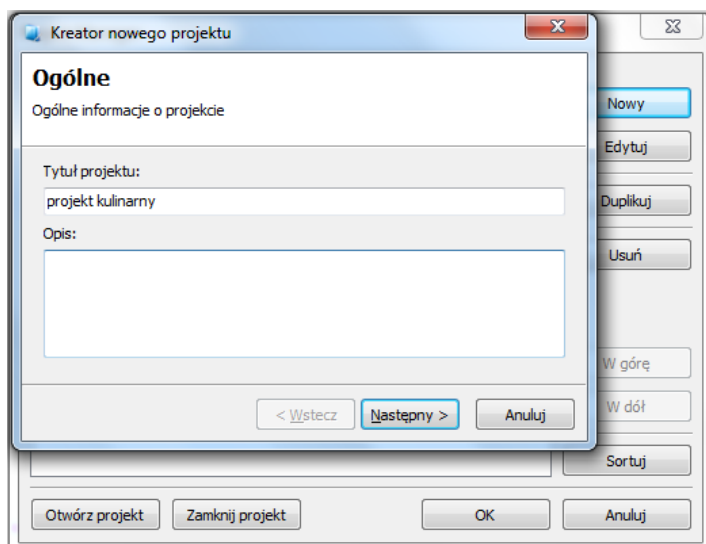
Komentarze tabeli: Silnik składowania: InnoDB Sortowanie:

Rys. 5 Dane wprowadzonego tabeli Users

Przy polu „id” wybieramy dodatkowe opcje: Indeks PRIMARY, aby to pole było naszym kluczem głównym oraz zaznaczamy opcję A_I czyli Auto Increment (tak samo postępujemy z każdą tabelą i polem „id”). Na końcu klikamy „Zapisz” i mamy stworzoną pierwszą tabelę w naszej bazie danych. W ten sam sposób postępujemy z kolejnymi tabelami, które tworzymy na podstawie wcześniej przygotowanego diagramu ERD. Pola Created i Modified są domyślnymi polami CakePHP, które będą zmieniane w momencie tworzenia nowego wiersza oraz jego modyfikacji. Nie musimy o tym w ogóle pamiętać, ponieważ Cake zrobi to za nas automatycznie.

5. Stworzenie nowego projektu w phpDesigner i połączenie go z bazą danych

Otwieramy program phpDesigner i tworzymy nowy projekt. W zakładce *Projekt* wybieramy zakładkę *Menadżer projektu*. W otwartym oknie klikamy na przycisk *Nowy* i przechodzimy przez Kreator nowego projektu. W pierwszym oknie (rysunek 6) wpisujemy tytuł oraz opis nowego projektu.



Rys. 6 Kreator tworzenia nowego projektu

W następnym oknie kreatora wybieramy ścieżkę główną projektu czyli miejsce, gdzie skopiowaliśmy pliki Cake (xampp/htdocs/app). W kolejnych oknach kreatora klikamy *Dalej* by na końcu kliknąć *Koniec*.

W ten sposób stworzyliśmy nowy projekt i możemy przystąpić do właściwej pracy. W oknie przeglądarki wpisujemy **localhost** i patrzymy na rezultaty. Naszym oczom ukaże się strona jak poniżej. Widzimy, że Cake pomaga nam wypisując na stronie dużo pomocnych informacji. Na żółtym pasku widzimy informację, że Cake nie może się połączyć z bazą danych i podpowiada nam, który plik należy zmodyfikować.



Rys. 7 Widok strony startowej Cake

Pierwszym zadaniem jest połączenie wcześniej stworzonej bazy danych z nowym projektem. W tym celu należy w drzewie projektu przejść do ścieżki **/app/Config/database.php** i skonfigurować plik tak jak na rysunku ósmym.

```

69 class DATABASE_CONFIG {
70
71     public $default = array(
72         'datasource' => 'Database/Mysql',
73         'persistent' => false,
74         'host' => 'localhost',
75         'login' => 'root',
76         'password' => '',
77         'database' => 'recipes',
78         'prefix' => '',
79         'encoding' => 'utf8',
80     );
81
82     public $test = array(
83         'datasource' => 'Database/Mysql',
84         'persistent' => false,
85         'host' => 'localhost',
86         'login' => 'user',
87         'password' => 'password',
88         'database' => 'test_database_name',
89         'prefix' => '',
90         //'encoding' => 'utf8',
91     );
92 }

```

Rys. 8 Poprawna konfiguracja pliku database.php

Po wprowadzeniu zmian zapisujemy projekt i odświeżamy stronę w przeglądarki. Naszym oczom powinno się ukazać okno jak poniżej.



CakeFest 2012:
The CakePHP Conference
Manchester, UK

August 30-September 2
2 days of Workshops
2 days of Conference
== 4 days of AWESOME

Release Notes for CakePHP 2.2.3.

[Read the changelog](#)

Your version of PHP is 5.2.8 or higher.

Your tmp directory is writable.

The FileEngine is being used for core caching. To change the config edit APP/Config/core.php

Your database configuration file is present.

Cake is able to connect to the database.

Editing this Page

To change the content of this page, edit: APP/View/Pages/home.ctp.
To change its layout, edit: APP/View/Layouts/default.ctp.
You can also add some CSS styles for your pages at: APP/webroot/css.

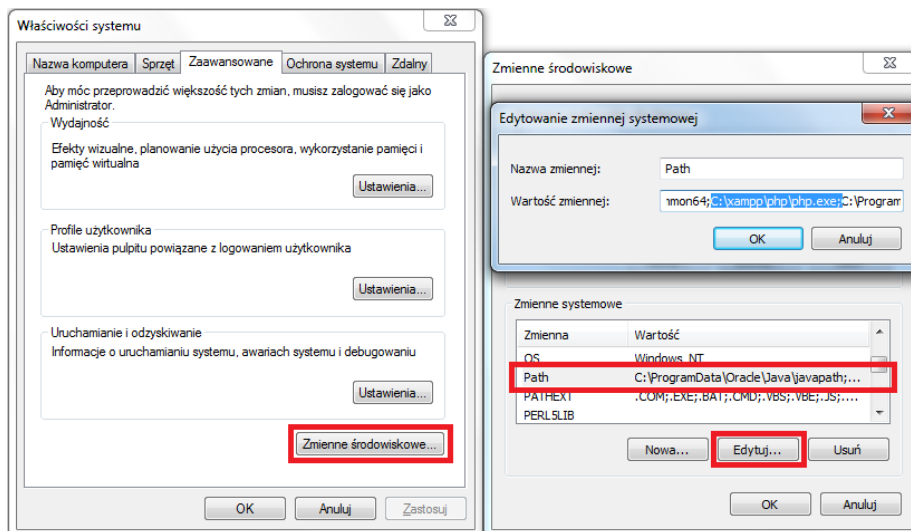
Getting Started

Rys. 9 Cake połączony z bazą danych i gotowy do właściwego pisania kodu

6. Wykorzystanie właściwości frameworka do tworzenia kodu projektu

Możemy przystąpić do tworzenia kodu. Cake udostępnia bardzo przyjemną funkcję, dzięki której kod jest generowany automatycznie.

Konsola CakePHP- Bake sprawia, że praca z frameworkiem jest łatwa i szybka. Konsola umożliwia nie tylko tworzenie szkieletów podstawowych składników: modeli, kontrolerów i widoków, ale pozwala na tworzenie pełni funkcjonalnych aplikacji i to w zaledwie kilka minut. Należy się tylko do tego odpowiednio przygotować. Naciskamy klawisz **Windows+R** i wpisujemy komendę **sysdm.cpl** . Przechodzimy do zakładki *Zaawansowane* i klikamy przycisk *Zmienne środowiskowe*. Tam do zmiennej *Path* dodajemy następującą treść: **C:\xampp\php\php.exe;**. Ważne żeby ciąg znaków był łączny, czyli bez spacji i innych separatorów.



Rys. 10 Dodanie zmiennej środowiskowej

Po zapisaniu zmian może być wymagany restart komputera. Przystępujemy do generowania kodu. Otwieramy linie poleceń i przechodzimy do lokalizacji naszej aplikacji czyli:

C:\xampp\htdocs a następnie wpisujemy komendę: **C:\xampp\php\php.exe app\Console\cake.php bake** .

The image is a screenshot of a Windows command prompt window. The title bar shows the command being executed: 'Administrator: C:\Windows\system32\cmd.exe - c:\xampp\php\php.exe app\Console\cake.php ba...'. The command prompt shows the following sequence of commands and output:

```
C:\xampp\htdocs>cd htdocs
System nie może odnaleźć określonej ścieżki.

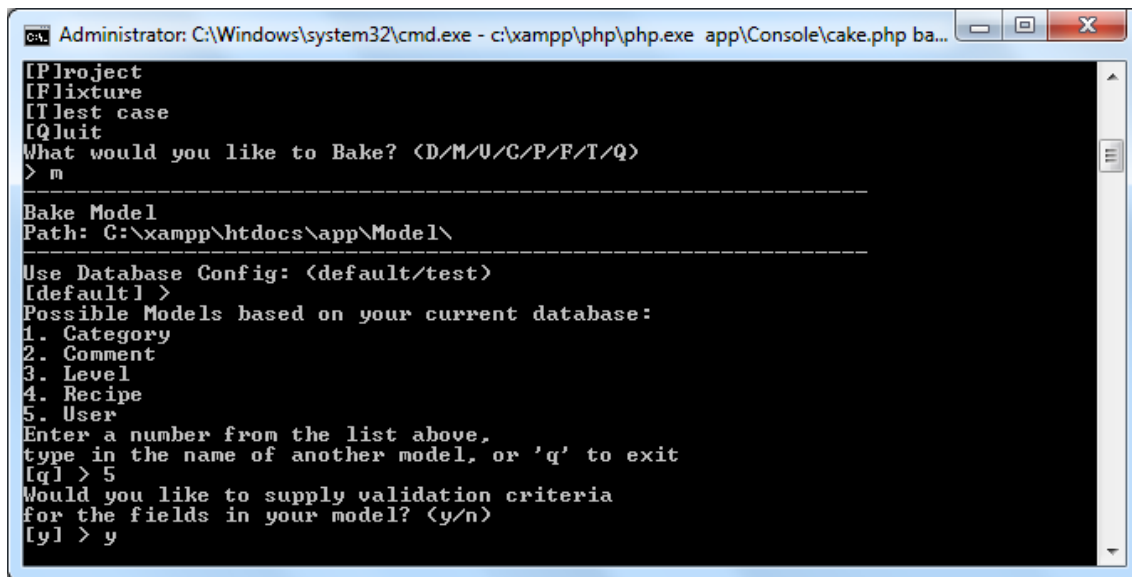
C:\xampp\htdocs>cd C:\xampp\htdocs

C:\xampp\htdocs>c:\xampp\php\php.exe app\Console\cake.php bake

Welcome to CakePHP v2.6.3 Console
-----
App : app
Path: C:\xampp\htdocs\app\
-----
Interactive Bake Shell
-----
[Database Configuration]
[M]odel
[U]iew
[C]ontroller
[P]roject
[F]ixture
[T]est case
[Q]uit
What would you like to Bake? <D/M/U/C/P/F/T/Q>
>
```

Rys. 11 Widok aplikacji konsolowej CakePHP

Pokażemy jak stworzyć plik modelu, plik kontrolera oraz plik widoku dla tabeli Users. Zaczynamy od stworzenia modelu. Klikamy literkę *m* i potwierdzamy *Enter*. Program pyta której konfiguracji bazy danych użyć. Wybieramy domyślną i potwierdzamy. Konsola Bake sprawdza, co znajduje się w naszej bazie danych i pyta dla jakiej tabeli ma wygenerować model. Należy zauważyć, że nazwy te są rzeczownikami w liczbie pojedynczej z nazw, jakie nadaliśmy tabelom przy tworzeniu bazy danych. Nadawanie odpowiednich nazw tabelom w bazie danych jest więc bardzo istotne. Potrzebujemy model dla tabeli *Users* więc naciskamy 5 i zatwierdzamy klawiszem *Enter* (rys. 12).



```
Administrator: C:\Windows\system32\cmd.exe - c:\xampp\php\php.exe app\Console\cake.php ba...
[P]roject
[F]ixture
[T]est case
[Q]uit
What would you like to Bake? <D/M/U/C/P/F/T/Q>
> m

-----
Bake Model
Path: C:\xampp\htdocs\app\Model\
-----
Use Database Config: <default/test>
[default] >
Possible Models based on your current database:
1. Category
2. Comment
3. Level
4. Recipe
5. User
Enter a number from the list above,
type in the name of another model, or 'q' to exit
[q] > 5
Would you like to supply validation criteria
for the fields in your model? <y/n>
[y] > y
```

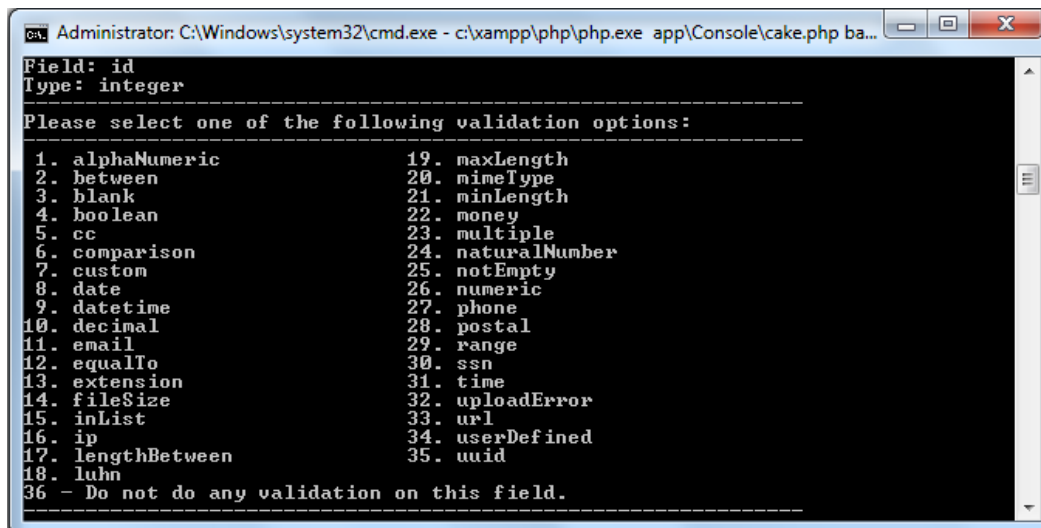
Rys. 12 Wybór tabeli do stworzenia modelu i pytanie o walidację

Kolejnym pytaniem przybliżającym nas do stworzenia modelu jest pytanie o walidację dla pól z naszej tabeli.

Walidacja danych jest to sprawdzanie poprawności wprowadzanych danych do systemu.

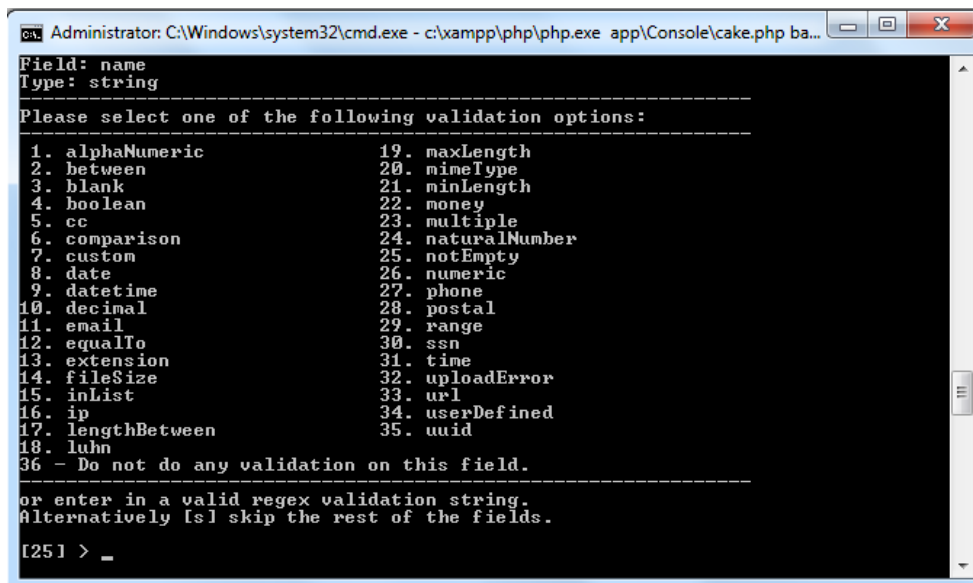
Na przykład, gdy chcemy, aby użytkownik wprowadził swój numer telefonu to aplikacja musi zapewnić, aby użytkownik nie mógł wprowadzić błędnych danych (np. tekstu) lub, aby aplikacja informowała, że użytkownik wprowadził błędne dane i żeby je poprawił tak, aby system mógł te dane zaakceptować.

Pierwszym polem jest pole *id* typu *integer*. Dla tego pola nie chcemy ustalać żadnych reguł walidacyjnych dlatego naciskamy 36 i zatwierdzamy.



Rys. 13 Lista dostępnych reguł walidacyjnych

Kolejnym polem w tej tabeli jest pole *name* typu *string*. Chcemy, żeby to pole nie mogło być puste (każdy użytkownik musi mieć jakąś nazwę). Domyślnie program podpowiada nam, żeby wybrać opcję 25- pole nie może być puste.



Rys. 14 Wybór reguł walidacyjnych dla pola *name*

Kolejnym polem jest *email*. Dla tego pola chcemy określić więcej niż jedną regułę. Pole to ma być polem typu email (wymagany znak @) i oczywiście nie może być puste. Wybieramy opcje 11 oraz 25.

```
Administrator: C:\Windows\system32\cmd.exe - c:\xampp\php\php.exe app\Console\cake.php ba...
[11] > 11
Would you like to add another validation rule
or skip the rest of the fields? (y/n/s)
[1] > y

Field: email
Type: string

-----
Please select one of the following validation options:
-----
1. alphaNumeric          19. maxLength
2. between               20. mimeType
3. blank                 21. minLength
4. boolean               22. money
5. cc                    23. multiple
6. comparison            24. naturalNumber
7. custom                25. notEmpty
8. date                  26. numeric
9. datetime              27. phone
10. decimal              28. postal
11. email                29. range
12. equalTo              30. ssn
13. extension            31. time
14. fileSize             32. uploadError
15. inList               33. url
16. ip                   34. userDefined
17. lengthBetween        35. uuid
18. luhn
36 - Do not do any validation on this field.
-----
or enter in a valid regex validation string.
Alternatively [s] skip the rest of the fields.
[11] > 25
Would you like to add another validation rule
or skip the rest of the fields? (y/n/s)
```

Rys. 15 Wybór reguł dla pola email

Dla pola *password* ustanawiamy reguły o numerach 21 oraz 25. Dla pola *avatar*, które przechowywać będzie zdjęcie profilu użytkownika wybieramy regułę o numerze 32.

```
Administrator: C:\Windows\system32\cmd.exe - c:\xampp\php\php.exe app\Console\cake.php ba...
Would you like to add another validation rule
or skip the rest of the fields? (y/n/s)
[1] > n

Field: password
Type: string

-----
Please select one of the following validation options:
-----
1. alphaNumeric          19. maxLength
2. between               20. mimeType
3. blank                 21. minLength
4. boolean               22. money
5. cc                    23. multiple
6. comparison            24. naturalNumber
7. custom                25. notEmpty
8. date                  26. numeric
9. datetime              27. phone
10. decimal              28. postal
11. email                29. range
12. equalTo              30. ssn
13. extension            31. time
14. fileSize             32. uploadError
15. inList               33. url
16. ip                   34. userDefined
17. lengthBetween        35. uuid
18. luhn
36 - Do not do any validation on this field.
-----
or enter in a valid regex validation string.
Alternatively [s] skip the rest of the fields.
[25] >
Would you like to add another validation rule
or skip the rest of the fields? (y/n/s)
[1] > y
```

Rys. 16 Wybór reguł walidacyjnych dla pola *password*

Po przejściu wszystkich pól i stworzeniu dla nich odpowiednich reguł program pyta czy ma stworzyć jakieś reguły asocjacyjne. Tworzenie takich reguł jest bardzo proste, bo wprowadzamy je do programu w sposób jednoznaczny. Użytkownik może dodawać wiele komentarzy oraz wiele przepisów. Jeżeli chcemy zdefiniować swoje reguły asocjacyjne możemy to zrobić w tym momencie (rys. 17). W naszym przypadku nie jest to potrzebne i przechodzimy dalej.

```
Administrator: C:\Windows\system32\cmd.exe - c:\xampp\php\php.exe app\Console\cake.php ba...
4. boolean                22. money
5. cc                     23. multiple
6. comparison             24. naturalNumber
7. custom                 25. notEmpty
8. date                   26. numeric
9. datetime               27. phone
10. decimal               28. postal
11. email                 29. range
12. equalTo               30. ssn
13. extension             31. time
14. fileSize              32. uploadError
15. inList                33. url
16. ip                    34. userDefined
17. lengthBetween         35. uuid
18. luhn
36 - Do not do any validation on this field.
-----
or enter in a valid regex validation string.
Alternatively [s] skip the rest of the fields.
[11] > s
Would you like to define model associations
(hasMany, hasOne, belongsTo, etc.)? <y/n>
[y] > y
One moment while the associations are detected.
-----
Please confirm the following associations:
-----
User hasMany Comment? <y/n>
[y] > y
User hasMany Recipe? <y/n>
[y] > y
Would you like to define some additional model associations? <y/n>
[n] > n
-----
```

Rys. 17 Definiowanie reguł asocjacyjnych

Program podsumowuje naszą pracę i prezentuje model, który chce dla nas stworzyć (rys. 18). Jeżeli nie mamy zastrzeżeń zatwierdzamy stworzenie modelu.

```
Administrator: C:\Windows\system32\cmd.exe - c:\xampp\php\php.exe app\Console\cake.php ba...
The following Model will be created:
-----
Name: User
DB Table: 'recipes'. 'users'
Validation: Array
<
  [name] => Array
  <
    [notEmpty] => notEmpty
  >
  [email] => Array
  <
    [email] => email
    [notEmpty] => notEmpty
  >
  [password] => Array
  <
    [notEmpty] => notEmpty
    [minLength] => minLength
  >
  [avatar] => Array
  <
    [uploadError] => uploadError
  >
>
Associations:
  User hasMany Comment
  User hasMany Recipe
-----
Look okay? <y/n>
[y] > y_
```

Rys. 18 Podsumowanie tworzenia modelu

Program stworzył dla nas model. Nie jest on zbyt skomplikowany, ale dzięki kilku kliknięciom oszczędziliśmy wielu linii pisania dosyć monotonnego kodu. Cake wygenerował nam 118 linii reguł walidacyjnych i asocjacyjnych. To oczywiście nie jest gotowy model i będziemy do niego dodawać własne metody, jednak jest bardzo dobrą podstawą do dalszej pracy. Gdy coś w naszej bazie, regułach, strukturze ulegnie zmianie możemy ponownie uruchomić program w konsoli i stworzyć nowy model, który nadpisze stary i nieaktualny.

```

1 <?php
2 App::uses('AppModel', 'Model');
3
4 * User Model
5
6 * Property Comment Comment
7 * Property Recipe Recipe
8
9
10 class User extends AppModel {
11
12 * Display field
13
14 * $var string
15
16 public $displayField = 'name';
17
18
19 * Validation rules
20
21 * $var array
22
23 public $validate = array(
24     'name' => array(
25         'notEmpty' => array(
26             'message' => 'Your custom :
27             'required' => false,
28             'last' => false, // Stop
29             'on' => 'create', // Limit
30         ),
31     ),
32     'email' => array(
33         'email' => array(
34             'rule' => array('email'),
35             'message' => 'Your custom :
36             'required' => false,
37             'last' => false, // Stop validation after this rule
38             'on' => 'create', // Limit validation to 'create' or 'update' operations
39         ),
40         'notEmpty' => array(
41             'rule' => array('notEmpty'),
42             'message' => 'Your custom message here',
43             'required' => false,
44             'last' => false, // Stop validation after this rule
45             'on' => 'create', // Limit validation to 'create' or 'update' operations
46         ),
47     ),
48     'password' => array(
49         'notEmpty' => array(
50             'rule' => array('notEmpty'),
51             'message' => 'Your custom message here',
52             'required' => false,
53             'last' => false, // Stop validation after this rule
54             'on' => 'create', // Limit validation to 'create' or 'update' operations
55         ),
56         'minLength' => array(
57             'rule' => array('minLength'),
58             'message' => 'Your custom message here',
59             'required' => false,
60             'last' => false, // Stop validation after this rule
61             'on' => 'create', // Limit validation to 'create' or 'update' operations
62         ),
63     ),
64     'avatar' => array(
65         'uploadError' => array(
66             'rule' => array('uploadError'),
67             'message' => 'Your custom message here',
68             'required' => false,
69             'last' => false, // Stop validation after this rule
70             'on' => 'create', // Limit validation to 'create' or 'update' operations
71         ),
72     ),
73 );
74
75
76
77
78
79
80

```

Rys. 19 Kod z pliku modelu User.php

Po stworzeniu modelu dla tabeli *User* przechodzimy do tworzenia kontrolera. Wybieramy literkę *c* i zatwierdzamy jak uprzednio klawiszem *Enter*. Konfiguracja bazy danych domyślna (default). Wybieramy numer 5 żeby móc wygenerować kontroler dla tabeli *Users*.

```
Administrator: C:\Windows\system32\cmd.exe - c:\xampp\php\php.exe app\Console\cake.php ba...
[1] > y
Wrote C:\xampp\htdocs\app\Test\Case\Model\UserTest.php

Interactive Bake Shell

-----
[1] Database Configuration
[1] Model
[1] View
[1] Controller
[1] Project
[1] Fixture
[1] Test case
[1] Quit
What would you like to Bake? <D/M/U/C/P/F/T/Q>
> c

-----
Bake Controller
Path: C:\xampp\htdocs\app\Controller\

-----
Use Database Config: <default/test>
[default] >
Possible Controllers based on your current database:

-----
1. Categories
2. Comments
3. Levels
4. Recipes
5. Users

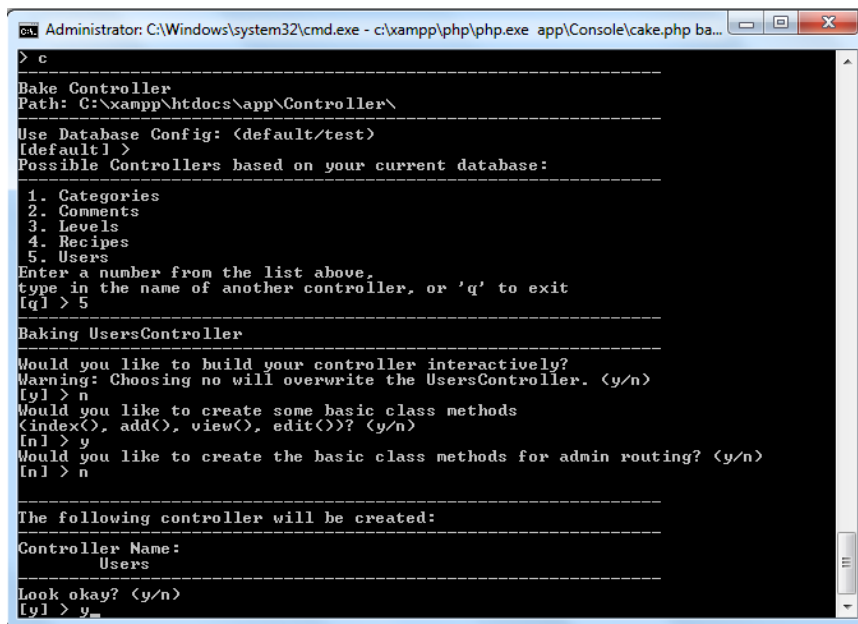
Enter a number from the list above,
type in the name of another controller, or 'q' to exit
[q] > 5

-----
Baking UsersController

-----
Would you like to build your controller interactively?
Warning: Choosing no will overwrite the UsersController. <y/n>
[y] > n
```

Rys. 20 Tworzenie kontrolera- początek

W ostatnim pytaniu na rys. 20 wybieramy opcję *no*. Zapobiegnie to stworzeniu metod scaffolding'owych nie dających się edytować. Metody te stworzymy sami w następnym kroku.



```
Administrator: C:\Windows\system32\cmd.exe - c:\xampp\php\php.exe app\Console\cake.php ba...
> c
-----
Bake Controller
Path: C:\xampp\htdocs\app\Controller\
-----
Use Database Config: <default/test>
[default] >
Possible Controllers based on your current database:
-----
1. Categories
2. Comments
3. Levels
4. Recipes
5. Users
Enter a number from the list above,
type in the name of another controller, or 'q' to exit
[q] > 5
-----
Baking UsersController
-----
Would you like to build your controller interactively?
Warning: Choosing no will overwrite the UsersController. <y/n>
[y] > n
Would you like to create some basic class methods
(index(), add(), view(), edit())? <y/n>
[n] > y
Would you like to create the basic class methods for admin routing? <y/n>
[n] > n
-----
The following controller will be created:
-----
Controller Name:
    Users
-----
Look okay? <y/n>
[y] > y_
```

Rys. 21 Tworzenie kontrolera- ciąg dalszy

Potwierdzamy chęć stworzenia podstawowych metod w klasie (index, add, view, edit) oraz wybieramy *no* dla ostatniej opcji stworzenia metod routing. Na końcu program pyta czy wygląda ok. Jeżeli tak to zatwierdzamy *Enterem* i mamy stworzony kontroler *Users*.

```

1 <?php
2 App::uses('AppController', 'Controller');
3 /**
4  * Users Controller
5  *
6  * @property User $User
7  * @property PaginatorComponent $Paginator
8  */
9 class UsersController extends AppController {
10
11 /**
12  * Components
13  *
14  * @var array
15  */
16 public $components = array('Paginator');
17
18 /**
19  * index method
20  *
21  * @return void
22  */
23 public function index() {
24     $this->User->recursive = 0;
25     $this->set('users', $this->Paginator->paginate());
26 }
27
28 /**
29  * view method
30  *
31  * @throws NotFoundException
32  * @param string $id
33  * @return void
34  */
35 public function view($id = null) {
36     if (!$this->User->exists($id)) {
37         throw new NotFoundException(__('Invalid user'));
38     }
39     $options = array('conditions' => array('User.' . $this->User->primaryKey => $id));
40     $this->set('user', $this->User->find('first', $options));
41 }
42
43 /**
44  * add method
45  *
46  * @return void
47  */
48 public function add() {
49     if ($this->request->is('post')) {
50         $this->User->create();
51         if ($this->User->save($this->request->data)) {
52             $this->Session->setFlash(__('The user has been saved.'));
53             return $this->redirect(array('action' => 'index'));
54         } else {
55             $this->Session->setFlash(__('The user could not be saved. Please, try again.'));
56         }
57     }
58 }
59
60 /**
61  * edit method
62  *
63  * @throws NotFoundException
64  * @param string $id
65  * @return void
66  */
67 public function edit($id = null) {
68     if (!$this->User->exists($id)) {
69         throw new NotFoundException(__('Invalid user'));
70     }
71     if ($this->request->is(array('post', 'put'))) {
72         if ($this->User->save($this->request->data)) {
73             $this->Session->setFlash(__('The user has been saved.'));
74             return $this->redirect(array('action' => 'index'));
75         } else {
76             $this->Session->setFlash(__('The user could not be saved. Please, try again.'));
77         }
78     } else {
79         $options = array('conditions' => array('User.' . $this->User->primaryKey => $id));
80         $this->request->data = $this->User->find('first', $options);
81     }
82 }
83
84 /**
85  * delete method
86  *
87  * @throws NotFoundException
88  * @param string $id
89  * @return void
90  */
91 public function delete($id = null) {
92     $this->User->id = $id;
93     if (!$this->User->exists()) {
94         throw new NotFoundException(__('Invalid user'));
95     }
96     $this->request->allowMethod('post', 'delete');
97     if ($this->User->delete()) {
98         $this->Session->setFlash(__('The user has been deleted.'));
99     } else {
100         $this->Session->setFlash(__('The user could not be deleted. Please, try again.'));
101     }
102     return $this->redirect(array('action' => 'index'));
103 }
104 }

```

Rys. 22 Gotowe metody wygenerowane automatycznie przez program Bake

Tworzenie widoków jest tak samo proste jak tworzenie modeli czy kontrolerów. W menu programu deklarujemy chęć stworzenia widoku (literka v), wybór kontrolera *Users* (numer 5) i odmowa tworzenia widoków interaktywnych. Cały proces przedstawia rysunek 23.


```
Administrator: C:\Windows\system32\cmd.exe - c:\xampp\php\php.exe app\Console\cake.php ba...
[Q]uit
What would you like to Bake? <D/M/U/C/P/F/T/Q>
> u

-----
Bake View
Path: C:\xampp\htdocs\app\View\
-----
Use Database Config: <default/test>
[default] >
Possible Controllers based on your current database:
-----
1. Categories
2. Comments
3. Levels
4. Recipes
5. Users
Enter a number from the list above,
type in the name of another controller, or 'q' to exit
[q] > 5
Would you like bake to build your views interactively?
Warning: Choosing no will overwrite Users views if they exist. <y/n>
[n] > n

Baking `index` view file...

Creating file C:\xampp\htdocs\app\View\Users\index.ctp
Wrote 'C:\xampp\htdocs\app\View\Users\index.ctp'

Baking `view` view file...

Creating file C:\xampp\htdocs\app\View\Users\view.ctp
Wrote 'C:\xampp\htdocs\app\View\Users\view.ctp'

Baking `add` view file...

Creating file C:\xampp\htdocs\app\View\Users\add.ctp
Wrote 'C:\xampp\htdocs\app\View\Users\add.ctp'

Baking `edit` view file...

Creating file C:\xampp\htdocs\app\View\Users\edit.ctp
Wrote 'C:\xampp\htdocs\app\View\Users\edit.ctp'

-----
View Scaffolding Complete.
```

Rys. 23 Okno tworzenia pliku widoku dla kontrolera *Users*

```
Code Debug Run Localhost PHP + XHTML + CSS + JavaScript
1 <div class="users form">
2 <?php echo $this->Form->create('User'); ?>
3 <fieldset>
4 <legend><?php echo __('Edit User'); ?></legend>
5 <?php
6 echo $this->Form->input('id');
7 echo $this->Form->input('name');
8 echo $this->Form->input('email');
9 echo $this->Form->input('password');
10 echo $this->Form->input('avatar');
11 ?>
12 </fieldset>
13 <?php echo $this->Form->end(__('Submit')); ?>
14 </div>
15 <div class="actions">
16 <h3><?php echo __('Actions'); ?></h3>
17 <ul>
18
19 <li><?php echo $this->Form->postLink(__('Delete'), array('action' => 'delete', $this->Form->value('User.id')), array(), __('Are you sure you want to d
20 <li><?php echo $this->Html->link(__('List Users'), array('action' => 'index')); ?></li>
21 <li><?php echo $this->Html->link(__('List Comments'), array('controller' => 'comments', 'action' => 'index')); ?> </li>
22 <li><?php echo $this->Html->link(__('New Comment'), array('controller' => 'comments', 'action' => 'add')); ?> </li>
23 <li><?php echo $this->Html->link(__('List Recipes'), array('controller' => 'recipes', 'action' => 'index')); ?> </li>
24 <li><?php echo $this->Html->link(__('New Recipe'), array('controller' => 'recipes', 'action' => 'add')); ?> </li>
25 </ul>
26 </div>
27
```

Rys. 24 Wygenerowany automatycznie widok dla metody *add (User)*

Otrzymane pliki są jedynie zaczątkami finalnych wersji plików. Dla samej tabeli *Users* stworzyliśmy i dodali następujące metody do kontrolera: *registration*, *login*, *logout* oraz *edit* odpowiedzialne za tworzenie użytkownika, jego logowanie i wylogowanie z serwisu. Cały projekt dostępny jest w repozytorium GitHub oraz na stronie: <http://mariusz.kgtech.pl/>

