Creating web applications with CodeIgniter

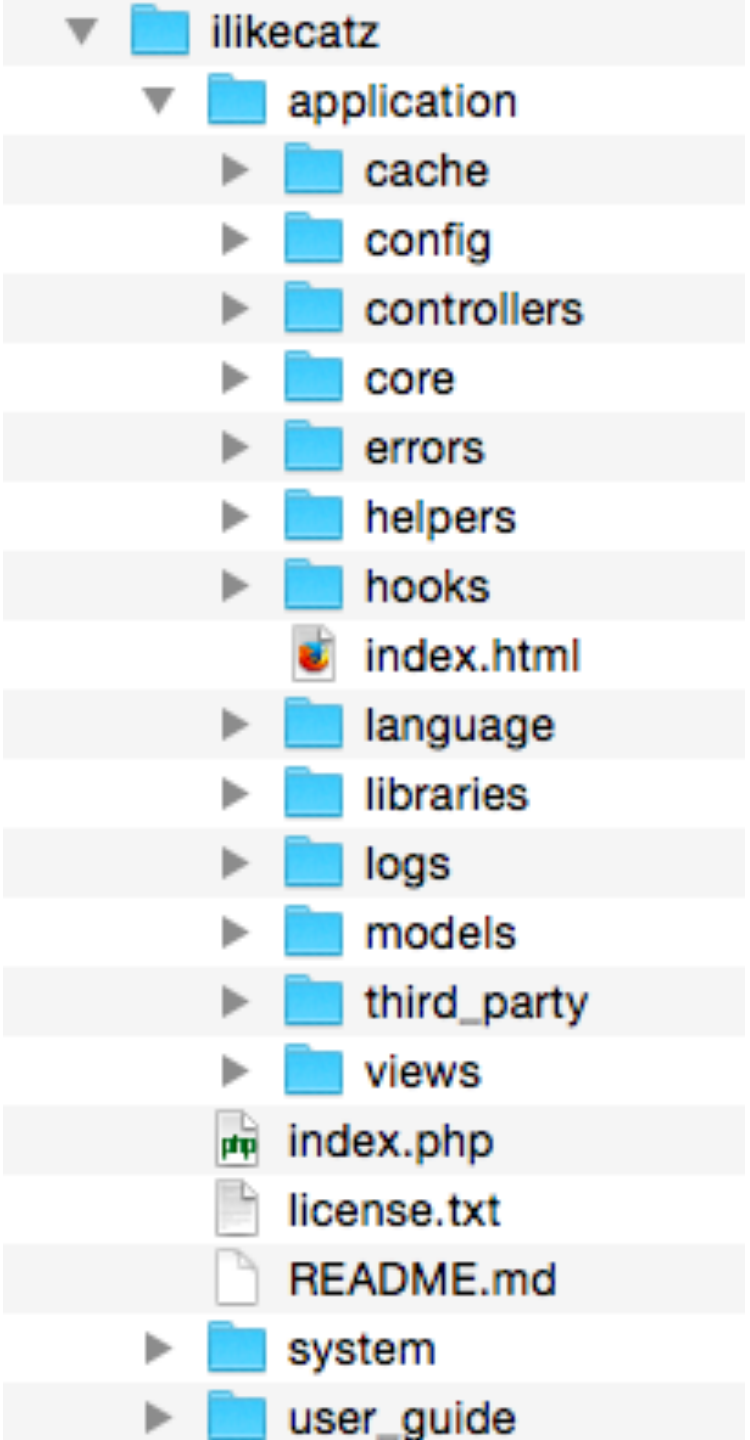
Aims for today

- Go through design and development of simple web application using CodeIgniter
  - Demonstrate how to pass data from form
  - How to create model
  - How to pass data from model to controller and view
- Databases

CodeIgniter – review from last week

- Web framework for php
- Installing CodeIgniter creates a barebones application – you add your code to customize it into your application
- Uses patterns
  - Mvc
  - Front controller
- MVC affects how you write the web application
  - Partition code into difference components (models, views etc)
- Front controller affects how you run the application
  - Single point of entry

CodeIgniter's directory structure



Running CodeIgniter

https://courtes.users.ecs.westminster.ac.uk/ci/index.php/Welcome/index

Sample web application

- Who am I?

Kanye West or Simon Courtenage?

Ask user to pick… and then tell them if they are right or wrong

To do this, we need to pass data from the browser (the user's choice) and make decisions about what content to generate

Where do we start?

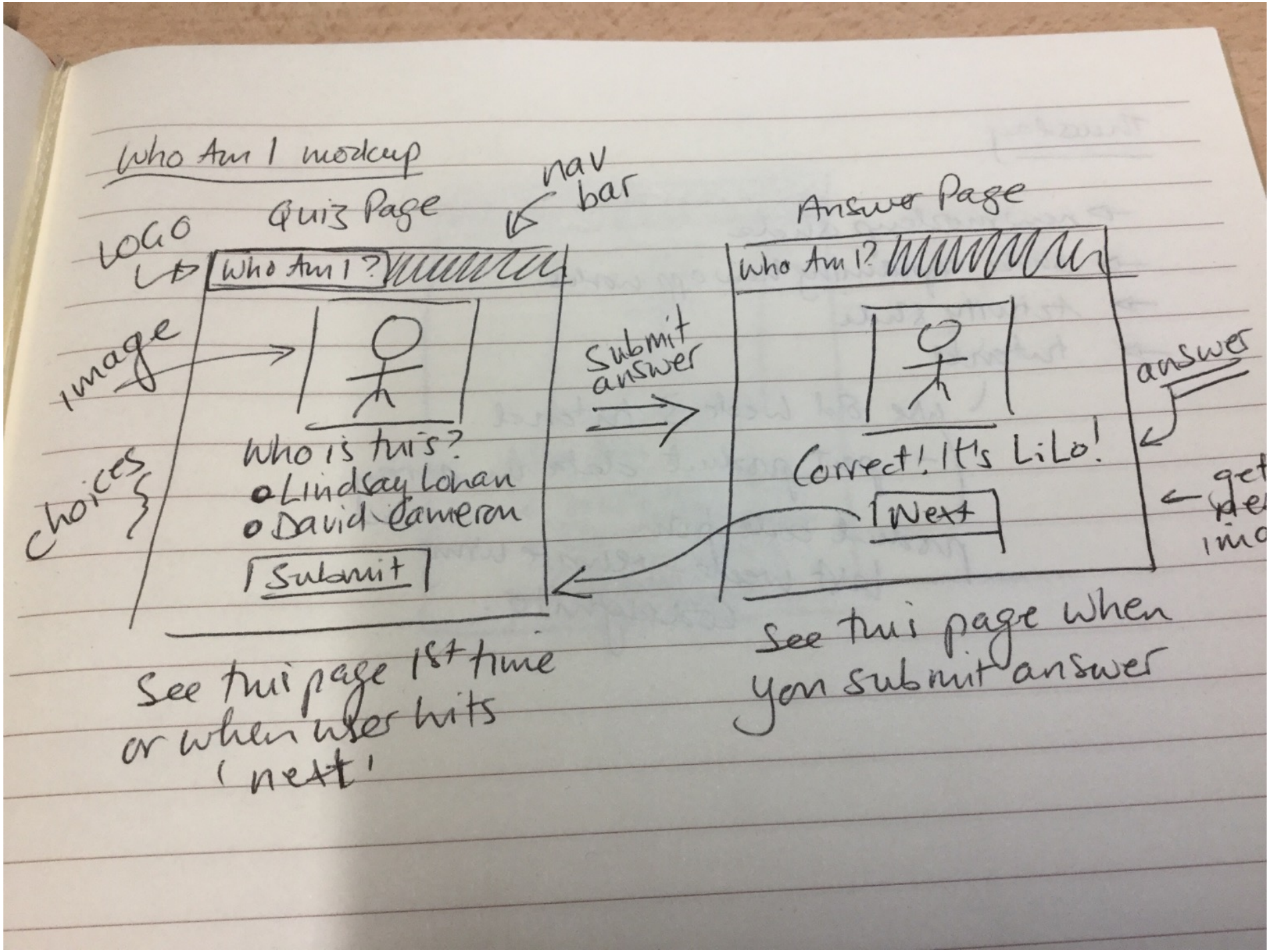
- Mockups!
- Mockups of web pages help visualise requirements
- Interface-driven design

My mockup for WhoAmI

Mockups sketch what the user sees – bit like a use case model

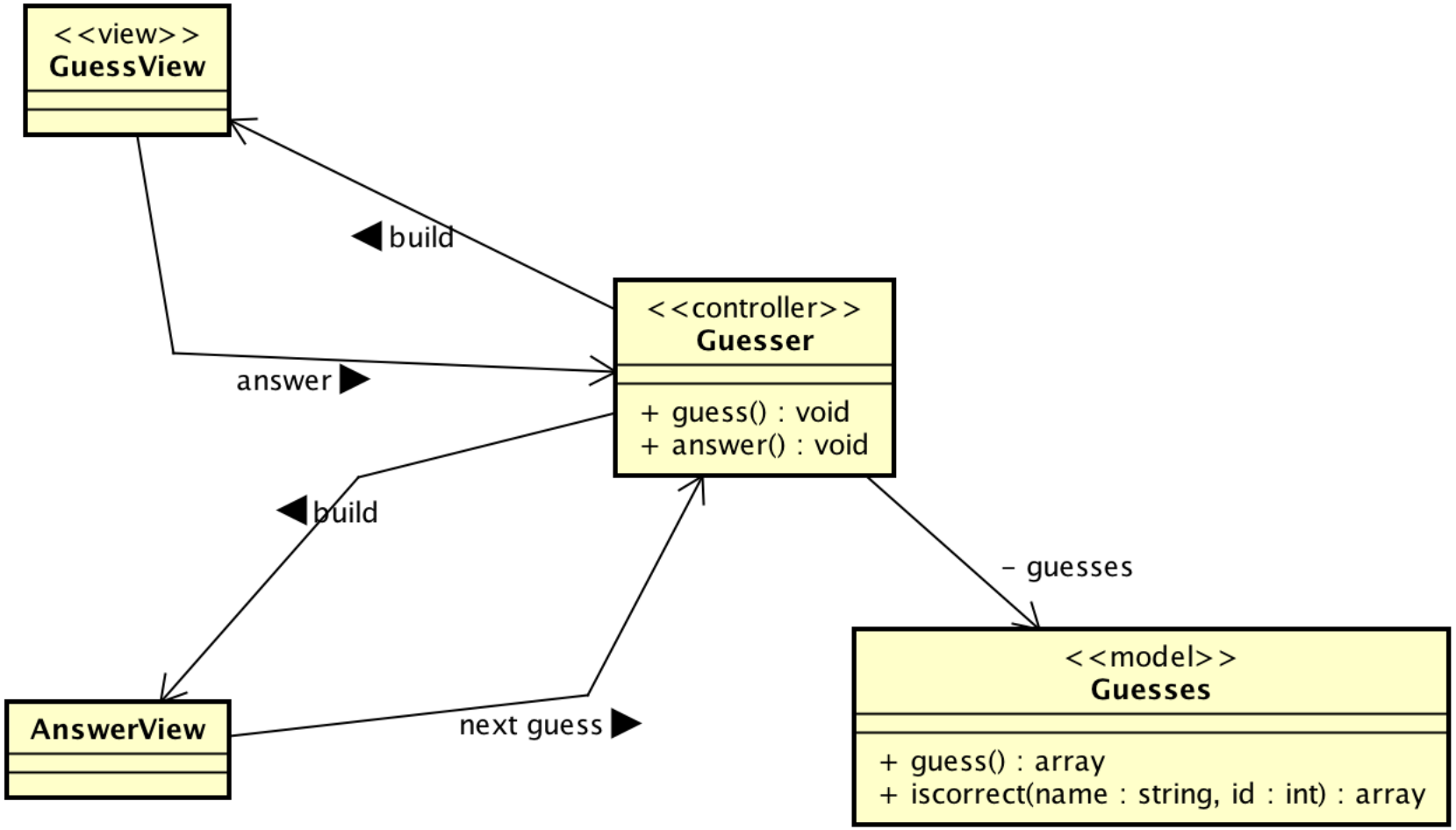
Questions:

1. Which bits are dynamic?
2. How many separate views?



What does the mockup tell us about the app?

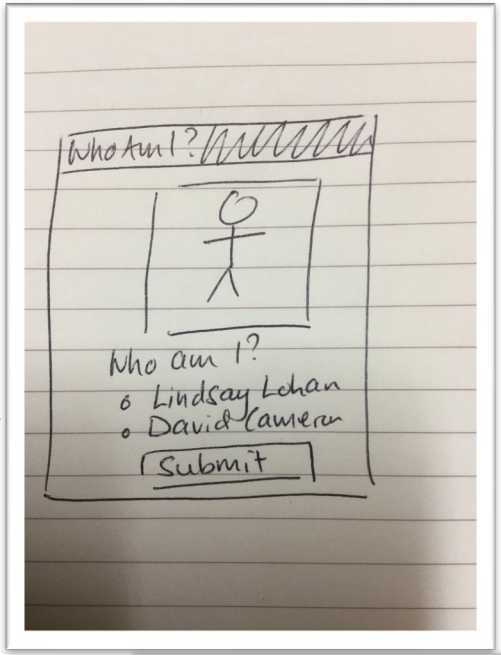
- The mockups show us that there are two separate views
- Each view sends a request back to the controller, which results in the other view being shown
- Structure – we'll create two methods in the controller
  1. To create the guess view with a new guess
  2. To display the answer view with information about whether answer was correct or not, and to allow user to get next guess



How will it work?

Request for guess – no form data

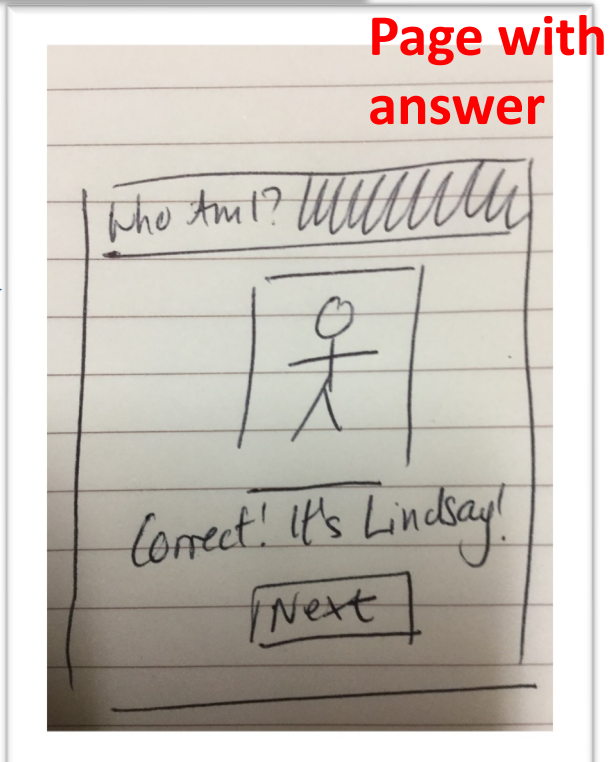
Controller



Page with new question

Request for answer **with** form data

Controller



Page with answer

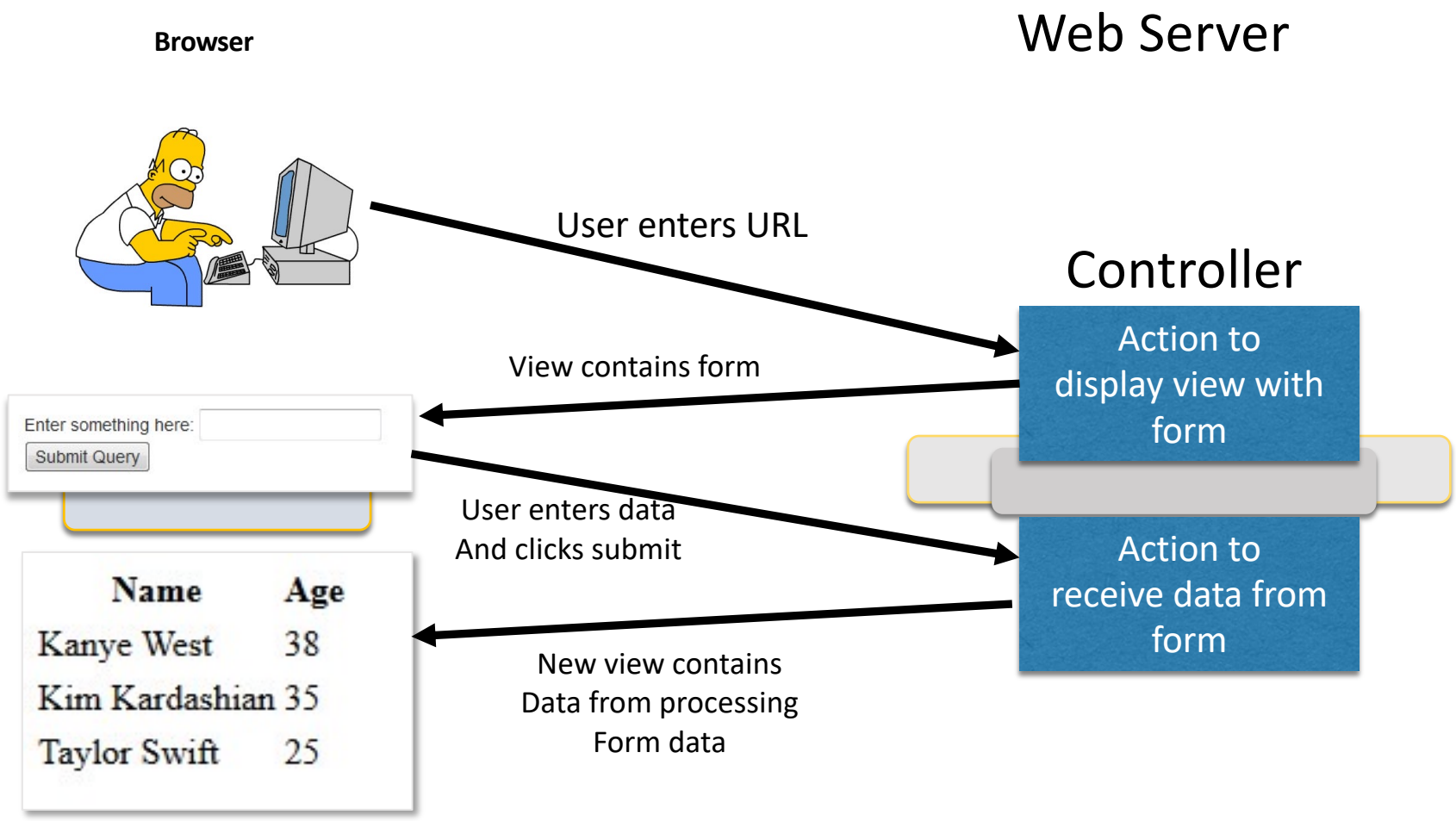
Code – 4 files

- Controller – controllers/Guesser.php
  - https://gist.github.com/simoncourtenage/f922d73e90f834e7d3f35a562d9a79ae
- Guess View – views/guessview.php
  - https://gist.github.com/simoncourtenage/800f284f234eae053572df1461ce9873
- Answer View – views/answerview.php
  - https://gist.github.com/simoncourtenage/8f06f1c93b58707190e83a615bb43df9
- Guesses model – models/Guesses.php
  - https://gist.github.com/simoncourtenage/0701a420ed5bdf2d272bf1e6a2c6397b

Passing data to controllers

- How do browsers pass data to controllers when they are called/requested?
- I.e., how is data sent in the HTTP request from the browser to server? And how does the controller access it?
- Two ways
  - Usual form data added to HTTP request (query string if GET, add to end of HTTP request if POST)
  - Add to URL as PATH data – **URL segments** in CodeIgniter

A typical scenario



GET – HTML form example

```
<h2>Find people</h2>
<div class="col-sm-2">
<form action="/people/index.php/People/find" method=GET>
<div class="form-group">
    <label>Age </label> <input class="form-control" type=text name=age>
    </div>
    <div class="form-group">
     <label>Gender </label><input class="form-control" type=text name=gender>
     </div>
    <input type=submit>
</form>
```

Code for web form

What URL is requested when the user fills in the form and clicks submit?

How form is displayed

GET requests

- if the user enters ‘30’ and ’female’, on the form then the requested URL is

`/people/index.php/People/find?age=30&gender=female`

- everything after ‘?’ is the **query string**
  - name/value pairs separated by ‘&’
  - Made up of data typed by user into form fields
  - Query string only used in URL for **GET** requests

How to access GET data

- In the controller, we can access form data using the ‘input’ object
- Does input sanitization
- POST is similar
- Uses HTML form field names to access data

```
class People extends CI_Controller {
    function getall()
    {
        $this->load->model('Personmodel');
        $data = $this->Personmodel->getall();
        $this->load->view('peoplelist',array('peeps' => $data));
    }

    function index()
    {
        $this->load->view('findview');
    }

    function find()
    {
        $age = $this->input->get('age');
        $gender = $this->input->get('gender');

        $this->load->model('Personmodel');
        $data = $this->Personmodel->find($age,$gender);
        $this->load->view('peoplelist',array('peeps' => $data));
    }
}
```

$this->input in controller

- $this->input
  - Object which is part of controller object – created by base controller class
  - Used to access and sanitize form data from user
- $this->input->get('NAME OF FORM FIELD')
  - If GET used on form
- $this->input->post('NAME OF FORM FIELD')
  - If POST used on form
- Return value – what user typed into that field in the form (or default value if not typed in or is hidden field)
- If form field not found, then returns NULL

CodeIgniter URLs - segments

- CodeIgniter splits URLs into “segments” after index.php
- E.g., /people/index.php/People/find
  - Segment 1 = “People” – name of controller
  - Segment 2 = “find” – name of action
- We can add PATH data after action e.g,
  - /people/index.php/People/find/music
  - Segment 1 = ”People”
  - Segment 2 = ‘find’
  - Segment 3 = ‘music’

Accessing segments in controllers

- In config/autoload.php, add ‘url’ to array of helpers
  - $autoload['helper'] = array('url');
- In controller, use code like this:

```
class People extends CI_Controller {

    function find()
    {
        // add lines to get form data here
        $category = $this->uri->segment(3);
```

- Can use 2nd argument to segment() to specify default value

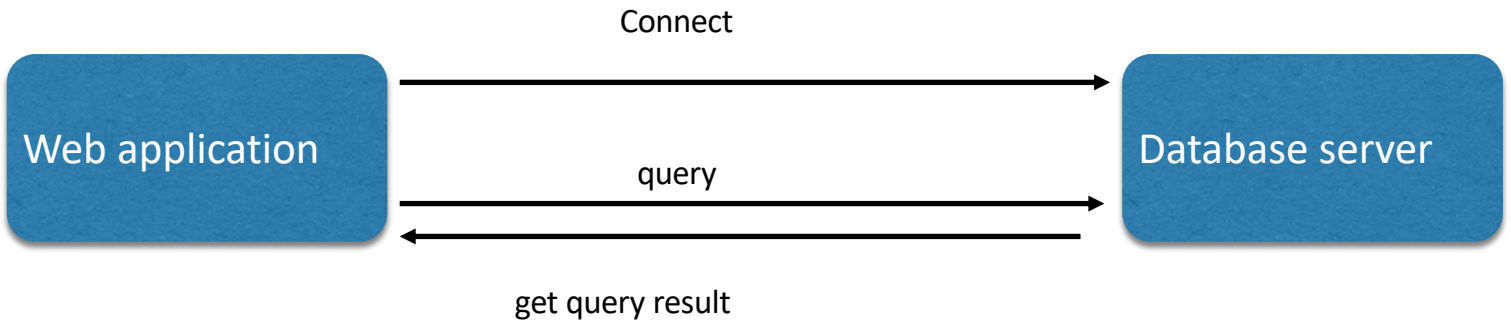
Databases

- Web applications and databases
- Working with databases - the wrong way
  - Why it’s the wrong way
- Working with databases - the right way
- The CodeIgniter way - and why it’s not quite the right way

Web applications and databases

- Web applications and databases are separate things
- Databases - managed by database server
- Web applications - managed by web server
  - Uses database by connecting to database server
  - Host name, database name, username, password

Connecting to database

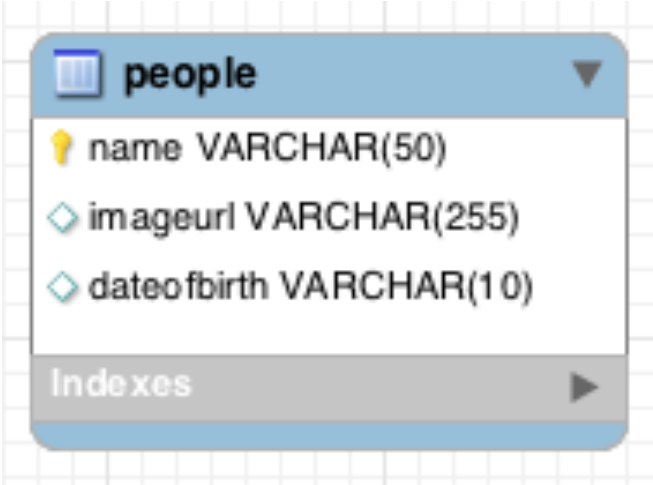



```
$dbhost = 'localhost';
$dbuser = 'root';
$dbpass = 'test1';

$db = mysql_connect($dbhost, $dbuser, $dbpass) ;
if (!$db)
  {
    die('Could not connect: ' . mysql_error());
  }
mysql_select_db("whoami", $db);

$SQL = "select * from people";
$runSQL = mysql_query($SQL) or die(mysql_error() . ": $SQL ");
while (($persondata = mysql_fetch_array($runSQL)) !== false) {
    print "name: "          . $persondata[0] . "\n";
    print "date of birth: " . $persondata[2] . "\n";
}
```

- Typical PHP code to use MySQL database
- DB functions hide away the details of the web app/database communication

Databases the wrong way

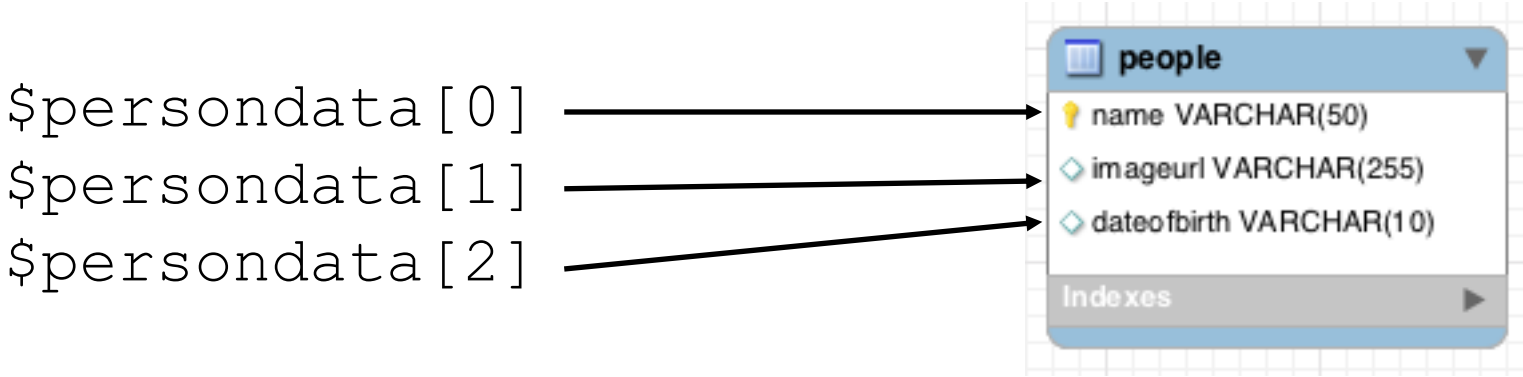
- Consider this code
- What's the relationship between code and table?



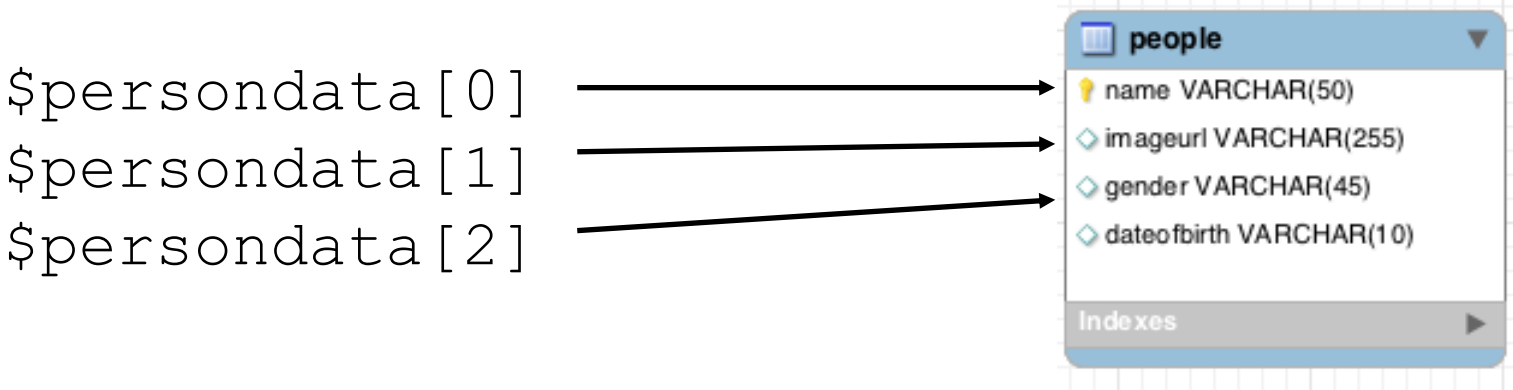

```php
$dbhost = 'localhost';
$dbuser = 'root';
$dbpass = 'test1';

$db = mysql_connect($dbhost, $dbuser, $dbpass) ;
if (!$db)
  {
    die('Could not connect: ' . mysql_error());
  }
mysql_select_db("whoami", $db);

$SQL = "select * from people";
$runSQL = mysql_query($SQL) or die(mysql_error() . ": $SQL ");
while (($persondata = mysql_fetch_array($runSQL)) !== false) {
    print "name: "          . $persondata[0] . "\n";
    print "date of birth: " . $persondata[2] . "\n";
}
```

What happens if we change the table?

Databases the wrong way



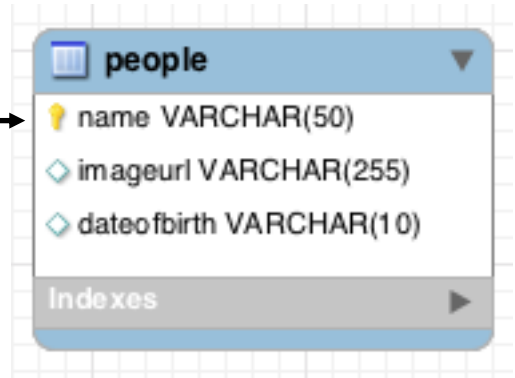
What happens if we change the table…



The problem

- If we change the table, then we need to remember to change the code
- Relationship between code and table is based on relative position
  - the first column in the table maps to first index position in array

```
$persondata[0]
$persondata[1]
$persondata[2]
```

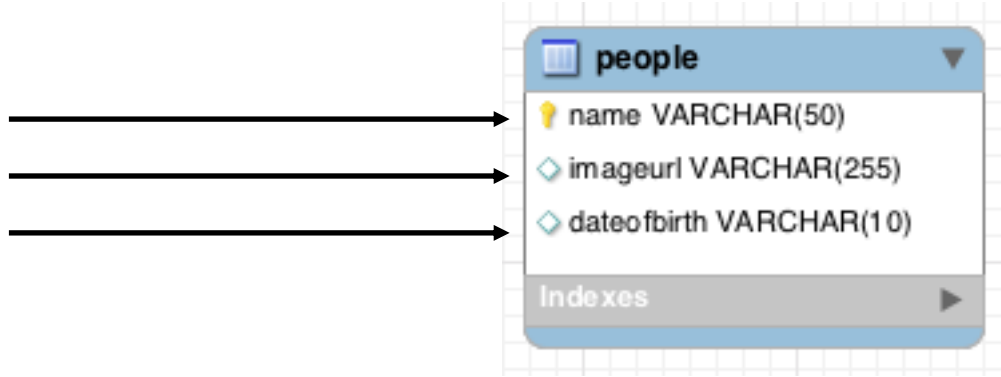


Databases the right way

"mapping"

- Class ⟷ table
- Object ⟷ row
- Attributes ⟷ columns

```
class Person {
    private $name;
    private $imageurl;
    private $dateofbirth;
}
```



What happens if the table changes? Nothing…

Objects - Tables

- To achieve this kind of object-table mapping, we need some additional support - through some kind of ORM
  - Object-Relational Mapping - usually an added library
  - Many different flavours of ORM - differ mostly in level of support
    - highest : class/object - table/row relationship almost invisible. E.g, updating object causes row update - **Doctrine 2** library (can used in Zend and Symfony PHP frameworks)
    - lowest : class/object - table/row relationship - support only via **associative arrays** - e.g, PDO (PHP built-in), mysql_fetch_assoc()

Active Record

- Active Record is a "**database pattern**" - describes ORM
  - Used by many MVC PHP frameworks (varying support)
- Objects tied to table rows, classes relate to tables
  - object attributes = columns in row
- What we would **like** to do…..create an object in our program

```
$person = new Person("Kanye West","http://www.pics.com/kanye1.jpg","08-06-1977");
```

And for this to be equivalent to doing an insert into the DB…

```
INSERT INTO
    PEOPLE ('name','imageurl','dateofbirth')
        VALUES ('Kanye West',"http://www.pics.com/kanye1.jpg","08-06-1977");
```

QueryBuilder in CodeIgniter

- CodeIgniter has a **modified** form of Active Record built-in - called Query Builder (called 'ActiveRecord' in CI v2.x)
- See CodeIgniter documentation
- Provides basic object-row mapping
- Some query support via API
- Not as advanced as other frameworks - could be considered serious limitation
  - E.g., creating an object does not automatically create row etc.

How to use QB in CI

- Step 1: configure CodeIgniter with your database information: hostname, database name, username, password – see config/database.php
- Step 2: load database object
  - automatically makes DB connection (destroying object closes connection)
- Step 3: build and make queries using QB API
- Step 4: process results – rows are objects with attributes matching table columns

Configuring CI for your database

- edit **application/config/database.php** file

```
$db['default']['hostname'] = 'elephant.ecs.westminster.ac.uk';
$db['default']['username'] = 'w1234567';
$db['default']['password'] = 'abcdefghijkl';
$db['default']['database'] = 'w1234567_0';
```

typical university configuration

```
$db['default']['hostname'] = 'localhost';
$db['default']['username'] = 'root';
$db['default']['password'] = 'test';
$db['default']['database'] = 'whoami';
```

typical home PC/Mac configuration

Using Query Builder

- Do this only in your Model classes - respect MVC!
- Load database object – usually in **model constructor**
  - `$this->load->database();`
- Model class will have database object as attribute `$this->db`
- Database object -> methods provide API to build queries etc.
- Associative arrays used to pass data to database object (for selects, inserts etc.)
- Get results back as objects - attributes -> columns

Example select

- Select all the rows and put names into an array

```php
$names = array();
$result = $this->db->get('people');
foreach ($result->result() as $row) {
    // $row is an object - attributes are column names
    $names[] = $row->name;
}
```

- $this->db = database object in model
- $this->db->get() = how to make select query

1. Call to DB object to ‘get’ rows
- DB object is ‘built-in’ to model class

2. get result object

```
$names = array();
$result = $this->db->get('people');
foreach ($result->result() as $row) {
    // $row is an object - attributes are column names
    $names[] = $row->name;
}
```

3. each call to $result->result()
returns one row as php object

4. php object has public
attributes whose names
are table columns

Queries

- SELECT * FROM people
- SELECT name,age FROM people WHERE age > 30
- UPDATE people SET age=104 WHERE name='Simon'
- DELETE FROM people WHERE age > 100 and GENDER = 'male';
- SELECT * FROM people WHERE name LIKE 'K%'

- **Usually specify whole query as one string**
- **QueryBuilder allows us to build up a query bit by bit before executing it**

Example select

- SELECT * FROM people WHERE gender = ‘male’

```php
// build up the query through series of
// ActiveRecord API calls
$this->db->where('gender','male'); // create the WHERE clause
$result = $this->db->get('people'); // build SELECT and make query
// check how many rows are in result
if ($result->num_rows() == 0) {
    return false; // no results? we could return false to tell caller
}
$names = array();
foreach ($result->result() as $row) {
    // $row is an object - attributes are column names
    $names[] = $row->name;
}
```

EXAMPLE SELECT

- Query Builder allows you to use associative arrays to pass data

```php
$this->db->where(array('gender' => 'female','name' => 'Kanye West'));
$result = $this->db->get('people'); // build SELECT and make query
// check how many rows are in result
if ($result->num_rows() == 0) {
    return false; // no results? we could return false to tell caller
}
$names = array();
foreach ($result->result() as $row) {
    // $row is an object - attributes are column names
    $names[] = $row->name;
}
```

How many rows will this return?

EXAMPLE INSERT

- INSERT INTO people (‘name’,’imageurl’,’dateofbirth’,’gender’) VALUES (‘Taylor Swift’,’http://www.pics.com/swift1.jpg',’13-12-1989',’female')

```
$data = array('name' => 'Taylor Swift',
              'imageurl' => 'http://www.pics.com/swift1.jpg',
              'dateofbirth' => '13-12-1989',
              'gender' => 'female');
$res = $this->db->insert('people',$data);
```

Summary

- Databases - access in models only (remember to load database in model constructor)
- Use $this->db object in model class
- Build query via ActiveRecord functions
- Rows returned as objects
- Also: GET/POST to get data into controller

Further reading

- CodeIgniter documentation
- https://codeigniter.com/user_guide/tutorial/index.html - basic CI tutorial
- https://codeigniter.com/user_guide/libraries/input.html - using the Input class
- https://codeigniter.com/user_guide/libraries/uri.html - the URI class (for segments)
- https://codeigniter.com/user_guide/database/index.html - database library in CI