Intro to PHP

message <<-MARKDOWN

### Goal

To teach you PHP we are going to use a "Real World" example. You've decided to create a voting system for you and your friends to play with. You've decided at a minimum, you'd like to allow users to:

* view the topics sorted by number of votes

* vote on topics

* create, edit, and destroy topics

You've sketched up an initial screenshot of what you want it to look like:

![screenshot](img/suggestotron_mockup.png)

### Meta-Goal

When you have completed today's goal of getting the basic application online you should understand:

* Basic PHP syntax

* How to try your PHP code (Psy Shell)

* How to go from requirements to a new working web application

* How to get your application online

* The basic tools a web developer uses (source control, editor, console, local server)

### Schedule

* 1-ish hour of introduction to the language

* 5-ish hours of building your application

This is just a rough guideline, not a mandate. Some steps you'll go over and some you'll go under. It'll all work out by the end of the day...

### Requirements

We're going to be working with:

* PHP
* MySQL Database Server
* Terminal
* Bootstrap 3
* Sublime Text Editor
* Psysh
* Vagrant & VirtualBox
* Ubuntu Linux

We will provide you with everything you need to get started on the day.

### Working Effectively and Efficiently

We highly recommend you do the following:

* Open your browser fresh or hide any windows you already have open.
  * Bring up one window with two tabs
    * One for this content
    * One for interacting with your app.
* Open your text editor and _do not ever close it_. We're not quitters.
* When we get to the Terminal, you'll want to keep that open too
* Hide all extra applications. Turn off twitter, IM, and all other distractions.

By minimizing the number of things you interact with, you reduce the amount of time spent switching between them and the context lost as you work through the lessons. Having 50 tabs open in your web browser gets confusing and wastes time.

### Format

Each lesson will look like this:

```
<div style="background: white; border: 1px solid black; padding-left: 4em;">
  <h1 class="doc_title" style="margin-top: 0em;">Step Title</h1>
  <div class="goals">
    <h1>Goal:</h1>
    <p>Description of the current step.
    <p>Red because big goals are scary
  </div>
  <div class="steps">
    <h1>Steps:</h1>
    <div class="step">
      <h1>
        <img src="img/fake_checkbox.png">&nbsp;<span class="prefix">Step 1</span>
      </h1>
      <div>
        <pre>Commands to run, or code to write</pre>
      </div>
      <p>Yellow because we've gotten it done, but we have no clue what's going on.
    </div>
  </div>
  <div class="explanation">
    <h1>Explanation</h1>
    <p>Details of what the steps actually did... spell out the cause and effect.
    <p>Green because we can tie everything together now.
  </div>
</div>
MARKDOWN
```

next_step "php_language"

```
goals do
  goal "Be able to use the basic building blocks of PHP code"
  goal "Use Psysh to run PHP code"
  goal "Do simple calculations"
  goal "Use and understand variables"
  goal "Use and understand arrays"
  goal "Use built-in functions"
  goal "Use loops and conditional statements"
  goal "Create and use a custom function"
end

steps do

  step do
    message "Start by connecting to the virtual machine:"

    console_without_message "vagrant ssh"
  end

  step do
    message "Type this in the terminal to start Psy Shell, a program which lets you try out PHP code:"

    console_without_message "psysh"

            message "Yours might look different, but it should look something like this:"

            console_without_message "Psy Shell v0.1.4 (PHP 5.5.3 — cli) by Justin Hileman\n>>>"
  end

  step do
    message "Next try some simple math that's built into PHP. Type these lines into psysh:"
    source_code :php, "3 + 3;\n7 * 6;"
  end

  step do
    message "**Variables** are names with values assigned to them. Each name starts with the `$` symbol."

    source_code :php, "$my_variable = 5;"
    message "This assigns the value `5` to the name `$my_variable`."
  end

  step do
    message "You can also do math with variables:"
    source_code :php, <<-PHP
$my_variable + 2;
$my_variable * 3;
    PHP
  end

  step do
    message "To display the value of a variable, or any other value, you use `echo`. Try the following"

    source_code :php, <<-PHP
    echo $my_variable;
    PHP

    source_code :php, <<-PHP
    echo "Programming is easy!";
    PHP

    source_code :php, <<-PHP
    echo "13 * 8";
    PHP

    source_code :php, <<-PHP
    echo 13 * 8;
    PHP

    source_code :php, <<-PHP
    echo "My variable is: $my_variable";
    PHP
  end

  step do
    message "Variables can also hold more than one value. This is called an **array**."

    source_code :php, '$fruits = ["kiwi", "strawberry", "plum"];'

    message "Here we're using the variable `$fruits` to hold a collection of fruit names."
  end

  step do
     message "Each value in an array has a key. These can be numbers or words:"

     source_code :php, '$fruits = [2 => "kiwi", "berry" => "strawberry", 6 => "plum"];'

     message "If you do not specify a key, they will be numbers starting from zero, or the last
number."
  end

  step do
    message "You can easily add a new value to an array:"

    source_code :php, '$fruits[] = "orange";'
  end

  step do
    message "You can also remove items from an array:"

    source_code :php, 'unset($fruits["berry"]);'

    message "Or remove the last item added:"

    source_code :php, 'array_pop($fruits);'
  end


  step do
    message "To inspect your array, you can use a built-in function"


    source_code :php, 'print_r($fruits);'


    message "For more detailed information about the variable, you can use `var_dump()`"


    source_code :php, 'var_dump($fruits);'
  end


  step do
    message "A **conditional** runs code only when a statement evaluates to true."


    source_code :php, <<-PHP
if ($my_variable > 1) {
  echo "YAY!";
}
    PHP


    message "This prints `YAY!` if the value stored in `$my_variable` is greater than 1."


    message "Try changing the `>` in the conditional to a `<`."


    message "`<` and `>` are called operators. There are many kinds of operators such as the `+`, `-`, `*` and `/` arithmetic operators."
  end


  step do
     message "To perform an action if the conditional statement does not evaluate to true, we use `else`:"


     source_code :php, <<-PHP
```

```
if ($my_variable < 1) {
        echo "YAY!";
} else {
        echo "OH NOES! 😢";
}
   PHP
  end


  step do
    message "You can combine these two, to add more conditionals using `elseif`:"


    source_code :php, <<-PHP
if ($my_variable < 1) {
  echo "YAY!";
} elseif ($my_variable > 4) {
  echo "YIPPEE! 😍";
} else {
  echo "OH NOES! 😢";
}
    PHP
  end


  step do
    message "A common operation is to perform the same task on each value in an array. We call this looping, or iteration."


    message "The simplest loop, is known as a `while` loop which will iterate until a given condition returns false:"


    source_code :php, <<-PHP
      $fruits = ["kiwi", "plum", "orange", "banana"];
      $i = 0;
      while ($i < sizeof($fruits)) {
        echo $fruits[$i] . PHP_EOL;
        $i++;
      }
    PHP
  end

  step do
    message <<-MARKDOWN
```

Another common loop, is known as the `for` loop. This loop has three conditions:

- The first condition is evaluated at the beginning no matter what
- The second is evaluated at the beginning of each iteration, and will cause the loop to end when it returns false (just like a while loop!)
- The third is evaluated at the end of each iteration

```
  MARKDOWN

  source_code :php, <<-PHP
for ($i = 0; $i < sizeof($fruits); $i++) {
  echo $fruits[$i] . PHP_EOL;
}
  PHP
end

step do
  message "Another common loop is `foreach` which is the most common and easiest way to loop through an array:"

  source_code :php, <<-PHP
foreach ($fruits as $fruit) {
  echo $fruit . PHP_EOL;
}
  PHP
end

step do
```

message "While PHP features thousands of built-in functions, you can also make your own:"

source_code :php, <<-PHP

```
function happy()
{
  echo "YAY!";
}
```

PHP

message "You run the function just like a built-in function:"

console_without_message "happy();"

end

step do

source_code :php, <<-PHP

```
function pluralize($word)
{
  return $word . "s";
}
pluralize("kiwi");
```

PHP

message "Functions can take **parameters**, which are the variables they work on. In this case, we made a function called `pluralize` that takes one parameter, a word."

message "Functions can also return data. In this case, pluralize returns the word with an 's' added to the end of it. In PHP, we use the `return` keyword to do this."

end

step do

message "When using PHP outside of Psysh, you should put it inside `<?php` and `?>` tags:"

source_code :php, <<-PHP

<?php

    // Code goes Here

    ?>

    PHP

    tip "You may notice the two-slashes at the start of the middle line above. This is known as a comment and is ignored by PHP."

  end

  step do

    message 'As a shortcut, you can use what is known as "short echo tags" to echo simple variables:'

    source_code :php, "<?=$variable;?>"

  end

end

explanation do

  message "With this small amount of code, you've learned a lot of the syntax you will use every day!"

end

next_step "getting_started"

img src: "img/phpinfo.png", alt: "Start Page"

goals do

  goal "Create Your New Application"

message "Let's get started! By the end of this step, we'll have a brand-new (empty) web app."

end

steps do

  insert "tip_TA"

  step do

    insert 'switch_to_home_directory'

```
  end

  step do
    console "mkdir suggestotron"
    message "This command creates a new directory for us to store our project in."
  end

  step do
    console "cd suggestotron"
  end

  step do
    console "mkdir public"
    message "This creates a directory to hold everything that will be publicly accessible"
  end

  step do
    message <<-MARKDOWN
Open the `suggestotron` directory as a project in your text editor.

In **Sublime Text**, you can use the `Project > Add Folder to Project...` menu option:

<img src='img/sublime_add_folder_to_project.png' />

Select your `suggestotron` directory from the file picker that opens. If everything works out Sublime should look something like this:

<img src='img/sublime_project_as_folder.png' />
    MARKDOWN
  end

  step do
    message "Next, create a new file called `index.php` in the `public` directory, and type the following:"
    source_code :php, <<-PHP
      <?php
      phpinfo();
      ?>
    PHP
  end

  step do
    message "Now, start the PHP web server"

    console "php -S 0.0.0.0:8080 -t ./public/"
  end

  step do
    text "Point your web browser to "
    url "http://localhost:8080"
    p 'See your "web app" actually running!'
  end
end

explanation do
  message "We now have a running PHP server, showing the configuration of our PHP installation."

  message "Any changes you make from now on will be immediately visible simply by refreshing your browser."
end
```

next_step "add_the_project_to_a_git_repo"

```
goals do
  goal "Create a local git repository"
  goal "Add all our files to the git repository"
  message "To track our changes over time, we are going to use a version control system — we have chosen to go with `git`."
end


steps do
  step do
    message "Open up a second terminal. For Mac OS X, press `Cmd+T`. On Linux, try `Ctrl+T` and on Windows, you will need to open a new window the same way you opened the current one."


    message "Once you have a second terminal, connect to the virtual machine again:"


    console_without_message "vagrant ssh"
  end


  step do
    message "Switch to our project directory"


    console "cd /var/www/suggestotron"
  end


  step "Setup Git" do
    console <<-CMD
    git config --global user.email "you@example.com"
    git config --global user.name "Your Name"
    CMD


    message "This tells git (and anyone who has access to your repository) who made the changes — this is critical when working in teams."
  end


  step "Initialize our Repository" do
    console "git init"
    result "Initialized empty Git repository in /var/www/suggestotron/.git/"

    message "It doesn't look like anything really happened, but 'git init' initialized its repository (repo) in a hidden directory called `.git`. You can see this by typing `ls -al` (list all files)."
  end

  step do
    console "git status"

    result <<-TEXT
    # On branch master
    #
    # Initial commit
    #
    # Untracked files:
    #   (use "git add <file>..." to include in what will be committed)
    #
    # public/
    #
    nothing added to commit but untracked files present (use "git add" to track)
    TEXT

    message "`git status` tells you everything git sees as modified, new, or missing."
  end

  step do
    console "git add ."
    message "`git add .` tells git that you want to add the current directory (aka `.`) and everything under it to the repo."
    tip "git add" do
      message <<-MARKDOWN
        With Git, there are usually many ways to do very similar things.

        * `git add foo.txt` adds a file named `foo.txt`
        * `git add .` adds all new files and changed files, but *keeps* files that you've deleted

        * `git add -A` adds everything, including deletions


        "Adding deletions" may sound weird, but if you think of a version control system as
keeping

        track of *changes*, it might make more sense. Most people use `git add .` but `git add -
A`

        can be safer. No matter what, `git status` is your friend.

      MARKDOWN

    end

  end


  step do

    console "git commit -m \"Created my first PHP file\""

    message "`git commit` tells git to actually _do_ all things you've said you wanted to do."

    message "This is done in two steps so you can group multiple changes together."

    message "`-m \"Created my first PHP file\"`  is just a shortcut to say what your commit
message is. You can skip that part and git will bring up an editor to fill out a more detailed
message."


    fuzzy_result <<-TEXT

    [master (root-commit) a74d816] Created my first PHP file

     1 file changed, 3 insertions(+)

     create mode 100644 public/index.php

    TEXT

  end

end


explanation do

  message <<-MARKDOWN

By checking your application into git now, you're creating a record of your starting point.
Whenever you make a change during today's workshop, we'll add it to git before moving on.
This way, if anything ever breaks, or you make a change you don't like, you can use git as an
all-powerful "undo" technique. But that only works when you remember to commit early and
often!

  MARKDOWN

end
```

next_step "creating_a_database"

```
goals do
    goal "Create a database"
    message "Lets create our Suggestotron database add some example data"
end

steps do

    step do
        message "Run the MySQL command line tool"
        console "mysql -u root -p"

        message "You will be prompted for a password. The password is `root`."

        important "Using the root username for a real website is a *bad idea*. For more information on adding a new user, see [this tutorial](https://www.digitalocean.com/community/articles/how-to-create-a-new-user-and-grant-permissions-in-mysql)."
    end

    step do
        message "Create your database"

        source_code :sql, <<-SQL
            CREATE DATABASE suggestotron;
            USE suggestotron;
        SQL
    end

    step do
        message "Next, create our table, it's going to look like this:"
        model_diagram header: 'topics', fields: %w(id title description)
        source_code :sql, <<-SQL
                CREATE TABLE topics (
                        id INT unsigned NOT NULL AUTO_INCREMENT,
                        title VARCHAR(255) NOT NULL,
                        description TEXT NULL,
                        PRIMARY KEY(id)
                );
        SQL
end

step do
        message "Now we can insert our test data"
        source_code :sql, <<-SQL
                INSERT INTO topics (
                        title,
                        description
                ) VALUES (
                        'Make Rainbow ElePHPants',
                        'Create an elePHPant with rainbow fur'
                );

                INSERT INTO topics (
                        title,
                        description
                ) VALUES (
                        'Make Giant Kittens',
                        'Like kittens, but larger'
                );

                INSERT INTO topics (
                        title,
                        description
                ) VALUES (
                        'Complete PHPBridge',
                'Because I am awesome'
            );
        SQL

        message "After each `INSERT` you will see something like:"
        fuzzy_result "Query OK, 1 row affected (0.02 sec)"
    end

    step do
        message "To view our data, we can `SELECT` it from the table:"
        source_code :sql, <<-SQL
            SELECT * FROM topics;
        SQL

        result <<-TEXT
            +----+------------------------+-------------------------------------+
            | id | title                  | description                         |
            +----+------------------------+-------------------------------------+
            |  1 | Make Rainbow ElePHPants | Create an elePHPant with rainbow fur |
            |  2 | Make Giant Kittens      | Like kittens, but larger             |
            |  3 | Complete PHPBridge      | Because I am awesome                 |
            +----+------------------------+-------------------------------------+
            3 rows in set (0.00 sec)
        TEXT
    end

    step do
        message "We are done with the database for now. To quit, type the following:"

        console_without_message "\\q"
    end
end
explanation do
```

```
    message "You have now create your first database, your first table, *and* your first rows of data!"


    message "We will be accessing this data via our PHP code in our application. Not only will our application be able to read it, but it will be able to create new data, edit, and delete existing data."
end
```

next_step "creating_a_data_class"

```
goals do
    goal "Write a class to get our data"
    goal "Display the topics on a web page"
    message "A class is a special piece of code for performing a given task"
end


steps do
    step do
        message "Create a new file called `TopicData.php` in the `public` directory"


        message "Type the following to create our empty class:"


        source_code :php, <<-PHP
            <?php
            class TopicData {
                // CLASS CONTENTS GO HERE
            }
            ?>
        PHP
    end


    step do
        message "Now create a connect function:"


        source_code :php, <<-PHP
```

```
      <?php
      class TopicData {
          protected $connection;

          public function connect()
          {
              $this->connection = new PDO("mysql:host=localhost;dbname=suggestotron", "root", "root");
          }
      }
      ?>
    PHP
  end

  step do
    message "Next lets make the class fetch all of our topics:"

    source_code :php, <<-PHP
      <?php
      class TopicData {

          protected $connection = null;

          public function connect()
          {
              $this->connection = new PDO("mysql:host=localhost;dbname=suggestotron", "root", null);
          }

          public function getAllTopics()
          {
              $query = $this->connection->prepare("SELECT * FROM topics");
              $query->execute();
```

```
                        return $query;
                    }
                }
                ?>
            PHP

            tip "Notice how we use the same `SELECT` as in the previous section"
        end

        step do
            message "Now we can use the data class in `index.php`, to get our topics:"

            source_code :php, <<-PHP
                <?php
                require 'TopicData.php';

                $data = new TopicData();
                $data->connect();

                $topics = $data->getAllTopics();
            PHP

            message "Now `$topics` is a database result object containing our topics."
        end

        step do
            message "Now we have our topics lets display them by using a `foreach` to
iterate over them:"
            source_code :php, <<-PHP
                foreach ($topics as $topic) {
                    echo "<h3>" .$topic['title']. " (ID: " .$topic['id']. ")</h3>";
                    echo "<p>";
                    echo nl2br($topic['description']);
                    echo "</p>";
                    }
          PHP
    end

    step do
          message "To see what this looks like, refresh the application in your browser!"

    img src: "img/List_of_topics.png", alt: "List of topics"
    end
end

explanation do
    message "With just these few lines of code we are able to connect to our database, fetch our data, and dynamically create an HTML page. *How neat is that?*"
end
```

next_step "adding_topics"

```
goals do
    goal "Create a way for users to add their own topics to the database"

    message "Now we are adding some interactivity to our site!"
end

steps do
    step do
          message "Create a new file, called `add.php` in the `public` directory"

          message "Then we will add an HTML form:"

          source_code :html, <<-HTML
                <h2>New Topic</h2>
                <form action="add.php" method="POST">
                      <label>
```

```
                        Title: <input type="text" name="title">
                    </label>
                    <br>
                    <label>
                        Description:
                        <br>
                        <textarea name="description" cols="50" rows="20"></textarea>
                    </label>
                    <br>
                    <input type="submit" value="Add Topic">
                </form>
            HTML

            message "You can browse to this file at <http://localhost:8080/add.php>"

            message 'When you click on "Add Topic", the form will be submitted back to the server'

            message 'Adding the following will let you see what was sent'

            source_code :php, <<-PHP
                <?php
                var_dump($_POST);
                ?>
            PHP

            tip "We are using a `POST` action in our `<form>`, therefore the data will be available in the `$_POST` super global."
        end

        step do
            message "Now that we have our data, we can go ahead and save it in our database."
            message "Add the following the top of `add.php`:"


            source_code :php, <<-PHP
                <?php
                require 'TopicData.php';


                if (isset($_POST) && sizeof($_POST) > 0) {
                    $data = new TopicData();
                    $data->add($_POST);
                }
                ?>
            PHP


            message "Submitting the form in your browser will now show this:"


            fuzzy_result "Fatal error: Call to undefined method TopicData::add() in /var/www/suggestotron/public/add.php on line 6"


            message "Don't worry! This is because we haven't added a `TopicData->add()` method yet. We will do that next!"
        end


        step do
            message "Going back to our `TopicData` class, add the `add` method:"


            important "For security, we are using a prepared query to ensure our data is escaped securely before sending it to our database"


            source_code :php, <<-PHP
                public function add($data)
                {
                    $query = $this->connection->prepare(
                        "INSERT INTO topics (
                            title,
                            description
                        ) VALUES (
                                :title,
                                :description
                        )"
                );

                $data = [
                        ':title' => $data['title'],
                        ':description' => $data['description']
                ];

                $query->execute($data);
        }
PHP
```

tip "Notice how we're using the same `INSERT` SQL code from earlier"

end

step do

message "If you submit your form now, you will see another error:"

fuzzy_result "Fatal error: Call to a member function prepare() on a non-object in /var/www/suggestotron/public/TopicData.php on line 20"

message "This is because we **forgot** to call `TopicData->connect()`. Wouldn't it be nice if we didn't even *have* to remember this?"

message "We can do this by using a special method called `__construct`. This is known as the constructor and is *automatically called* whenever we create a `new` instance of the class."

```
source_code :php, <<-PHP
        public function __construct()
        {
                $this->connect();
```

```
            }
        PHP

        message "Now, whenever we call `new TopicData()` it will automatically connect to the database"
    end

    step do
        message "**Now** when you submit your form, it will save the topic, yay! You can verify this by looking at `index.php`, <http://localhost:8080>"
    end

    step do
        message "We can automatically forward our users to the list by using the `header()` method with a `Location: /url` argument."

        message "Add the following after the call to `$data->add($_POST)`:"

        source_code :php, <<-PHP
            header("Location: /");
            exit;
        PHP

        message "Once we send the header, we must be sure to `exit;` so no other code is run."
    end
end

explanation do
    message "Our users can now add their own topics, no SQL knowledge required!"

    message "We taking the users input from an HTML form, `$_POST`, and using `INSERT` to add it to our database."
end
```

next_step "editing_topics"

```
goals do
    goal "Allow users to edit topics"

    message "Let users change existing data"
end

steps do
    step do
        message "Let us add an edit link for each Topic"

        message "In `index.php` change our foreach to include the link:"

        source_code :php, <<-PHP
            <?php
            foreach ($topics as $topic) {
                echo "<h3>" .$topic['title']. " (ID: " .$topic['id']. ")</h3>";
                echo "<p>";
                echo nl2br($topic['description']);
                echo "</p>";
                echo "<p><a href='/edit.php?id=" .$topic['id']. "'>Edit</a></p>";
            }
            ?>
        PHP

        message "The link has been added at the end of our `foreach`. The link has an argument for the `id`."

        tip "URL arguments are known as `GET` arguments. They are added to the URL after a `?` and can be found in the `$_GET` superglobal array (just like `$_POST`). Multiple arguments are separated by an `&`."
    end

    step do
```

message "Next create another new page, `edit.php`, and add an edit form. This will look almost identical to your new topic form:"

```
source_code :html, <<-HTML
    <h2>Edit Topic</h2>
    <form action="edit.php" method="POST">
        <label>
            Title: <input type="text" name="title" value="<?=$topic['title'];?>">
        </label>
        <br>
        <label>
            Description:
            <br>
            <textarea name="description" cols="50" rows="20"><?=$topic['description'];?></textarea>
        </label>
        <br>
        <input type="hidden" name="id" value="<?=$topic['id'];?>">
        <input type="submit" value="Edit Topic">
    </form>
HTML
```

message "We use echo tags `<?=$variable;?>` to output the current values into the form, and a hidden input to submit the topics ID back to the server, so we know which one we are editing."

end

step do

message "Then we need to fetch the requested topic, so that we can fill in the data."

message "We do this, by adding a `getTopic()` method to our `TopicData` class."

```
source_code :php, <<-PHP
    public function getTopic($id)
    {
        $sql = "SELECT * FROM topics WHERE id = :id LIMIT 1";
        $query = $this->connection->prepare($sql);

        $values = [':id' => $id];
        $query->execute($values);

        return $query->fetch(PDO::FETCH_ASSOC);
    }
PHP
```

message "Here, we introduce a `LIMIT 1` to ensure only one row is returned. We then use `$query->fetch(PDO::FETCH_ASSOC)` to return just the single row as an array."

```
fuzzy_result 'array(3) {
  ["id"]=>
  string(1) "3"
  ["title"]=>
  string(18) "Complete PHPBridge"
  ["description"]=>
  string(20) "Because I am awesome"
}'
end
```

step do

message "Now that you have a way to get the topic, we can use it in `edit.php` by adding the following at the top:"

```
source_code :php, <<-PHP
<?php
require 'TopicData.php';

$data = new TopicData();
```

```
        $topic = $data->getTopic($_GET['id']);

        ?>

        PHP


        message "At this point, you should be able to see your topic data in the edit form, but if you submit the form nothing will change (yet)."

    end


    step do

        message "We don't yet have any error checking in case a user tries to visit a link with a bad ID, or without an ID. Go ahead and play with the URL to see what happens!"


        message "Here are some example URLs:"


        ul {

            li {

                text "No ID: "

                url "http://localhost:8080/edit.php"

            }


            li {

                text "Invalid ID: "

                url"http://localhost:8080/edit.php?id=1337"

            }

        }


    end


    step do

        message "We can handle this by adding some extra checks in to your code:"


        source_code :php, <<-PHP

        <?php

        require 'TopicData.php';
```

```
            if (!isset($_GET['id']) || empty($_GET['id'])) {
                echo "You did not pass in an ID.";
                exit;
            }

            $data = new TopicData();
            $topic = $data->getTopic($_GET['id']);

            if ($topic === false) {
                echo "Topic not found!";
                exit;
            }
            ?>
            PHP
```

message "We use `isset`, and `empty` to check that the variable exists, and has a value"

message "We also check, to make sure that we did not get a `false` response from `TopicData->getTopic()` which would mean that no topic was found"

```
        end


        step do
```

message "Now that we have our form, we can go ahead and update the row in the database. First, lets add an `TopicData->update()` method"

```
            source_code :php, <<-PHP
                public function update($data)
                {
                    $query = $this->connection->prepare(
                        "UPDATE topics
                            SET
                                title = :title,
                                description = :description
                            WHERE
                        id = :id"
                    );

                    $data = [
                        ':id' => $data['id'],
                        ':title' => $data['title'],
                        ':description' => $data['description']
                    ];

                    return $query->execute($data);
                }
    PHP
  end

  step do
```

message "Finally, like with adding topics, we need to call it in `edit.php` by adding the following under the `require 'TopicData.php';`:"

```
    source_code :php, <<-PHP
                if (isset($_POST['id']) && !empty($_POST['id'])) {
                    $data = new TopicData();
                    if ($data->update($_POST)) {
                        header("Location: /index.php");
                        exit;
                    } else {
                        echo "An error occurred";
                        exit;
                    }
                }
          PHP

          message "Once you've made this change, check it out in your browser!"

          message "<http://localhost:8080/edit.php?id=2>"
```

end

end

explanation do

message "Similar to when our users created topics, we take our users input, `$_POST`, but this time we perform an `UPDATE` SQL command."

message "Also, we've started to add some input validation, which is critical for security!"

end

next_step "deleting_topics"

goals do

goal "Be able to delete topics from the database"

message "Now nobody will see your mistakes!"

end

tip "Deleting is very similar to editing or creating, so we'll make this brief!"

steps do

step do

message "First, we modify our `foreach` to include a Delete link, pointing to `delete.php`:"

```
source_code :php, <<-PHP
    <?php
    foreach ($topics as $topic) {
        echo "<h3>" .$topic['title']. " (ID: " .$topic['id']. ")</h3>";
        echo "<p>";
        echo nl2br($topic['description']);
        echo "</p>";
        echo "<p>";
        echo "<a href='/edit.php?id=" .$topic['id']. "'>Edit</a>";
```

```
                echo " | ";
                echo "<a href='/delete.php?id=" .$topic['id']. "'>Delete</a>";
                echo "</p>";
            }
            ?>
        PHP
    end


    step do
        message "Then we create `delete.php`, which will delete the topic, and then redirect back to `index.php` again"


        source_code :php, <<-PHP
            <?php
            require 'TopicData.php';

            if (!isset($_GET['id']) || empty($_GET['id'])) {
              echo "You did not pass in an ID.";
              exit;
            }

            $data = new TopicData();
            $topic = $data->getTopic($_GET['id']);

            if ($topic === false) {
              echo "Topic not found!";
              exit;
            }

            if ($data->delete($_GET['id'])) {
                header("Location: /index.php");
              exit;
            } else {
              echo "An error occurred";
```

```
                }
                ?>
            PHP
        end
        step do
            message "Finally, we add our `TopicData->delete()` method:"

            source_code :php, <<-PHP
                public function delete($id) {
                    $query = $this->connection->prepare(
                        "DELETE FROM topics
                            WHERE
                                id = :id"
                    );

                    $data = [
                        ':id' => $id,
                    ];

                    return $query->execute($data);
                }
            PHP
        end

        step do
            message "Once again, you can check this out in your browser. Try going to topic list and deleting the new topic you added earlier:"

            message "<http://localhost:8080/>"
        end
end

explanation do
        message "By now, you've should have a pretty good handle on how this works."
```

message "You're able to create, retrieve, update, and delete rows from the database, this is known as **CRUD**, and is something you will find in almost every application."

end

next_step "styling_suggestotron"

goals do

goal "Make our application look visually appealing"

end

steps do

step do

message "The first thing we need to do is to go back to our original pages (`index.php`, `add.php` and `edit.php`) and add more HTML to turn them into complete pages:"

message "Rather than using `echo` to output our HTML, we will instead put it *outside* our PHP, by using the closing and opening tags."

message "For example, `index.php` now becomes:"

```
source_code :php, <<-PHP
  <?php
  require 'TopicData.php';

  $data = new TopicData();
  $data->connect();

  $topics = $data->getAllTopics();
  ?>
  <!doctype html>
  <html>
    <head>
      <title>Suggestotron</title>
    </head>
    <body>
```

```
          <?php
          foreach ($topics as $topic) {
            echo "<h3>" .$topic['title']. " (ID: " .$topic['id']. ")</h3>";
            echo "<p>";
            echo nl2br($topic['description']);
            echo "</p>";
            echo "<p>";
            echo "<a href='/edit.php?id=" .$topic['id']. "'>Edit</a>";
            echo " | ";
            echo "<a href='/delete.php?id=" .$topic['id']. "'>Delete</a>";
            echo "</p>";
          }
          ?>
        </body>
      </html>
    PHP
  end

  step do
    message "We're going to use the Bootstrap framework to quickly and easily make our application look great!"

    message "You can download the framework from [getbootstrap.com](http://getbootstrap.com)"

    message "After unzipping it, copy the `css`, `js` and `fonts` directories in to your `public` directory"

    img src: "img/unpack_bootstrap.png"

    insert "tip_TA"
  end

  step do
```

```
    message "Now we can start by adding it to our document. In the `<head>` of `index.php` add the link to the stylesheet:"

    source_code :html, <<-HTML
    <link rel="stylesheet" type="text/css" href="/css/bootstrap.min.css">
    HTML

    message "Refresh your browser and you will now see something like this:"

    img src: "img/initial_bootstrap.png"
  end

  step do
    message "With the addition of Bootstrap, we can very quickly make our app look much more professional"
    message "For example, by adding a simple class to our Edit and Delete links, we can make them look like buttons:"

    source_code :php, <<-PHP
      echo "<a href='/edit.php?id=" .$topic['id']. "' class='btn btn-primary'>Edit</a>";
      echo " ";
      echo "<a href='/delete.php?id=" .$topic['id']. "' class='btn btn-danger'>Delete</a>";
    PHP
  end

  step do
    message "We can add a header bar:"

    source_code :html, <<-HTML
    <body>
      <nav class="navbar navbar-default" role="navigation">
        <div class="container-fluid">
          <div class="navbar-header">
            <a class="navbar-brand" href="#">Suggestotron</a>
          </div>
          </div>
        </nav>
        ...
      HTML
    end

    step do
      message 'We can add other things to our header, such as an "Add Topic" button:'

      source_code :html, <<-HTML
        <form class="navbar-form navbar-right" role="search">
          <a href="/add.php" class="btn btn-default">
            <span class="glyphicon glyphicon-plus-sign"></span>
            Add Topic
          </a>
        </form>
      HTML

      tip "There are many other Glyphicons, for example for Edit and Delete buttons, go ahead
and play around with them!"
    end

    step do
      message "Now we can make the rest of our page look nicer!"

      message "Bootstrap's grid system allows us to use rows and columns to layout our page"

      message "We are going to change our `foreach` to format each topic into a `<section>`
with rows and columns:"

      source_code :php, <<-PHP
        <div class="container">
          <?php
          foreach ($topics as $topic) {
                ?>
                <section>
                    <div class="row">
                        <div class="col-xs-12">
                            <h3><?=$topic['title'];?></h3>
                        </div>
                    </div>
                    <div class="row">
                        <div class="col-xs-8">
                            <p class="text-muted">
                                <?=nl2br($topic['description']);?>
                            </p>
                        </div>
                        <div class="col-xs-4">
                            <p class="pull-right">
                                <a href="/edit.php?id=<?=$topic['id']; ?>" class="btn btn-primary">Edit</a>
                                <a href="/delete.php?id=<?=$topic['id']; ?>" class="btn btn-danger">Delete</a>
                            </p>
                        </div>
                    </div>
                </section>
                <hr>
                <?php
            }
            ?>
        </div>
    PHP
```

message "Notice how we break out of the PHP tags for larger sections of HTML, and use short echo tags to display our data."

```
  end


  step do
```

message "The last thing we're going to do, is to add a warning for when people go to delete topics"

message "To use the Bootstrap javascript, we need to also include jQuery, which you can get from jquery.com."

message "Place the jquery Javascript file in the `public/js` directory."

message "We then include both Javascript files in our `<head>` section:"

```
    source_code :html, <<-HTML
      <script type="text/javascript" src="/js/jquery-2.1.0.min.js"></script>
      <script type="text/javascript" src="/js/bootstrap.min.js"></script>
    HTML
  end

  step do
    message "Now add the popover attributes to our Delete button:"

    source_code :php, <<-PHP
      <a href="/delete.php?id=<?=$topic['id']; ?>" class="btn btn-danger" data-container="body" data-toggle="popover" data-trigger="hover" data-placement="top" data-title="Are you sure?" data-content=" This cannot be undone!">Delete</a>
    PHP
  end

  step do
    message "Finally, initialize the popovers by adding this at the bottom of our file (right above the `</body>` tag):"

    source_code :html, <<-HTML
      <script type="text/javascript">
        $('[data-toggle="popover"]').popover();
      </script>
    HTML
  end
end
```

explanation do

message "By using off-the-shelf frameworks, Bootstrap 3 and jquery, we are able to quickly add a professional user interface to our application, drastically improving our user experience."

end

next_step "introducing_templates"

goals do

goal "Add templates to easily style all of Suggestotron"

goal "Learn about namespaces"

message "Templates allow us to re-use our work easily!"

end

steps do

step do

message "We are going to create a new class for handling our templates."

message "Now that we are adding more classes, let us create a directory especially for them:"

message "This directory should live in the root of our project, and is called `src`"

img src: "img/add_src_directory.png"

end

step do

message "Our template class will live in the `src/Suggestotron/Template.php` file:"

```
source_code :php, <<-PHP
    <?php
    namespace Suggestotron;

    class Template {
```

```
                    protected $base_template;
                    protected $page;

                    public function __construct($base_template)
                    {
                        $this->base_template = $base_template;
                    }

                    public function render($page, $data = array())
                    {
                        foreach ($data as $key => $value) {
                            $this->{$key} = $value;
                        }

                        $this->page = $page;
                        require $this->base_template;
                    }

                    public function content()
                    {
                        require $this->page;
                    }
                }
                ?>
        PHP
```

important "We are not the first people to create a `Template` class, to prevent it from conflicting with other peoples code, we use a namespace to make it unique. In this case, `Suggestotron`."

message "To refer to our `Template` class, we should now use its full name `\\Suggestotron\\Template`"

end

step do

message "Now that we have a specific place for them to live, we should move our `TopicData` class to `src/Suggestotron/TopicData.php`, don't forget to add our namespace:"

```
source_code :php, <<-PHP
<?php
namespace Suggestotron;

class TopicData {
PHP
```

message "Our `TopicData` class will now be `\\Suggestotron\\TopicData`"

img src: "img/topicdata_namespace.png"

end

step do

message "We can re-use a lot of our HTML on all our pages. Essentially, the only part unique to each page is the content inside the `container` div:"

```
source_code :html, <<-HTML
<div class="container">
        <!-- Specific Page Content Goes Here -->
</div>
HTML
```

message "We can split our HTML into re-usable templates, that will live in the `views` directory."

message "This directory should live in the root of our project, just like `src`."

message "We are going to create our template file, `views/base.phtml`, with the common HTML:"

tip "We use the `.phtml` extension to differentiate between regular PHP files, and templates files"

```
message "This includes our header, our container, and our footer"

source_code :html, <<-HTML
    <!doctype html>
    <html>
        <head>
            <title>Suggestotron</title>
            <link rel="stylesheet" type="text/css"
href="/css/bootstrap.min.css">
            <script type="text/javascript" src="/js/jquery-
2.1.0.min.js"></script>
            <script type="text/javascript"
src="/js/bootstrap.min.js"></script>
        </head>
        <body>
            <nav class="navbar navbar-default" role="navigation">
                <div class="container-fluid">
                <div class="navbar-header">
                    <a class="navbar-brand"
href="/">Suggestotron</a>
                </div>
                <form class="navbar-form navbar-right"
role="search">
                    <a href="/add.php" class="btn btn-
default">
                        <span class="glyphicon glyphicon-
plus-sign"></span>
                        Add Topic
                    </a>
                </form>
                </div>
            </nav>
            <div class="container">
                <!-- Specific Page Content Goes Here -->
                <?php $this->content(); ?>
```

```
                    </div>
                    <script type="text/javascript">
                        $('[data-toggle="popover"]').popover();
                    </script>
                </body>
            </html>
        HTML
    end

    step do
        message "We then take our `index.php` specific HTML and create `views/index/index.phtml`."

        source_code :php, <<-PHP
        <?php
        $topics = $this->topics;
        foreach ($topics as $topic) {
          ?>
          <section>
            <div class="row">
              <div class="col-xs-12">
                <h3><?=$topic['title'];?></h3>
              </div>
            </div>
            <div class="row">
              <div class="col-xs-8">
                <p class="text-muted">
                  <?=nl2br($topic['description']);?>
                </p>
              </div>
              <div class="col-xs-4">
                <p class="pull-right">
                  <a href="/edit.php?id=<?=$topic['id']; ?>" class="btn btn-primary">Edit</a>
```

```
                    <a href="/delete.php?id=<?=$topic['id']; ?>" class="btn btn-danger" data-container="body" data-toggle="popover" data-trigger="hover" data-placement="top" data-title="Are you sure?" data-content=" This cannot be undone!">Delete</a>
                  </p>
                </div>
              </div>
            </section>
            <hr>
            <?php
          }
          ?>
          PHP


          message "Our topic data, previously assigned to `$topics` is now assigned to `$this->topics` by our template class. The first thing we do in our template file is re-assign it to `$topics` so we can use it easily inside our template."
      end
      step do
          message "Now we can update our `public/index.php` to use our template:"


          source_code :php, <<-PHP
              <?php
              require '../src/Suggestotron/TopicData.php';
              require '../src/Suggestotron/Template.php';


              $data = new \\Suggestotron\\TopicData();


              $topics = $data->getAllTopics();


              $template = new \\Suggestotron\\Template("../views/base.phtml");
              $template->render("../views/index/index.phtml", ['topics' => $topics]);
              ?>
          PHP
```

tip "Notice how we pass the `$topics` variable into the `$template->render()` function. This allows our template to access the data we want it to. It will be accessible as `$this->topics` within the template."

end

step do

message "If we check our site right now, you'll likely see that it's still broken!"

fuzzy_result "Fatal error: Class 'suggestotron\\PDO' not found in <path>/suggestotron/src/Suggestotron/TopicData.php on line 15"

message "This is because the `TopicData` class is now inside the `Suggestotron` namespace, where the `PDO` class does not live"

message "To fix this, we must fully-qualify the PDO class in `\\Suggestotron\\TopicData->connect()`, by prefixing it with a `\\`:"

```
source_code :php, <<-PHP
public function connect()
{
    $this->connection = new \\PDO("mysql:host=192.168.1.10;dbname=suggestotron", "root", null);
}
PHP
```

message "Additionally, we need to prefix the `\\PDO::FETCH_ASSOC` constant passed to `$query->fetch()` in `\\Suggestotron\\TopicData->getTopic()`:"

```
source_code :php, <<-PHP
return $query->fetch(\\PDO::FETCH_ASSOC);
PHP
```

end

step do

message "Now that our templates are working, lets add them to our other pages!"

```
message "For `add.php`, we will create a `views/index/add.phtml`:"

source_code :html, <<-HTML
          <h2>New Topic</h2>
          <form action="add.php" method="POST">
            <label>
              Title: <input type="text" name="title">
            </label>
            <br>
            <label>
              Description:
              <br>
              <textarea name="description" cols="50" rows="20"></textarea>
            </label>
            <br>
            <input type="submit" class="btn btn-primary" value="Add Topic">
          </form>
     HTML

     message "Then add the templates to `add.php`:"

     source_code :php, <<-PHP
          <?php
          require '../src/Suggestotron/TopicData.php';
          require '../src/Suggestotron/Template.php';

          if (isset($_POST) && sizeof($_POST) > 0) {
            $data = new \\Suggestotron\\TopicData();
            $data->add($_POST);
            header("Location: /");
            exit;
          }

          $template = new \\Suggestotron\\Template("../views/base.phtml");
```

```
                    $template->render("../views/index/add.phtml");
                    ?>
            PHP
    end


    step do
            message "Now, do the same for `edit.php`. For `edit.php`, we will create a
`views/index/edit.phtml`:"


    source_code :php, <<-HTML
                    <?php
                    $topic = $this->topic;
                    ?>
                    <h2>Edit Topic</h2>
                    <form action="edit.php" method="POST">
                      <label>
                        Title: <input type="text" name="title" value="<?=$topic['title'];?>">
                      </label>
                      <br>
                      <label>
                        Description:
                        <br>
                        <textarea name="description" cols="50"
rows="20"><?=$topic['description'];?></textarea>
                      </label>
                      <br>
                      <input type="hidden" name="id" value="<?=$topic['id'];?>">
                      <input type="submit" class="btn btn-primary" value="Edit Topic">
                    </form>
            HTML


            message "Then `edit.php` needs to use the templates:"


            source_code :php, <<-PHP
```

```
                <?php
                require '../src/Suggestotron/TopicData.php';
                require '../src/Suggestotron/Template.php';

                if (isset($_POST['id']) && !empty($_POST['id'])) {
                    $data = new \\Suggestotron\\TopicData();
                    if ($data->update($_POST)) {
                        header("Location: /index.php");
                        exit;
                    } else {
                        echo "An error occurred";
                    }
                }

                if (!isset($_GET['id']) || empty($_GET['id'])) {
                    echo "You did not pass in an ID.";
                    exit;
                }

                $data = new \\Suggestotron\\TopicData();
                $topic = $data->getTopic($_GET['id']);

                if ($topic === false) {
                    echo "Topic not found!";
                    exit;
                }

                $template = new \\Suggestotron\\Template("../views/base.phtml");
                $template->render("../views/index/edit.phtml", ['topic' => $topic]);
                ?>
            PHP
end

step do
```

message "For `delete.php`, as we don't actually have any output, we just need to update the `TopicData` class"

```
            source_code :php, <<-PHP
                <?php
                require_once '../src/Suggestotron/TopicData.php';

                if (!isset($_GET['id']) || empty($_GET['id'])) {
                    echo "You did not pass in an ID.";
                    exit;
                }

                $data = new \\Suggestotron\\TopicData();
                $topic = $data->getTopic($_GET['id']);

                if ($topic === false) {
                    echo "Topic not found!";
                    exit;
                }

                if ($data->delete($_GET['id'])) {
                        header("Location: /index.php");
                    exit;
                } else {
                    echo "An error occurred";
                }
                ?>
            PHP
        end
end
```

explanation do

message "By adding templates, we can easily make our sites styles consistent. Additionally, we stop ourselves from repeating the same HTML code everywhere, and can change it in one place and effect all of our pages at once!"

end

next_step "autoloading"

situation "We keep repeating ourselves!" do

message 'In each of our files we have these repetetive `require "../src/Suggestotron/*.php"` lines. What if we could get rid of them?'

end

goals do

goal "Make our code simpler, and easier to write, using autoloading"

message "An autoloader will automatically load your classes when you need them!"

end

steps do

step do

message "An autoloader is just a simple function that will automatically find and load a class based on its name"

message "We will place our autoloader in `src/Suggestotron/Autoloader.php`:"

source_code :php, <<-PHP

```
<?php
namespace Suggestotron;

class Autoloader {
    public function load($className)
    {
        $file = __DIR__ . "/../" . str_replace("\\\\", "/", $className) . '.php';

        if (file_exists($file)) {
            require $file;
```

```
                    } else {
                        return false;
                    }
                }
                public function register()
                {
                    spl_autoload_register([$this, 'load']);
                }
            }

            $loader = new Autoloader();
            $loader->register();
        PHP
```

important "This autoloader is as simple as possible, but will *not* handle every situation. We highly recommend learning about [Composer](http://getcomposer.org) as it will automate most of this for you!"

end

step do

message "We can replace all our other `require` statements, with a single `require` for the autoloader itself:"

```
        source_code :php, <<-PHP
            require '../src/Suggestotron/Autoloader.php';
        PHP
```

message "Make this change in `index.php`, `add.php`, `edit.php` and `delete.php`."

end

end

explanation do

message "Now that we have an autoloader, every time we use `new \\Some\\ClassName` it will try to autoload by passing the name to the `\\Suggestotron\\Autoloader->load()` method automatically."

end

next_step "configuration"

goals do

goal "Create a simple, global, configuration file to allow you to easily customize your application"

message "A configuration file is critical for allowing you to do things like moving your site between servers"

end

steps do

step do

message "The first thing we will do is add a `config` directory, at the same level as `src` and `views`, and place an `autoload.php` file within it:"

```
source_code :php, <<-PHP
        <?php
        return [
                'class_path' => realpath('../src')
        ];
        ?>
PHP
```

end

step do

message "Next, we will create our `\\Suggestotron\\Config` class in `src/Suggestotron/Config.php`:"

```
source_code :php, <<-PHP
        <?php
        namespace Suggestotron;

        class Config {
```

```
                    static public $directory;
                    static public $config = [];

                    static public function setDirectory($path)
                    {
                        self::$directory = $path;
                    }

                    static public function get($config)
                    {
                        $config = strtolower($config);

                        self::$config[$config] = require self::$directory . '/' .
$config . '.php';

                        return self::$config[$config];
                    }
                }
        PHP
```

message "To use the `\\Suggestotron\\Config` class we must still include it manually, to setup everything else"

```
source_code :php, <<-PHP
<?php
require_once '../src/Suggestotron/Config';
\\Suggestotron\\Config::setDirectory('../config');
?>
PHP
```

message "Once you have done this, the configuration is available everywhere using:"

```
source_code :php, <<-PHP
$config = \\Suggestotron\\Config::get('autoload');
```

```
        PHP
    end


    step do
        message "Next, we will update our autoloader to use the configuration settings:"


        source_code :php, <<-PHP
        $config = \\Suggestotron\\Config::get('autoload');


        $file = $config['class_path'] . '/' . str_replace("\\\\", "/", $className) . '.php';
        PHP
    end


    step do
        message "Then, update each of our `index.php`, `add.php`, `edit.php`,
`delete.php` files to use the config:"


        source_code :php, <<-PHP
        <?php
        require_once '../src/Suggestotron/Config.php';
        \\Suggestotron\\Config::setDirectory('../config');


        $config = \\Suggestotron\\Config::get('autoload');
        require_once $config['class_path'] . '/Suggestotron/Autoloader.php';
        PHP
    end


    step do
        message "Other configuration options, might be your database, for example.
Now create a `config/database.php`:"


        source_code :php, <<-PHP
        <?php
        return [
                    "username" => "root",
                    "password" => "root",
                    "hostname" => "localhost",
                    "dbname" => "suggestotron",
                ];
                ?>
                PHP
        end

        step do
                message "Then just update our `\\Suggestotron\\TopicData` class to use the configuration:"

                source_code :php, <<-PHP
                public function connect()
                {
                   $config = \\Suggestotron\\Config::get('database');

                   $this->connection = new \\PDO("mysql:host=" .$config['hostname']. ";dbname=" .$config['dbname'], $config['username'], $config['password']);
                }
      PHP
        end

        step do
                message "Let us add one more configuration file, for customizing our Suggestotron, `config/site.php`:"

                source_code :php, <<-PHP
                <?php
                return [
                        "title" => "Suggestotron"
                ];
                ?>
                PHP
```

```
    end

    step do
        message "Finally, lets update our base template, `views/base.phtml`:"

        message "At the top, first get the configuration:"
        source_code :php, <<-PHP
        <?php
        $config = \\Suggestotron\\Config::get('site');
        ?>
        <!doctype html>
        ...
        PHP

        message "Then, update the `<title>` to use the configuration option:"
        source_code :php, <<-PHP
            <title><?=$config['title'];?></title>
        PHP

        message "Now, if you change the configuration file, your page title will automatically update everywhere. Go ahead, play!"
    end
end
```

next_step "pretty_urls"

```
goals do
    goal 'Get rid of the unsightly ".php" in our URLs, modernizing our app!'
    goal 'Reduce duplication of code'

    message "Modern web applications use magical URLs that don't map 1:1 with files, to make them more dynamic and maintainable."
end
```

important "Different web servers (e.g. Nginx, Apache) must be configured differently for this to work, but the most popular ones all support it"

situation "Dynamic URLs" do

message "By default with our PHP server, if we enter a URL that does not exist, we are sent to `index.php`"

message "We can then look at the `$_SERVER` super-global to find out the page they requested. This value lives in `$_SERVER['PATH_INFO']`"

message "For example, if we visit <http://localhost:8080/add>, `$_SERVER['PATH_INFO']` is set to `/add`"

end

steps do

step do

message "We will start by creating a Router class. This will take the dynamic URL and map it to our application code:"

```
source_code :php, <<-PHP
<?php
namespace Suggestotron;

class Router {
	public function start($route)
	{
		$path = realpath("./" . $route . ".php");
		if (file_exists($path)) {
			require $path;
		} else {
			require 'error.php';
		}
	}
}
PHP
```

```
                message "This will look for a file with the same name as the route and include it, or include an `error.php` file"

        end


        step do

                message "Now we need to re-purpose our `index.php` to use our router, instead of simply showing our list. First, we will move our list to `list.php`:"


                source_code :php, <<-PHP

                        <?php

                        require_once '../src/Suggestotron/Config.php';

                        \\Suggestotron\\Config::setDirectory('../config');


                        $config = \\Suggestotron\\Config::get('autoload');

                        require_once $config['class_path'] . '/Suggestotron/Autoloader.php';


                        $data = new \\Suggestotron\\TopicData();


                        $topics = $data->getAllTopics();


                        $template = new \\Suggestotron\\Template("../views/base.phtml");

                        $template->render("../views/index/list.phtml", ['topics' => $topics]);

                        ?>

                PHP

        end


        step do

                message "We then move our template from `views/index/index.phtml` to `views/index/list.phtml`, and update our links to point to `/edit` and `/delete`:"


                source_code :php, <<-PHP

                        <a href="/edit?id=<?=$topic['id']; ?>" class="btn btn-primary">Edit</a>

                        <a href="/delete?id=<?=$topic['id']; ?>" class="btn btn-danger" data-container="

                PHP
                message "Also, we should update our link to `add.php` in the base template (`base.phtml`):"

                source_code :html, <<-HTML
                        <a href="/add" class="btn btn-default">
                            <span class="glyphicon glyphicon-plus-sign"></span>
                            Add Topic
                        </a>
                HTML

                message "Finally, update the `<form>` actions in `views/index/add.phtml` and `views/index/edit.phtml`, to point to the correct URLs."
        end

        step do
                message "Then, we can make the necessary changes to our `index.php`:"

                source_code :php, <<-PHP
                <?php
                require_once '../src/Suggestotron/Config.php';
                \\Suggestotron\\Config::setDirectory('../config');
                $config = \\Suggestotron\\Config::get('autoload');
                require_once $config['class_path'] . '/Suggestotron/Autoloader.php';

                if (!isset($_SERVER['PATH_INFO']) || empty($_SERVER['PATH_INFO']) || $_SERVER['PATH_INFO'] == '/') {
                        $route = 'list';
                } else {
                        $route = $_SERVER['PATH_INFO'];
                }

                $router = new \\Suggestotron\\Router();
                $router->start($route);
                ?>
                PHP
        end

        step do
                message "Now, if you visit the site, and click around, you will see our URLs are much nicer"

                message "Additionally, we can start to remove some duplicated code, we no longer need to setup our config and our autoloader in each of these files"

                message "Go ahead and remove the following, and you'll see the site still works!"

                source_code :php, <<-PHP
                        require_once '../src/Suggestotron/Config.php';
                        \\Suggestotron\\Config::setDirectory('../config');

                        $config = \\Suggestotron\\Config::get('autoload');
                        require_once $config['class_path'] . '/Suggestotron/Autoloader.php';
                PHP
        end
end

explanation do
    message "By programmitically handling our URLs, we can create pretty URLs in any structure we want, without needing to create complex directory structures."

    message "This allows us to share common code between many pages — similiar to our templates — and reduce our applications complexity."
end
```

next_step "getting_dry"

```
situation "DRY — Don't Repeat Yourself" do
```

message "As developers we try not to repeat ourselves. As you've seen, doing similar things multiple times, means that we have update multiple places when we want to make changes."

message "By trying to be more DRY, we reduce the number of places where changes need to be made."

end

goals do

goal "Make our code more DRY by combining similar functionality"

end

situation "Introducing Controllers" do

message "If you try to use one of our old URLs (e.g. <http://localhost:8080/add.php>) you will notice it is now broken!"

message "Because we now have a router, our users should not be accessing these files directly anymore — we *can* solve this by moving the files out of the `public` directory."

message "**However**, there is another way: Controllers — a special class for containing functionality relating to a specific thing — like Topics."

end

steps do

step do

message "Our controller — which could be one of many — will live in the class, `\\Suggestotron\\Controller\\Topics`, and will have one method for each action (list, add, edit, delete) that our application has:"

```
source_code :php, <<-PHP
    <?php
    namespace Suggestotron\\Controller;

    class Topics {
        public function listAction()
        {
        }

        public function addAction()
        {

        }

        public function editAction()
        {

        }

        public function deleteAction()
        {

        }
    }
    ?>
PHP
```

message "Our router will call these methods, instead of including our `.php` files."

end

step do

message "Next, we can migrate the contents of our `.php` files to their respective methods. For example, the `listAction()` would look like this:"

```
            source_code :php, <<-PHP
                public function listAction() {
                    $data = new \\Suggestotron\\TopicData();

                    $topics = $data->getAllTopics();
                    $template = new \\Suggestotron\\Template("../views/base.phtml");
                    $template->render("../views/index/list.phtml", ['topics' => $topics]);
               }
          PHP
     end

     step do
          message "Once you have completed all the methods, you will notice there is a lot of repeated code, specifically:"

          source_code :php, <<-PHP
               $data = new \\Suggestotron\\TopicData();
          PHP

          message "and:"
          source_code :php, <<-PHP
               $template = new \\Suggestotron\\Template("../views/base.phtml");
          PHP

          message "To remove this duplication, we can move those lines to their own method, and assign the objects to properties."

          message "Because **all** of our actions need this, we can do it automatically in the special `__construct()` method:"

          source_code :php, <<-PHP
          protected $data;
          protected $template;

          public function __construct()
          {
               $this->data = new \\Suggestotron\\TopicData();
               $this->template = new \\Suggestotron\\Template("../views/base.phtml");
          }
```

```
        PHP


        message "Now we just update the actions, to remove those duplicated lines, and instead of `$data` or `$template` we use `$this->data` and `$this->template` respectively."


        message "For example, our `addAction` will look like this:"


        source_code :php, <<-PHP
            public function addAction()
            {
                if (isset($_POST) && sizeof($_POST) > 0) {
                  $this->data->add($_POST);
                  header("Location: /");
                      exit;
                }

                $this->template->render("../views/index/add.phtml");
            }
        PHP


        message "Notice how small it is now! Just 6 lines of code!"
    end


    step do
        message "Additionally, we have a lot of duplicate paths like `../views/`. Like our Autoloader, we should put this in our configuration."


        message "Rather than add a whole new file for this, let's just add it to our `config/site.php`:"


        source_code :php, <<-PHP
            <?php
            return [
                'title' => 'Suggestotron!',
                'view_path' => realpath('../views')
                ];
                ?>
        PHP
```

message "You probably already noticed that our calls the `$this->template->render()` are very similar in each action."

message "We can create a new helper method to simplify this:"

```
        source_code :php, <<-PHP
        protected function render($template, $data = array())
        {
                $config = \\Suggestotron\\Config::get('site');

                $this->template->render($config['view_path'] . "/" . $template, $data);
        }
        PHP
```

message "Now update your actions to use the new method instead of `$this->template->render():`"

```
        source_code :php, <<-PHP
                public function listAction()
                {
                        $topics = $this->data->getAllTopics();

                        $this->render("index/list.phtml", ['topics' => $topics]);
                }
        PHP
    end

    step do
```

message "The last thing we need to do is to update the path to the base template, because this needs the config also, lets move that to a property and update our `__construct()` method:"

```
        source_code :php, <<-PHP
            protected $config;


            public function __construct()
            {
                $this->config = \\Suggestotron\\Config::get('site');
                $this->data = new \\Suggestotron\\TopicData();
                $this->template = new \\Suggestotron\\Template($this->config['view_path'] . "/base.phtml");
            }
        PHP

        message "Don't forget to replace `$config` with `$this->config` in our `render()` method."
    end

    step do
        message "To use our controller, we just update our router:"

        source_code :php, <<-PHP
            public function start($route)
            {
                // If our route starts with a /, remove it
                if ($route{0} == "/") {
                    $route = substr($route, 1);
                }

                $controller = new \\Suggestotron\\Controller\\Topics();

                $method = [$controller, $route . 'Action'];

                if (is_callable($method)) {
```

```
                                        return $method();
                                }

                                require 'error.php';
                        }
                PHP
        end

        step do
                message "Finally, if you didn't already, you can remove the old `.php` files (except `index.php`!)"
        end
end

explanation do
        message "By implementing a Controller we have further simplified our code. It should be a goal to keep your code as simple as possible: Future you will thank you."
end
```

next_step "multiple_controllers"

```
situation "Multiple Features : Multiple Controllers" do
        message "As our application grows, we can continue to add more actions
        to our `\\Suggestotron\\Controller\\Topics` controller but this would make
        it harder to maintain."

        message "To help our future selves, we should allow for us to separate our features in to multiple controllers."

        message "This means our router needs to be able to tell which controller is being requested, and to call the correct one."
end

goals do
        goal "Allow our app to grow using multiple controllers"
        goal "Automatically route to controllers and actions"
        goal "Allow easy setup of routes via configuration"
end

steps do
        step do
                message "We are going to start with the configuration as it will determine how
our code needs to work."

                message "Our configuration needs to determine several things:"

                ul {
                        li {
                                text "The URL to match"
                        }

                        li {
                                text "The default action, if none is specified"
                        }

                        li {
                                text "The default controller, if none is specified"
                        }

                        li {
                                text "An error controller for when an error is encountered"
                        }
                }

                message "Our configuration file, `routes.php`, might look like this:"

                source_code :php, <<-PHP
                        <?php
                        return [
                    'default' => '/topic/list',
                    'errors' => '/error/:code',
                    'routes' => [
                        '/topic(/:action(/:id))' => [
                            'controller' => '\\Suggestotron\\Controller\\Topics',
                            'action' => 'list',
                        ],
                        '/:controller(/:action)' => [
                            'controller' => '\\Suggestotron\\Controller\\:controller',
                            'action' => 'index',
                        ]
                    ]
                ];
        PHP
```

message "Here we have defined two routes. Within the path specified, we have variables, which if specified in the URL, will replace the default values within each route."

message "In our first route, `/topic(/:action(/:id))`, if a user browses to `/topic/add`, then the `action` will be set to `add`. If they go to just `/topic`, it will be set to the default, `list`."

message "In our second route, we have two placeholders, `:controller`, and `:action`. This means that we can now dynamically choose the controller based on the route itself."

message "If the were to browse to `/vote`, it will use the `\\Suggestotron\\Controller\\Vote` controller, and call the default `index` action."

end

step do

message "Now that we have our config, we can use it to re-write our `\\Suggestotron\\Router->start()` method:"

```
        source_code :php, <<-PHP
        class Router {
```

```
                    protected $config;

                    public function start($route)
                    {
                            $this->config = \\Suggestotron\\Config::get('routes');
            PHP
```

message "In our config we defined a default, so our first step is to check if we need to use it:"

```
            source_code :php, <<-PHP
                    if (empty($route) || $route == '/') {
                            if (isset($this->config['default'])) {
                                    $route = $this->config['default'];
                            } else {
                                    $this->error();
                            }
                    }
            PHP
    end

    situation "Try... Catch" do
```

message "To help with error handling, we can wrap our code in a `try { } catch { }` block. Whenever an error is encountered, the code within the `catch { }` block is run instead."

```
    end

    step do
```

message "We are going to use a `try... catch` around our routing, in case something goes wrong!"

```
            source_code :php, <<-PHP
            try {

            } catch (\\Suggestotron\\Controller\\Exception $e) {
}
PHP
```

message "Inside our `try` block, we will iterate over each of the defined routes, trying to find a match for the URL:"

tip "Here we are using a regular expression with `preg_replace()` and `preg_match()`.

Regular expressions is a way to match patterns in text. *We are using a complicated one here, so don't worry if you don't yet understand it!*"

```
source_code :php, <<-PHP
try {
        foreach ($this->config['routes'] as $path => $defaults) {
                $regex = '@' . preg_replace(
        '@:([\\w]+)@',
        '(?P<$1>[^/]+)',
        str_replace(')', ')?', (string) $path)
      ) . '@';
                $matches = [];
                if (preg_match($regex, $route, $matches)) {
PHP
```

message "If we find a match, we merge the defaults from our config, with the values specified in the URL:"

```
source_code :php, <<-PHP
                        $options = $defaults;
                        foreach ($matches as $key => $value) {
                                if (is_numeric($key)) {
                                        continue;
                                }
                                $options[$key] = $value;
                                if (isset($defaults[$key])) {
```

```
                            if (strpos($defaults[$key], ":$key") !==
false) {
                                $options[$key] =
str_replace(":$key", $value, $defaults[$key]);
                            }
                        }
                    }
        PHP

        message "Then finally, we check that we have a controller and action, and if
valid, we call it:"

        source_code :php, <<-PHP
                    if (isset($options['controller']) && isset($options['action']))
{
                        $callable = [$options['controller'],
$options['action'] . 'Action'];
                        if (is_callable($callable)) {
                            $callable = [new $options['controller'],
$options['action'] . 'Action'];
                            $callable($options);
                            return;
                        } else {
                            $this->error();
                        }
                    } else {
                        $this->error();
                    }
                }
            }
        }
        PHP
    end

    step do
        message "We then call `$this->error()` in our `catch` block:"
        source_code :php, <<-PHP
        catch (\\Suggestotron\\Controller\\Exception $e) {
            $this->error();
        }
        PHP
    end

    step do
        message "We must also define the `\\Suggestotron\\Router->error()` method:"

        source_code :php, <<-PHP
        public function error()
        {
            if (isset($this->config['errors'])) {
                $route = $this->config['errors'];
                $this->start($route);
            } else {
                echo "An unknown error occurred, please try again!";
            }

            exit;
        }
        PHP
    end

    step do
        message "Now that we have a configured default, we should update `index.php` to no-longer handle this:"

        message "Replace the following:"
        source_code :php, <<-PHP
            if (!isset($_SERVER['PATH_INFO']) || empty($_SERVER['PATH_INFO']) || $_SERVER['PATH_INFO'] == '/') {
                $route = 'list';
            } else {
                $route = $_SERVER['PATH_INFO'];
            }
        PHP

        message "With this:"

        source_code :php, <<-PHP
        $route = null;
        if (isset($_SERVER['PATH_INFO'])) {
            $route = $_SERVER['PATH_INFO'];
        }
        PHP
    end

    step do
        message "Our first new controller, is going to be our error controller, `\\Suggestotron\\Controller\\Errors`, **however**, now that we will have multiple controllers, this is a good time to refactor again!"

        message "We will first create a base controller, `\\Suggestotron\\Controller`:"

        source_code :php, <<-PHP
        <?php
        namespace Suggestotron;
        class Controller {
            protected $config;
            protected $template;

            public function __construct()
            {
                $this->config = \\Suggestotron\\Config::get('site');
                $this->template = new \\Suggestotron\\Template($this->config['view_path'] . "/base.phtml");
                }


                protected function render($template, $data = array())
                {
                        $this->template->render($this->config['view_path'] . "/" .
$template, $data);
                }
        }
        PHP
```

message "Here we have consolidated our common constructor, and our `render()` methods that all controllers will need."

end

step do

message "Our error controller, will then `extend` our base controller, which means that it will inherit all of it's properties and methods."

```
        source_code :php, <<-PHP
        <?php
        namespace Suggestotron\\Controller;

        class Error extends \\Suggestotron\\Controller {
                public function indexAction($options)
                {
                        header("HTTP/1.0 404 Not Found");
                        $this->render("/errors/index.phtml", ['message' => "Page not
found!" ]);
                }
        }
        ?>
        PHP
```

message "This simple controller sends the 404 error header, and then renders the appropriate view, `errors/index` which looks like this:"

```
        source_code :php, <<-PHP
        <div class="alert alert-danger">
            <?=$this->message;?>
        </div>
        PHP
    end
```

    step do

message "We can now take advantage of the options passed to the action via the URL, making our URLs even prettier!"

message "We need to update `\\Suggestotron\\Controller\\Topics` so that each action can take an argument, `$options`, and for our edit/delete methods, we can now switch to using `$options['id']` instead of `$_GET['id']`."

message "Additionally, be sure to correct any `header()` redirects, to point to the new locations:"

message "For example, the delete action, will look like this:"

```
        source_code :php, <<-PHP
        public function deleteAction($options)
        {
            if (!isset($options['id']) || empty($options['id'])) {
                echo "You did not pass in an ID.";
                exit;
            }

            $topic = $this->data->getTopic($options['id']);

            if ($topic === false) {
                echo "Topic not found!";
                exit;
            }
```

```
                        if ($this->data->delete($options['id'])) {
                                header("Location: /");
                           exit;
                        } else {
                           echo "An error occurred";
                        }
                }
                PHP
        end

        situation "Prettier URLs" do
                message "With these new changes, our URLs are now as follows:"

                message <<-MD
                        - **List Topics:** <http://localhost:8080/> *or* <http://localhost:8080/topic/list>
                        - **New Topic:** <http://localhost:8080/topic/add>
                        - **Edit Topic:** <http://localhost:8080/topic/edit/1> (where `1` is our topic ID)
                        - **Delete Topic:** http://localhost:8080/topic/delete/1 (where `1` is our topic ID)
                MD

                message "We should update our views, to reflect these new URLs."
        end

        step do
                message "In our `base.phtml`, our *Add Topic* link, should now point to `/topic/add`:"

                source_code :html, <<-HTML
                        <a href="/topic/add" class="btn btn-default">
                                <span class="glyphicon glyphicon-plus-sign"></span>
                                Add Topic
```

```
				</a>
			HTML
	end

	step do
			message "In our `index/list.phtml`, our links should be updated:"

			source_code :php, <<-PHP
				<a href="/topic/edit/<?=$topic['id']; ?>" class="btn btn-primary">Edit</a>
				<a href="/topic/delete/<?=$topic['id']; ?>" class="btn btn-danger" data-container="body" data-toggle="popover" data-trigger="hover" data-placement="top" data-title="Are you sure?" data-content=" This cannot be undone!">Delete</a>
			PHP

			message "Notice how we now use `/topic/<action>/<id>` as our URL, *no more `GET` arguments!*"
	end

	step do
			message "Almost there! We just need to update our `<form>` tags."

			message "In `index/add.phtml`:"

			source_code :html, <<-HTML
				<form action="/topic/add" method="POST">
			HTML

			message "In `index/edit.phtml`:"

			source_code :html, <<-HTML
			<form action="/topic/edit" method="POST">
			HTML
	end
```

```
    step do
        message "Our final step, is to update our existing controller, `\\Suggestotron\\Controller\\Topics`, to use the new base controller:"

        message "Just like with `\\Suggestotron\\Controller\\Errors`, we `extend` the base controller."

        source_code :php, <<-PHP
        class Topics extends \\Suggestotron\\Controller {
        PHP

        message "Then we can start removing the now-duplicated code."

        source_code :php, <<-PHP
            protected $template;
            protected $config;
        PHP
        message "Our constructor can be simplified too:"

        source_code :php, <<-PHP
            public function __construct()
            {
                parent::__construct();
                $this->data = new \\Suggestotron\\TopicData();
            }
        PHP

        tip "We use a special method called, `parent::__construct()` to call the `\\Suggestotron\\Controller->__construct()` method."

        message "We can also remove the `render()` function entirely."
    end
end

explanation do
```

message "By adding the ability for multiple controllers, we have given ourselves a structure in to which we can continue to add new features to our application easily."

end

next_step "introducing_models"

message "A `model` is just a fancy name for a class that specifically encapsulates all functionality related to a thing, e.g. topics, votes, or users."

message "Our `\\Suggestotron\\TopicData` class, is an example of a model class."

situation "Managing Database Connections" do

message "Currently, we create the database connection every time we instantiate `\\Suggestotron\\TopicData`.

However, what if we want multiple instances of the object? What we need the database connection in other models?"

message "We should instead, have a single way to create a single shared connection, that any object can easily re-use."

end

goals do

goal "Refactor our database connection code, so we can re-use the connections in many places"

end

steps do

step do

message "We are going to create what is known as a singleton class, which is responsible for managing our connection."

message "We will call this class, `\\Suggestotron\\Db`."

source_code :php, <<-PHP

<?php

namespace Suggestotron;

```
                    class Db {
                        static protected $instance = null;

                        protected $connection = null;
                        protected function __construct() {
                      $config = \\Suggestotron\\Config::get('database');

                      $this->connection        =        new        \\PDO("mysql:host="
.$config['hostname'].       ";dbname="       .$config['dbname'],       $config['username'],
$config['password']);
                        }

                        public function getConnection()
                        {
                            return $this->connection;
                        }

                        static public function getInstance()
                        {
                            if (!(static::$instance instanceof static)) {
                                static::$instance = new static();
                            }

                            return static::$instance->getConnection();
                        }
                    }
                PHP
```

message "This class sets `__construct()` to protected, which means that it cannot be instantiated outside of this class (or it's children) and therefore requires the use of `\\Suggestotron\\Db::getInstance()` to create a new object."

message "`\\Suggestotron\\Db::getInstance()` will check for an existing copy and return that instead if one exists. Otherwise, it creates and stores a new instance."

end

step do

message "To use our new class, we simply replace all instances of `$this->connection` with `\\Suggestotron\\Db::getInstance()` in our `\\Suggestotron\\TopicData` class, and remove the existing database connection code."

message "We can completely remove the following code:"

```
source_code :php, <<-PHP
protected $connection = null;

public function __construct()
{
    $this->connect();
}

public function connect()
{
    $config = \\Suggestotron\\Config::get('database');

    $this->connection = new \\PDO("mysql:host=" .$config['hostname']. ";dbname=" .$config['dbname'], $config['username'], $config['password']);
}
PHP
```

message "Then, our `getAllTopics()` method for instance, will look something like this:"

```
source_code :php, <<-PHP
public function getAllTopics()
{
    $query = \\Suggestotron\\Db::getInstance()->prepare("SELECT * FROM topics");
    $query->execute();
    return $query;
}
PHP
end
```

step do

message "To better organize our code, we're going to rename our `\\Suggestotron\\TopicData` class to identify it as a model, in the same way we do controllers. Therefore, it will be called `\\Suggestotron\\Model\\Topics`."

message "First, we will move the file to `/Suggestotron/Model/Topics.php`, and then we will update the namespace, and the class name:"

```
source_code :php, <<-PHP
    <?php
    namespace Suggestotron\\Model;

    class Topics {
    PHP

    message "Finally, update our `\\Suggestotron\\Controller\\Topics` to use our renamed class."

    source_code :php, <<-PHP
    class Topics extends \\Suggestotron\\Controller {
        protected $data;

        public function __construct()
        {
            parent::__construct();
            $this->data = new \\Suggestotron\\Model\\Topics();
        }
    PHP
end
```

explanation do

message <<-MD

- You can now access the database connection from **anywhere** using `\\Suggestotron\\Db::getInstance()`.

- Also, our models are now consistently named, similar to controllers.

```
            MD
    end
end
```

next_step "completing_suggestotron"

```
situation "Complete Suggestotron" do
      message "Suggestrotron is not really complete unless we can rank suggestions by popular vote."

      message "We're going to use all of the new skills we've learned, to build out this new feature."
end

goals do
      goal "Add voting to Suggestotron"
end

steps do
      step do
            message "Just like with our topics, we will start out by defining our database table, `votes`:"

            model_diagram header: 'votes', fields: %w(id title_id count)

            message "To get started, we run the `mysql` command in the terminal:"

            console "mysql -u root -p"

            message "When prompted, enter the password: `root`."
      end

      step do
            message "We then run our SQL code:"
```

```
        source_code :sql, <<-SQL
        USE suggestotron;

        CREATE TABLE votes (
                id INT unsigned NOT NULL AUTO_INCREMENT,
                topic_id INT unsigned NOT NULL,
                count INT NOT NULL DEFAULT 0,
                PRIMARY KEY(id)
        );
        SQL
    end

    step do
        message "Create empty vote records for each of your existing topics:"

        source_code :sql, <<-SQL
        INSERT INTO votes (
                topic_id,
                count
        ) SELECT id, 0 FROM topics;
        SQL

        message "Verify our data:"

        source_code :sql, <<-SQL
        SELECT * FROM votes;
        SQL

        fuzzy_result <<-TEXT
        SELECT * FROM votes;
        +----+----------+-------+
        | id | topic_id | count |
        +----+----------+-------+
        |  1 |        1 |     0 |
        | 2 |       2 |    0 |
        | 3 |       3 |    0 |
        +----+----------+-------+
        3 rows in set (0.00 sec)
        TEXT
    end


    step do
        message "Update our Topics model class, to insert an empty row when creating new topics automatically:"


        source_code :php, <<-PHP
        public function add($data)
        {
            $query = \\Suggestotron\\Db::getInstance()->prepare(
              "INSERT INTO topics (
                  title,
                  description
              ) VALUES (
                  :title,
                  :description
              )"
            );

            $data = [
              ':title' => $data['title'],
              ':description' => $data['description']
            ];

            $query->execute($data);

            // Grab the newly created topic ID
            $id = \\Suggestotron\\Db::getInstance()->lastInsertId();
```

```
                        // Add empty vote row
                        $sql = "INSERT INTO votes (
                                                topic_id,
                                                count
                                        ) VALUES (
                                                :id,
                                                0
                                        )";

                        $data = [
                                ':id' => $id
                        ];

                        $query = \\Suggestotron\\Db::getInstance()->prepare($sql);
                        $query->execute($data);
                }
   PHP
end

step do
     message "We must also remove this data, when deleting the topic:"

     source_code :php, <<-PHP
     public function delete($id) {
     $query = \Suggestotron\Db::getInstance()->prepare(
        "DELETE FROM topics
           WHERE
              id = :id"
     );

     $data = [
        ':id' => $id,
     ];
     $result = $query->execute($data);
```

```
         if (!$result) {
            return false;
         }

         $sql = "DELETE FROM votes WHERE topic_id = :id";
         $query = \\Suggestotron\\Db::getInstance()->prepare($sql);

         return $query->execute($data);
      }
      PHP
   end

         step do
                  message "We now need a model class to manage our votes,
`\\Suggestotron\\Model\\Votes`:"

                  source_code :php, <<-PHP
                  <?php
                  namespace Suggestotron\\Model;

                  class Votes {
                           public function addVote($topic_id)
                           {
                                    $sql = "UPDATE votes
                        SET
                           count = count + 1
                        WHERE
                           topic_id = :id";

                        $query = \\Suggestotron\\Db::getInstance()->prepare($sql);
                        $data = [
                           ':id' => $topic_id,
                        ];
                        return $query->execute($data);
                          }
                }
                PHP
        end

        step do
                message "Next up, we create our controller, `\\Suggestotron\\Controller\\Votes`, with an `add` action."

                source_code :php, <<-PHP
                        <?php
                        namespace Suggestotron\\Controller;

                        class Votes extends \\Suggestotron\\Controller {
                                public function addAction($options) {
                                        if (!isset($options['id']) || empty($options['id'])) {
                                                echo "No topic id specified!";
                                                exit;
                                        }

                                        $votes = new \\Suggestotron\\Model\\Votes();
                                        $votes->addVote($options['id']);

                                        header("Location: /");
                                }
                        }
                PHP
        end

        step do
                message "To access our new controller, we should add a route to `config/routes.php`:"
```

```
            source_code :php, <<-PHP
    '/vote(/:action(/:id))' => [
        'controller' => '\\Suggestotron\\Controller\\Votes',
    ],
    PHP
```

message "This route should be placed **above** the generic `/:controller(/:action)` route which will otherwise catch the request."

end

step do

message "To allow our users to actually vote, we'll add a button to our topic list view, `index/list.phtml`, before our `Edit` and `Delete` buttons:"

```
            source_code :php, <<-PHP
            <a href="/vote/add/<?=$topic['id']; ?>" class="btn btn-success">
                  <span class="glyphicon glyphicon-thumbs-up">
                        <strong><?=(isset($topic['count'])) ? $topic['count'] : 0;?></strong>
                  </span>
            </a>
            PHP
```

end

step do

message "Finally, we need to update our `\\Suggestotron\\Model\\Topics` model, to both retrieve the votes for each topic, and sort by the number of votes:"

```
      source_code :php, <<-PHP
    public function getAllTopics()
    {
      $sql = "SELECT
              topics.*,
              votes.count
```

```
                FROM topics INNER JOIN votes ON (
                    votes.topic_id = topics.id
                )
                ORDER BY votes.count DESC, topics.title ASC";

            $query = \\Suggestotron\\Db::getInstance()->prepare($sql);
            $query->execute();
            return $query;
        }
        PHP
    end
end
```

```
explanation do
        message 'Guess what?  _**You\'re done!!!**_ Congratulations, you just "finished" your first web app!'
        message "(They're never _really_ ever finished... have fun tweaking it!)"

        message "Go take a look at your masterpiece: <http://localhost:8080>"

        message "It should look something like this:"

        img src: "img/finished_app.png"
end
```

next_step "credits_and_next_steps"

message <<-MARKDOWN

Now that you've finished the Suggestotron curriculum, what next?

# Extra Credit

If you got all the way through Suggestotron with some time to spare, here's some extra stuff you can try:

* Add [appropriate icons](http://getbootstrap.com/components/#glyphicons) to the Edit and Delete buttons

* Add a downvote button that does the opposite of what the upvote button does

* Add an 'about' page, linked from the bottom of the Suggestotron topics list. Link back to the Topics list from the About page so users don't get stranded.

* Add success messages when adding/editing topics

* Add error messages when adding/editing/deleting topics

* Make it so that all error messages are shown in the templates (e.g. going to the edit page without an ID)

# Authors

- [Davey Shafik](http://twitter.com/dshafik)

- [Michelle Sanver](http://twitter.com/michellesanver)

## Lovingly Based on RailsBridge

Thanks to the amazing work of the volunteers behind [RailsBridge](http://railsbridge.org) and their

decision to release all of their content under a Creative Common (CC-BY) license, we are able to create

PHPBridge by standing on their shoulders and building up.

# What next?

- It's probably time for the closing presentation.

- After that, start a project, tutorial, and come back again!

- All our favorite resources can be found on the PHPBridge site: <http://phpbridge.org/learn/resources>

MARKDOWN