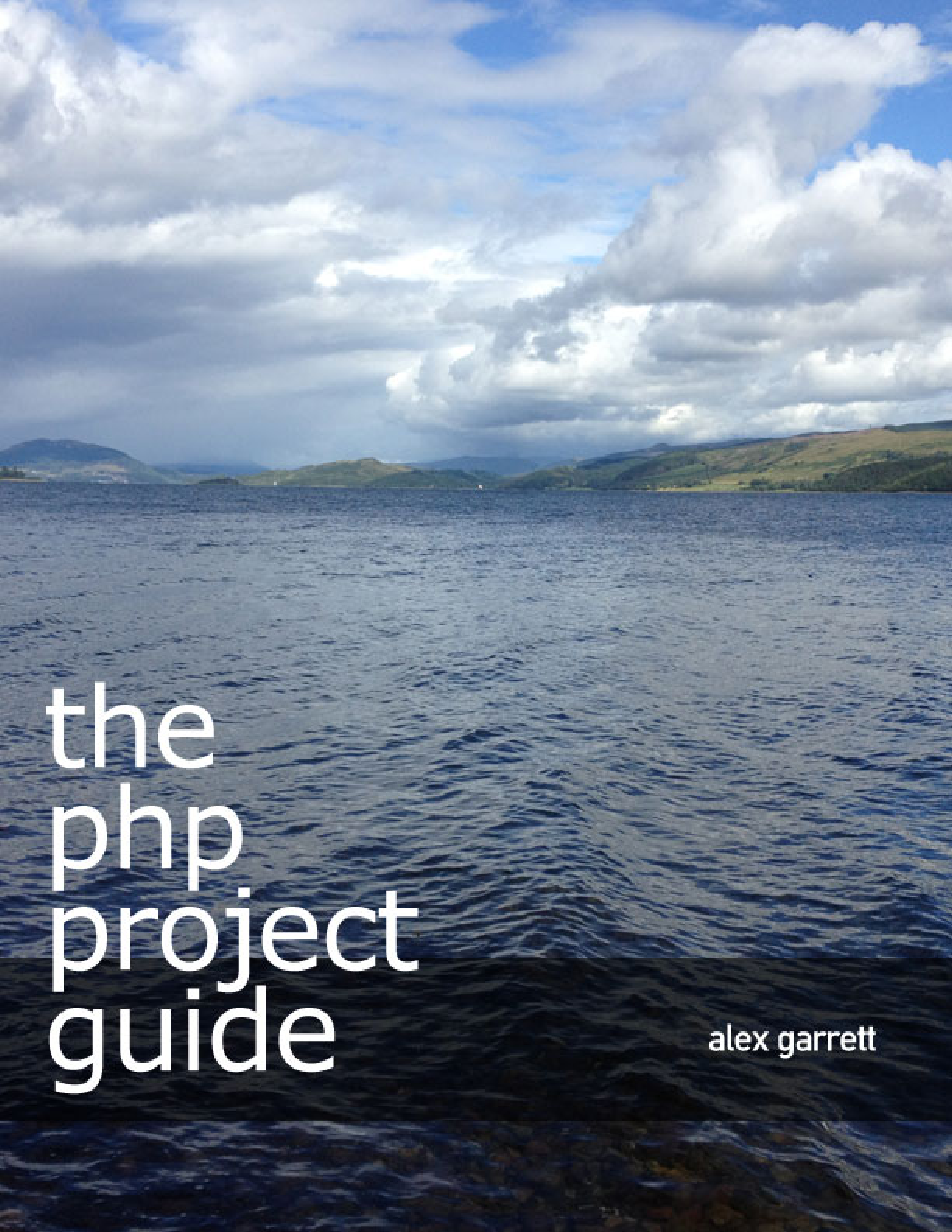

the php project guide

alex garrett

The PHP Project Guide

Alex Garrett

This book is for sale at http://leanpub.com/phpprojectguide

This version was published on 2014-06-13

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

©2013 - 2014 Alex Garrett

Tweet This Book!

Please help Alex Garrett by spreading the word about this book on Twitter!

The suggested hashtag for this book is #phpprojectguide.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search?q=#phpprojectguide

To Ma

Contents

1 Introduction

Most programming books will teach you the basics of the language, run through some example applications and perhaps give you some additional resources like a disc, or online resources that are barely updated if even touched by the author at all.

The problem with writing a book about a language is that things change, and even worse, all of the information within these books is freely available on the internet. Programming books can't offer information about future security, speed and other issues within a language, simply because they don't exist yet. You also finish a book with no idea of where to go next, or what to learn next. You may have gained knowledge about a language, its syntax, built in functions, etc., but what happens when you need to build your application? Do you really know how to piece together what you've learned?

This book has been written to guide you through PHP and give you the knowledge to move forward in building applications. It'll guide you through almost everything you need to consider when writing PHP and how you can become a programmer who can build anything, not just by knowing everything, but learning how to address things you're unsure of. This book aims to remove the stage that most people get to when learning a new language… the phase where you understand the language but aren't fully flexible in building whatever you want.

1.1 Typical programming books are hard to learn from

In this book, we'll take a problem or something we want to build and we'll spend the majority of time discussing it instead of listing reams of code. This is sensible, because you simply can't learn from looking at code. The more time that's spent discussing and planning a project means you can then use the tools you need to build what you know how to. If you don't understand how to perform a particular action within a programming language, that's fine! The difference is after thorough thought is that you understand how to address it and can learn how to do it. Once you've learned the hard way to do something, it's unlikely you'll forget it. We'll discuss this more in chapter How to learn.

2 Who this book is for

If you've learned PHP and you don't feel you're as flexible as you can be with taking building applications to the next stage, then this book is for you.

No PHP code knowledge is required for this book, so you can read this before you even begin learning. On the other hand, if you feel you're really good at the language, have learned everything you feel you can but can't quite put everything together, this book is for you.

We'll also discuss some general topics that apply to most web applications, like best practices, caching and security, so you may benefit from this book if you're already building applications but want to gain some additional knowledge that will help you on your way. Because of these general topics, you may find this book helpful if you're working with another web-based language.

3 What you should get from reading this book

This book teaches you how to learn better and gives a general insight into web development with a focus on PHP. By the time you've finished reading this book you'll have a better understanding of development process, resources and practices you need to be able to work confidently on a product. This book doesn't cover absolutely everything you need to know, nor will it teach you everything you need to know about PHP, but you'll be able to build on what's written on the following pages and use the help and advice to achieve your goals.

4 Errata

I’ve taken as much care as I can to ensure that everything in this book is correct. If you notice that any technical details, code, claims, information or spelling grammar is incorrect please get in touch and I’ll make any changes as required. I’d love to have produced something that contained absolutely no error but as with anything, human error does occur!

The beauty of publishing with Leanpub means I can make amends at any time and they’ll be immediately available to you, having already bought the book.

5 Useful notations

As this book is very conversational and contains more text than code, you won't find it filled with pages of code. Where any code is displayed, you'll see it written like this:

```php
<?php
foreach($x = 1; $x <= 10; $x++) {
        echo $x, '<br>';
}
```

Advanced stuff! If a line of code is particularly long, it'll be shown with a \ (backslash) to notify of a line break. Don't mistake these for actual backslashes within the code. Although of course, you may occasionally find backslashes in code. Just watch out!

```php
<?php
echo 'Did you know that the sentence "The quick brown fox jumps over the lazy dog\
" contains all letters of the English alphabet? Wow!';
```

Other notations exist such as warnings, tips, information and recommended exercises. These are obvious so I won't waste valuable space listing them out here.

6 Starting a project

When I talk about a project, I basically mean something you're aiming to build with a specific purpose that will be used by people to achieve something, or will be used to increase or create value for a specific purpose. A project doesn't have to be a commercial, it can simply be something you'd like to do to learn more about a language or could be something personal that will help you manage something better.

6.1 Project examples

Let's look at some reasons you may be starting a project and what you'll need to achieve the end result. This often helps visualise how you'd start your own project.

- *A system that logs and records items.* For example, you may play golf and feel like a system to record when you've played, your score and any notes. You'd need an interface, fields to enter specifics about your game and a database to hold this information. You'd then need to query the database to retrieve aggregates like the total rounds played in a chosen date range.
- *A chat system.* You need a system to chat in real-time within the browser to support workers and clients within a workplace. You could use one of the many already available online but want something simple. You'd need a basic interface, the ability for clients to enter some basic details about themselves to use the chat and then the ability for a client to post text and see the chat manuscript as it's happening. You'll also need pretty much the same functionality for whoever is talking to clients or workers.
- *A shop.* If you're selling items online, you might want to build a shop to allow adding to basket, registration and checking out. You'd need to build a secure authentication system to allow users to register and you then need to build in functionality to add items to a basket within that session. Checking out could then be handled by either PayPal cart checkout, PayPal Payments Advanced[1] for credit card acceptance or another credit card processing API. These are usually well documented and support PHP.

You'll notice a common theme amongst the examples above, whether or not they apply to you. The common theme is that they all involve database interaction and storing data. There are a few ways you can store data for later retrieval to display to a user, but there are thousands of different websites that can be built with a frontend, PHP and a database. For this reason, you'll find we cover quite a lot of database theory (not boring theory about the inner workings of databases, but how to think logically about what is required from your database). We'll also look at security as we're working

[1]https://www.paypal.com/webapps/mpp/paypal-payments-advanced

with a dynamic language and you'll often require user input of some kind. There are ways you can help yourself too, by using a framework, so we'll be looking into these and considerations that surround choosing a framework.

6.2 What you don't need

There are a variety of things you absolutely don't need when starting a project for the reason they cost you time and money and take the focus away from the important thing - the end product.

- Business cards
- Expensive software
- Expensive hardware
- An expensive, powerful webserver
- Advertising
- Staff

Of course, if you're starting a design company, you may need something like Photoshop. But, if you just need a logo for your website, there's no need to go out and buy Photoshop for this reason. Use something free like GIMP[2], instead.

Expensive hardware is also unnecessary as you can build a website on the most basic of hardware. If your computer isn't the greatest, there's no need to worry. Likewise, you won't need a powerful server to host your project on until the demand is significant enough to do so. During your development and launch phase, there is no need to cater for this demand unless you've spent thousands on advertising that'll invoke a large number of people visiting your website.

Business cards are useful sometimes, but it's unlikely you'll need them in the initial stages of your project. If you meet someone who you may potentially work with or would like to test out your website, you don't need to hand them a business card - just give them your contact details as you would anyone else.

In today's world, there isn't a lot of need to advertise unless you need to reach people on an epic scale or if you have a large budget and want to increase brand awareness. If you create a good project and provide excellent service, people will talk about your product and therefore eliminate the need for advertising. Carefully incorporating social networking aspects into your product can also lead to your website becoming popular through social media. Please be careful though, adding a 'like' button at the top of every page or article is less valuable than it may seem.

And lastly, employing staff is almost always unnecessary from the outset. If you're working with someone and splitting the company 50/50, this isn't staff. If you hire someone to answer customer queries by email when you don't need to, this is unnecessary. Try and deal with things like that first before wasting money on hiring someone for something that isn't a problem yet.

[2]http://www.gimp.org/

6.3 Taking the first steps

The first step to a successful product is to plan it out and get stuff done as quickly as possible. This means cutting out the time you take to carefully design wireframes and hire designers to put together a great design for you. So what if you end up with a working product that doesn't look quite right? You can perfect that after you know people want to use it.

It's a really good idea to start planning the functionality and start building as soon as you can, learning along the way. If your goal is to learn, these rules still apply. Building a really great looking template isn't going to help you learn PHP.

I often start building parts of a website in isolation. For example, if you need a currency converter on your website, create a blank file, research a currency conversion or finance API and start building something that works. It won't disappear, so will always be there for when you need to come back to it. It's simply another thing ticked off the list of what you need to do.

"Fail to plan, plan to fail" is a saying you may have heard. It's partly true, but can be taken in the wrong way. Planning isn't writing up long documents, holding loads of meetings or sitting by yourself writing reams of notes on what you need to do. Planning can be as simple as a non-prioritised to-do list. What bits do you need to build to make this website work? Short, simple tasks for the day will help you progress quickly without getting bogged down by massive amounts of information or instructions. If you're building the project initially for yourself or you're part of a very small team building an in-house product, minor details aren't likely to matter anyway, you can simply tweak them later after you'd received feedback.

But what if I just want to learn?

If you are still very new to PHP, your primary goal is probably to learn. Change that primary goal into building a product that works. The by-product of this will automatically be that you learn about PHP and the way it works since you're applying yourself to a real-world situation. Learning the theory of variables, loops and functions is all good, but how are you going to use them?

My first ever PHP project was building an application that counted daily intake of nutrition in food. At the time I actually believed it would be a useful product and built an interface where users could register, enter information about what they'd eaten, select items from a list that they'd previously entered, modify that information if required and see daily intake statistics and charts, as well as look back at previous days. This took me months and months to build, something that I could now realistically be built properly in a couple of weeks. Being able to develop something quickly comes with experience, not just knowledge.

A lot of people will get frustrated when they're building something and can't quite achieve the result they're after - but this is a *good* thing. It really helps when you struggle, because once you finally grasp something, it's unlikely you'll forget it.

6.4 How long should a project take?

As we saw in my example above, it took me months to build my first real project. I no longer have the source code but the knowledge I gained from it was valuable.

There is no specific time you can give to learning anything, it’s the time you put into it that counts. More importantly, it’s what you do with that time. I’d highly recommend anyone learning PHP, or wanting to advance what they know already, to build a project. The reason this book is entitled “The PHP Project Guide” is because it’ll guide you through what you need to know to start and work on a project, hence stimulating the learning process by building a useful product.

Ask any seasoned developer how long it took them to grasp PHP and they probably won’t know exactly, and probably because they learned while working on projects and were unaware of the time it took them to grasp the language.

Let’s jump in!

7 File structure

7.1 Why does file structure matter?

A well thought out, well executed file structure will not only make it easier for you to manage your project in the short run, but will also mean that as your website grows and requires more files and directories, you and your team will be able to quickly and easily locate files, whether or not they've been working with your website for a long time.

Let's say you're working with a framework. This will almost force you to keep a consistent file structure due to the nature of most frameworks implementing the MVC (Model View Controller) structure. However, working without a framework means you have to take extra care to ensure everything is where you need it. Lumping all of your project files into a root directory will not help. Carefully designing a file structure might also make it harder for attackers to locate files, particularly if you have error reporting off (which you should have turned off in a live environment).

7.2 Planning your file structure

This isn't easy, because just as every other planning phase, you need to look to the future. This means it's extremely frustrating to attempt to plan something to cater for future scale. The best thing to do is start small and only plan for separating main groups of files. This could be having a directory that contains several files all containing functions that relate to different areas of your application, or broken up pieces of template that you need to include as part of your overall design. If you're frustrated with planning out your directory structure, try and plan around what makes sense. For example:

```
require 'template/header/logo.php';
```

Looks and reads much better than:

```
require 'designs/logo.php';
```

7.3 Naming files

Files can be named anything as long as they can be read by your server. Usually, files are just given a short, lowercase name describing their contents. For example, it's safe to assume that init.php is an initialisation file, or logout.php terminates a user session.

An often preferred way of naming files is with an extra piece of readable information. Let's say you had three files:

```
require 'user.php';
require 'user.php';
require 'User.php';
```

These files could be located within different directories, and could also contain entirely different code. But, within the grand scheme of things, this can make it difficult to distinguish between them. We could have a user profile page, an include displaying a snippet of information about a user and a user class. It would make perfect sense to name all these files user.php providing they were within carefully named directories:

```
require 'user.php';
require 'assets/display/user.php';
require 'classes/User.php';
```

There! Now we're more aware of the difference between each file.

7.4 Watch your extension

As you may already know, PHP files can be named with any extension to identify them, you don't really need a .php extension. There are **extremely** important security considerations when renaming file extensions which we'll discuss in a moment. For now, let's look at an example:

```
require 'template/sidebar/latest.inc';
```

If we have PHP code within latest.inc, PHP will interpret it and everything should work as expected. However, you're now eliminating the webserver's ability to recognise this file as a PHP file, and therefore will be served as plain text. Even if you don't have sensitive information within this file, it can still give clues about the way your code works and is therefore a security risk. You can register any file extension to be interpreted as PHP file by adding the following to an .htaccess file:

```
AddType application/x-httpd-php .inc
```

You can specify multiple extensions like so:

```
AddType application/x-httpd-php .inc .class
```

However, it's always better to just stick with naming a file with a .php extension. This keeps consistency, eliminates confusion and means you're protected even if you switch your code to another server.

7.5 Organising classes

If you're working in an object oriented style, it's highly recommended when working with classes to place all class definition files within a separate directory. If you've not worked with object oriented PHP before, a class looks like the following and can contain properties (variables) and methods (functions):

```
class User {
        public $auth = false;

        public function login($username, $password, $remember) {
                // some delightful code here
        }
}
```

You may have a method to include a class as it's instantiated (when a usable object is created from a class). This means that by placing a file, for example, User.php into your classes directory, you know exactly what the class name is, and you can then instantiate it.

You may even have an autoloader, which would include any class found within a particular directory when you instantiate it. This means that to make use of a class, all you'd need to do is create a new file with a valid class defined and then place it within this directory. You're then free to instantiate it when required. Let's take a look at how we can do this in practice with some code:

```
spl_autoload_register(function($class) {
    require 'core/classes/' . $class . '.php';
});
```

How does this work? spl_autoload_register will automatically detect any classes that you instantiate. So when you do:

```
$user = new User();
```

This will find the file named User.php and include it ready for use. This is an extremely useful way of including classes, meaning you don't have to include a long list of class includes like so:

```
require 'core/classes/Database.php';
require 'core/classes/User.php';
require 'core/classes/Topic.php';
```

8 Development environment

Your development environment matters. Its where you'll spend all or most of your time writing code, writing tests and fixing bugs. Your development environment needs to suit you and the needs of your project.

You also may need to get used to working within the same environment in different physical locations, so an environment that works on a variety of operating systems is helpful. Of course, I'm simply talking about a text editor, whether this is on the device you're using or an online editor.

This chapter won't be too long, cover too much or tell you what you need to use. After all, it's personal.

8.1 Choosing a text editor

Texts editor come in many forms - from the lightweight to the bulky, meaning they're more like an IDE (Integrated Development Environment). For basic projects or when you're first starting out, it's probably a good idea to start with a basic text editor to allow you to focus on the code. If you're still learning PHP in depth and have a long way to go, it's a good idea to use a text editor that doesn't autocomplete your code for you. By this, I mean editors that finish a function name for you based on what you're typing, usually pulled from a library that comes bundled with the editor. This makes it really fast to develop because you don't need to go anywhere else to look these autocompletes up.

There are drawbacks to this though. * You might not absorb what you're writing if it's being autocompleted for you. * If something is deprecated you won't know. It's better to look it up in the manual.

Personally, I learn more when going and looking up information, as I often find myself lost in a tangent of information and jump from comment to comment or website to website finding better ways to do what I originally wanted to. This really helps when learning because you get different opinions and get a fuller picture.

Text editors can often be bulky and get in the way with what you're doing. Remember, all you're writing is a plain text file to be compiled by your server. Do you really need hundreds of fancy functions within your text editor to assist you along the way? What happens when you're put into an environment when you're writing code without this functionality and can't expand and collapse code blocks and request all sorts of useful things to happen with the click of button? However unlikely it may seem that you'll be pulled away from such an environment, it helps you become a better programmer when you don't rely on a vast amount of tools.

8.2 Development envionment settings

Problems can arise with members of a team using different settings, such as line indentation when tabbing. You may find that one person has two spaces assigned to a tab where some have four. This means that when code is working on by more than one person and settings vary, you can end up with a mess, or, it'll be really hard to modify files as you'll have to keep manually changing tab spacing or character encoding.

The solution is to use something like EditorConfig[1]. This nifty tool allows you to define specific standards like indentation type and size, as well as if line endings should be removed and the character encoding of the project files.

A typical EditorConfig file looks like this.

```
root = true

[*]
charset = utf-8
indent_style = space
indent_size = 4
trim_trailing_whitespace = true
```

This sets the character encoding to utf-8, sets the indent to 4 spaces and removes trailing whitespace on all lines. This file (named .editorconfig) would be transferred around with the project in the root directory and will apply these rules to all developers who have the EditorConfig plugin installed.

Horray for consistency!

8.3 Version control software

Version control software allows you to carefully manage changes made to your application, typically through 'commits' and then syncing with the main project. You can usually create 'branches' which give you a fresh copy of the codebase and let you do what you want with it, including the freedom to mess everything up if you desire. You can then merge the changes (if all is well) or simply discard the branch.

This has a massive advantage over any other method of development as it allows any changes to the main application to be made in isolation from the live code. The benefit of working like this is the obvious, changes don't interfere with live functionality and often branches can be rolled back once pushed live, so if something goes wrong everything can simply be undone and the branch development can continue, either to fix the problem or remove it altogether.

[1]http://editorconfig.org/

The only fiddly part to developing this way is conflicting file changes. Conflicts occur when two or more developers have worked on the same file. Because changes per developer don't immediately show for all developers, files need to be merged so the changes from each file are incorporated into one before they go live. This isn't too much trouble if there are only a couple of changes, but sometimes a vast amount of changes can leave both developers confused about where code has been added, removed or modified. This can prove particularly complicated when both developers may forget which small changes they've made and are unsure which is the correct change. For example, changing a CSS width value from 10px to 11px. Luckily, there are tools that allow merges to be made and assist in the process by highlighting changed lines so the files can be easily merged. Typically, the two files sit side by side in a text editor environment and can be compared side by side.

Git

Git is version control software. It's extremely fast, flexible and easy to use. It handles versioning extremely quickly. Git can be extremely easy to use, but offers powerful functionality that give you complete control and flexibility when working either by yourself or as part of a larger team.

Explaining the ins and outs of Git is way beyond the scope of this book, so go ahead and read up on it.

I'd highly recommend Pro Git[2] by Scott Chacon. Don't let the name put you off, it'll slowly introduce you to Git as well as cover the more advanced use of it.

GitHub

GitHub is a popular resource for hosting code and implementing Git version control. With GitHub, you can create public (open source) or private repositories that allow to you host your application's code and allow a larger team to contribute to.

The beauty of working with something like GitHub is social coding. If you have an idea for a project and want to host it for others to use and contribute to, chances are you're going to end up with a much better codebase than you had before. By opening up your idea with the world, you also get more exposure and open source software is a generally a great way to go. Even if you're making a profit from your idea, opening up the code is a very admirable thing to do and doesn't nessasarily mean others will steal your idea.

As mentioned, if you want a private repository, that's fine! It works in the same way, it's just closed off from the outside world and only those invited as contributers to the project can work on it and see the code.

[2]http://git-scm.com/book

9 Frameworks

Although we won't be discussing anything in relation to frameworks throughout this book, it's important to touch on what frameworks are, and the obvious benefits and drawbacks to consider. A ready-built framework isn't something that every project should necessarily use, but if you understand the benefits and drawbacks of using a framework you can apply them successfully to a project, which in turn could save you time in the future.

Technically, however, when you build a website, the code you use is your website's framework. When we refer to a framework we refer to a ready-built framework that's used by more than one person to create a standard. Every project needs consistency and the majority of ready-built frameworks are consistently built throughout. On the other hand, if your website code uses consistent code and you're working within your own framework, this is absolutely fine. You'll often find a lot of frameworks are built and released coming initially from projects. Developers may re-use their framework for each project they start on and therefore can potentially be used by others.

9.1 What's a framework?

A framework is a ready built structure for your application. These are mainly based on the MVC pattern, which means that Models, Views and Controllers are used, helping to separate the logic of your application. Otherwise, a framework will simply provide a basis for your application to be built on top of, meaning you'll have a ready built file structure, style of writing your code and access to helpers that save time when writing code.

Frameworks include helpers that allow you to do things quickly and easily. We're not talking about the basics like str_to_upper functionality, we're talking about things that would usually take a significant of time to write yourself. For example, user authentication is used on many websites and therefore is built into most frameworks. This means that instead of creating the entire functionality to sign users in, allow them to create accounts, recover their details, etc., it's already there for you.

A framework will typically be downloaded and extracted to a working directory. There may be some configuration to get it up and running, but this will more than likely just be database configuration, depending on which framework is used. There may also be some important security setup, like generating and specifying values for security operations like salting passwords. We'll learn more about this later on in the book.

9.2 More about MVC

MVC (Model View Controller) is a pattern that allows business logic to be separated from views, which is essentially what you output to a user. Models tend to fetch data from a database or resource and return it, where the controller will take this data, process it and decide what to do with it to make it useful to the view. Logic will go within a controller to process anything from listing topics in a forum to processing a user login request. The view will then make use of the Controller's process and output what it needs to the user.

So, how does this benefit a programmer? For a single programmer, you have a lot more control over the structure of your code. This means you can concentrate on outputting what you need and don't always have to worry about what goes where. You have a defined way of processing and outputting and therefore you won't be repeating yourself.

9.3 Don't repeat yourself!

This is abbreviated to DRY and is an important concept in any development, and is particularly suited to working with an MVC pattern. As we've already discussed, models allow data to be loaded and used by controllers. This makes it extremely easy to create generic functionality to retrieve data. For example, if all your tables had a primary key 'id', you could create a generic handler for retrieving data by just passing the table name and which information you want to retrieve. This would cut down on the amount of code you'd need to repeat to select data from different tables.

Implementing DRY design keeps your application development time to a minimum and also means maintenance is easier due to less spaghetti code.

9.4 What a framework won't do

Let's get this straight! A framework will not allow you to learn PHP quicker, as helpers often exist for functions you'd usually find in PHP. For example, look at the different between the two lines of code below. They both do the same thing, but one is native to PHP and one to a framework we might be working with:

```
echo str_to_upper('Don\'t shout'); // native to PHP
echo Str::upper('Don\'t shout'); // used by a framework
```

Obviously this isn't complicated functionality, but it could be. You're taken away from native PHP and are plunged into a world of using functionality defined by the framework you're working with. This isn't all bad though. If you know how to do something anyway then why not make life easier? Good frameworks are built and designed by developers who have an extremely good understanding of the language they're writing them for, and are those who have identified things that could have been done better and faster.

Of course, you could begin to use frameworks and never stop throughout your life, but is this really feasible? You owe it to PHP to learn its native functionality!

9.5 The benefits of using a framework

As we’ve already seen, a well built framework will provide a solid base for you to build your application on. This can help give a good, clear structure, particularly with a popular framework that is well recognised. Let’s say you choose a popular framework with a wide user base. This will make it easier for someone else to jump in and start working on your application if required, which will mean that less training is required. If you find a popular framework that suits your needs, you’ll also have the ability to join a community of people all working within the same framework, which means you can potentially get help faster, and gain useful information and tips about the framework. You may also have the ability to contribute to the framework if you discover a bug, think something can be done better or you’d like to build a specific feature. Many frameworks also provide a library of add-ons which you can freely contribute to increase the framework’s functionality.

As mentioned, the majority of frameworks take advantage of MVC. MVC allows the logic of an application to be separated and is commonly put into practice for applications with a lot of developers. Let’s say you have developers that specifically work front end, writing HTML, CSS and creating some functionality that nicely outputs database stored data. Whether or not these front end developers are qualified or interested in the backend part of the application, their job is to build on the front, so with MVC, they’d have the ability to work on the views, pulling what they need from controllers that utilise models. Likewise, the backend work would be done on the models and controllers for the front end developers to make use of. Either way, if you were to have developers working on both, it still makes logical sense to keep everything separate, making it a lot easier to manage code.

9.6 The drawbacks of using a framework

Up until now, it seems that frameworks seem like a good idea to base an application on. However, there are a few problems to address if not now, into the future of your application. There are also some considerations to make when choosing the framework for your application, which can have negative effects if you don’t make quite the right decision.

Frameworks are not all the same. You’ll find that one framework will be suited better for a type of application than another. Likewise, you may find frameworks bloated with features that try and suit to everybody. Whatever your requirement, it’s extremely important to research any frameworks you may have come across and judge whether it’ll work well now and into the future. For example, your application may rely on fast execution time, so a feature packed framework may not be suitable for you. And, of course, in this situation you’ll need to gauge execution time of any helpers that the framework may use.

A common problem with the perception of frameworks is that they’re built more securely. The reality is that even the most popular framework will have security flaws. This isn’t the problem, because security flaws will always pop up. The problem is that the assumption that frameworks are always secure means that sometimes these problems get ignored. If you choose a framework with a large community, it's more likely that any problems with the framework will be reported to the community and be fixed and released to download. It's very important to keep the framework you're using up to date by using newer well tested versions and patching any security flaws where patches are available.

The last consideration depends on how much work you do outside your framework. By this I mean that if you're constantly working within your framework, you may be stopping yourself from exploring what raw PHP offers. This obviously depends on how much you've learned prior to using a particular framework. If you decide to use a framework, don't let it become so normal that it overshadows PHP on its own.

It's also worth mentioning that people often use frameworks incorrectly. That is, they write code within the framework not in the way it was originally intended to be written. I could simply download a framework, start using it but go wild, creating files in the wrong place, modifying code that isn't meant to be modified, and using helpers incorrectly. A good example of this would be to literally set a session on the spot (perhaps for a logged in user) rather than using the helper built into the framework to handle this. There are no constraints within pure code, so it's always a good idea to get to know the framework you're using either through the documentation or someone who can teach you to use it properly.

Tip

All frameworks will boast their functionality. Remember that this may not always be the case. If in doubt, gain advice from people who have made use of the framework you're considering using.

9.7 So what next?

If you're deciding which framework to use, base it around several factors. These are:

1. What your application requires and may require in the future. The more helpers that are available that will of use to you, the better. On the other hand, don't just pick something with the most features in a panic, or you may be sacrificing performance for features you don't need.
2. What looks easiest to learn or the most widely used. Really, this is for if you plan to employ people to work on a team. The last thing you want to do is pay for any training or waste time waiting for someone to learn a framework. Ideally, you need to be in the position to employ people that know the framework well.
3. Community support if you're learning the framework yourself. Support from people who know what they're talking about will help greatly in learning a framework simply because it's so easy to divert away from the way the framework was intended for. Most frameworks have standards that should be adhered to.

If you choose a framework, congratulations! You're more than likely to have an easier, faster time developing now and into the future. If you're still yet to choose, think carefully and choose something you think will best suit your project now and into the future as it's extremely difficult to move from one framework to the other.

If you're learning, perhaps give working with a framework a miss until you've grasped PHP on its own.

10 Following coding standards

Coding standards are important when working on an application to ensure your code is consistent in design and syntax. For example, using naming conventions for variables like camel caps (textThatLooksLikeThis) or using underscores between words. There are also a host of other ways you can keep consistency in your application. Following a chosen coding standard will ensure that you and anyone who works in your application will understand its design and the code will be easier to work with throughout the development process.

10.1 Zend Coding Standard

The Zend coding standard is taken from the Zend framework and is probably one of the most commonly followed, even if you're unaware you're following it. Some of these rules include using single quotes when defining string literals. For example:

```
$name = 'Billy';
```

Coding standard like Zend often get down to the nitty gritty of even defining spaces correctly, such as when concatenating (joining) two strings with a full stop. These types of rules also include what should be brought to a new line and what shouldn't. A very commonly discussed standard for readability of code includes whether or not curly braces should be brought to a new line. Zend says these should be, for example:

```
public function userDetails()
{
        // class code here
}
```

Whereas otherwise this may be:

```
public function userDetails() {
        // class code here
}
```

There are a lot of people that prefer including the first bracket on the first line, however this is personal preference. The Zend style is in place because it makes the code contained within the brackets to be more easily readable.

10.2 Your own coding style

It’s acceptable to define your own coding style. If you’re working on a project that other people may work on in the future, it’s a good idea to define your style within documentation so this can be picked up and followed by anyone else working on your code. Remember that people sometimes prefer to follow a particular standard, whilst others are absolutely fine with picking up a coding standard and sticking to it in the long run, so bear this is mind when deciding - particularly if you’ll have a lot of people working on your website.

There are a vast amount of programmers who write barely legible code. This is easy enough to get into the habit of doing, perhaps the result of pasting snippets in and writing code so franticly in order to learn or debug errors. If you’re working like this, stop right now! The cleaner and more readable your code is:

1. The more likely you are to understand it when you approach it at a later date.
2. The more likely that other people can read and understand your code quicker, meaning they’ll be able to provide you with better help and support.
3. You’ll learn best practices which are valuable when working in a production environment.

Sort your existing code with a code beautifier

A code beautifier is an application written to change the style of your code immediately by pasting it in and having the application rearrange it for you. These are often not complete, but will give you a better page of code if your code is a mess. Bear in mind that these applications will more than likely not change or suggest different variable names to give better meaning, and therefore you shouldn’t rely on them to fix your coding style. Code beautifiers are often configurable so you can change their output to your taste.

It’s probably recommended to go through your code manually and fix it up, as this gives you practice on the coding style you’re following, whether it be official or your own.

10.3 The heredoc syntax

Not necessarily anything directly related to coding standards, but the heredoc syntax will help you clean up any large multiline output by providing a clean and easy way to write these chunks.

Consider a large SQL query, perhaps 40 lines long. Writing this with the standard notation and escaping characters within these large blocks can become confusing and don’t maintain readability very well. The heredoc syntax avoids this, meaning you don’t need to escape with quotes within these blocks. An example of outputting a large query could be:

```
$name = 'smith';
$sql = <<<EOD
        SELECT `name`
        FROM `customers`
        WHERE `name` LIKE "%{$name}%"
        EOD;
```

What you write in place of ‘EOD’ doesn’t matter. This is simply the identifier. In this case, it may be better to describe what you’re encasing, e.g. use ‘SQL’ or ‘HTML’.

So, when do you use this? Well, for large HTML output and large queries, it seems like a good idea due to the amount of escaping you may need to do, if not now into the future. For a small query that will require absolutely no variables injected into this, it would be unnecessary to do so. You can find more about this syntax in the PHP manual, including examples.

10.4 Confused about what to do?

Really, as long as you’re being consistent in your code design and it’s readable, it doesn’t matter. Readability is important for other to understand your code, and for you to understand your code when you return to it. Having code that flows well and can be read with ease generally creates a more harmonious experience, although in the short term it can be confusing to decide what to do and maintain your style.

Your code is your own, and it deserves to be written with care and caution so everything looks better and displays an expressive nature. This is one of the main reasons that OOP is favoured when designing your application, as it gives more meaning and structure in an expressive form. We’ll learn more about this later on.

11 Frontend design

You'll more than likely want to include a design for your website, after all, it's a website! This is probably something you've already thought about, perhaps even designed or had designed for you and you may even have the design sat in front of you waiting to embrace functionality.

The fact that you already have a design is good. It's a great part of the process that will allow you to see your website come to life. It's not advisable to spend a lot of time or money on an initial design as along the way you may need to change it anyway.

What you now need to do is build your application into your design, and start to see some functionality. But, how do you break up your design so you can easily integrate it? You need a template system. The reason you need this is so you can avoid duplication of frontend code. Part of good development means you can quickly alter and update your website and for this reason you need only one instance of your design that can be updated and reflected through your whole application. Thinking about this carefully means you can avoid the frustration in future of having to go through every page of your website and update something simple.

Your website should be designed in such a way anyway that you avoid inline styles and use valid HTML and CSS, with the stylesheet included in the head of your document so changes can be made to this external file, as well as it being downloaded and cached by the browser. We won't go too much into frontend considerations, but it's important that we touch on it as a general web development concept.

11.1 What about while learning?

Don't let a design overshadow your learning process. If you're learning, a completely blank canvas is fine. If you're focused on the code and output, you'll probably be better off. Worrying about implementing what you're learning on the backend into the frontend could cause hours of pain and I don't recommend it. It's extremely easy to incorporate code into a design once you've finished, anyway.

11.2 Learning for the frontend

Frontend design is a whole different thing to working in the backend on PHP, and many people try to do both. This, of course, is absolutely fine! You'll find that some companies either employ frontend or backend developers, which does make sense, but that doesn't mean you can't learn both. Working in frontend design is sometimes difficult and can be very challenging to keep up with. CSS is evolving rapidly all the time and something that was acceptable only months ago may become warned against today.

I've known many people that find CSS properties easy to learn and understand that something like the following will change the body background colour to black and the body text colour to white.

```
body {
        background:#000;
        color:#fff;
}
```

Some people know how to style elements to look just the way they want and work with pseudo elements (e.g. hover) and more advanced CSS, but just can't put together a whole template and make it look right. This isn't the learners fault, as CSS has a very poor layout system. In fact, CSS until recently didn't have a specifically designed layout system, and relied on the developer to float elements left and right against each other and using the clear property. We're now starting to see more specific tailored structure for laying out websites in CSS, and if you're interested in working on the frontend as well as the backend, check these out. This obviously isn't a CSS book, but these are worth mentioning as it ties in closely with web development.

11.3 Cutting up your template

We're now going to look at a basic way of cutting up a template ready for you to write code in it. This isn't always recommended but can benefit when you're diving into your first project and need to quickly cut up a design and output some dynamic content.

The best way to start cutting up your template ready for use is to identify the areas that will change and the areas that will remain static in terms of site content. You may have several assets along the sidebar that can be included as part of the main design. You'll more than likely have a main content area where the majority of your website information is displayed. To start cutting up your design, it's best to start with smaller assets. Here are some steps that should point you in the right direction. We'll also talk about directory structure here, as we'll be including all design files within a main template directory.

1. Start by identifying the main assets on your website. These could be login areas, lists of data, common buttons grouped together or small feeds. Don't include larger items like the header and footer of your website.
2. Create your directory structure so you can start chopping out the design and including these assets in different files. Perhaps create a directory called template and within this another called assets. You'll need to create a file for every asset.
3. Paste the code from each asset into these files.
4. Where you've removed the code from your HTML, use a PHP include to place this back in. For example:

```
<?php include 'assets/login.php'; ?>
```

Note that the above code assumes you've already broken up the header, footer and sidebar - which we'll look at in the next step.

After you've broken up assets, you're ready to start panning out and breaking up the template into the header and footer so you can quickly and easily create pages to house content in between. You'll have an overall header, and overall footer.

1. Identify where your content will be placed. This will most likely be somewhere like the following:

```
<body>
        <section id="content">
                Main site content here
        </section>
        <aside id="main">
                Sidebar stuff here
        </aside>
</body>
```

1. From the top of the document to where the section element starts is your header. Remove this code and place within a file called header.php in your template directory.
2. Place code from the end div downwards in a file called footer.php.
3. There's a problem. Your sidebar is now within the footer file. You can remove this and place it within a file of its own so it makes sense, and then include it.

All of the advice and steps here may not apply to your website at all. Designs, markup and needs vary dramatically and therefore you need to look at your code and make sense of how it should be broken up.

When you've broken up your code and have previewed your design to ensure everything is working, you can now easily create a new file in your root directory and include both your header and footer files. The point is that now, you have everything broken up and can quickly create files and make amends to your design as you go. The only problem you may face is that you need varied page layouts. The best way to address this is to broaden your main content area and simply include different layouts within this, such as different grid structures depending on how you layout is designed.

You can now start to focus on your content and include your global files within each template. You could of course include all of your classes, functions, initialisation, etc. within your header, but some may not be needed. You may be looking to include specific functionality on a page by page basis as and where it's required. This can be done at a later stage to address speed problems, but it's always a good idea to start to think in advanced.

11.4 Embrace MVC

MVC is a pattern that separates business logic from the user interface and keeps code clean and easy to manage. These are often found within frameworks, however you can implement your own structure if you know how. Frameworks will almost always make use of some kind of template engine to help with the MVC pattern, meaning that you'll spend less time fiddling around with views (strictly for output) and more time actually outputting stuff you've written in your controller.

11.5 Template engines

A far easier way to include dynamic output within a template is to use a template engine. These will assist in keeping your output as clean as possible while providing a simple, fast way to output data. Originally, we may have something that looks like the following:

```
<div class="asset online">
        <ul>
                <?php
                foreach($users->online() as $user) {
                        echo '<a href="/user/', $user['username'], '">', $user['username'], '</a>
                }
                ?>
        </ul>
</div>
```

Not entirely messy, but a large page filled without output will soon become unmanageable. With a template engine, the above snippet of code may start to look a bit smarter:

```
<div class="asset online">
        <ul>
                {foreach($users->online() as $user}
                        <a href="/user/{$user.username}">{$user.username}</a>
                {/endforeach}
        </ul>
</div>
```

Much better! We now have something a little smaller and a lot more readable. The above code is actually possible thanks to Smarty[1], a popular PHP template engine.

The PHP framework Laravel uses its own template engine called Blade and is extremely easy to output data within the framework. Passing variables from a controller mean you only have to do something like:

[1]http://www.smarty.net

```
Hello, {{ $user }}!
```

Or more appropriately:

```
@if (Auth::check())
        Hello, {{ $user }}!
@endif
```

And of course, another benefit of using the MVC pattern is that your views benefit from using data passed to them, which mean you can keep logic strictly away from your views.

12 Procedural or Object Oriented?

A lot of people start learning PHP using procedural code. This is perfectly acceptable because it introduces the language in a way that feels more natural. On the other hand, it's important to grasp Object Oriented Programming (OOP) with PHP because it makes for more maintainable, reusable code. OOP within PHP can be confusing and complicated at first, but building this way can ease development into the future.

12.1 What is Object Oriented Programming?

OOP was introduced with PHP 4 and is often confused as extended features. I've often heard it referred to as an add-on, plugin or extension. Let's take a quick overview of the fundamental parts of OOP.

OOP is a style of programming that takes away from the procedural and functional nature of programming. It makes programming modular and assists in separating business logic and views by allowing for better patterns to exist. In short, OOP isn't extended functionality in terms of what a language can do, it just allows for code to be written in a particular way. An inexperienced programmer could mix the two styles of writing which would be incorrect. Writing object oriented code can be as simple as separating functionality of your website into classes and just having properties and methods, or could be as advanced as it allows, creating very powerful easy to manage code.

Classes are object definitions, that become objects when instantiated (used). These contain properties (variables) and methods (functions). If you had a class called User, you would instantiate this, and would perhaps make use of methods like login, or createUser. Or perhaps you had a Database class, could have several properties - host, username, password and database. On instantiation of this class, you could pass arguments to then set properties within the object. Another method, connect, may then make use of these properties to connect to a MySQL database. This class would provide functionality to query your database tables and retrieve results. Basic OOP really is as simple this, with a host of additional functionality that allows you to proceed into advanced OOP programming.

At least, it's good to start with basic classes, methods and properties. The reason for this is that in the future you can quickly expand on what you've built initially without having to rebuild your entire website.

12.2 Tips for learning and retaining

If you've already tried to learn OOP, you'll probably have already seen examples that relate to everyday objects that can have properties and methods. These are often presented as real world things like cars or animals. There is a massive problem with the bridge between learning OOP and examples like this, because as with any technical subject, it doesn't give you the practical benefits as you're learning. Sure, you can learn that an animal can be an object, it can have different properties, inherit from a superclass species and can contain functions like meow when you create a Cat class, for example. However, when it comes to setting up a Database class, how is this helpful? You'd now need to apply this knowledge to your actual application but have absolutely no idea what you're supposed to do with a Database class. The answer is simple, but harder in practice - and that is to practice! As with anything, practice can make perfect, and working with OOP in a real application will help you understand how everything should tie together. Of course, you can still grasp the many techniques from these resources that teach OOP in relation to real world objects, but can then logically apply the thinking towards your application.

For example, let's say you need to create a registration and login system. If you start to think of each user as an object, you can start to apply this. You'd go ahead and create a class User, which could then be instantiated and you'd then make use of the many methods you may create within it, for example, login. But, what happens when you then need to create a slightly different variation of this class, for example for administrators? You may like to keep a separate Administrator class so you can instantiate this for when administrative functions are required. So, you would then be able to apply what you've learned in a practical way and create an Administrator class, which would inherit all properties and methods from your User class. You could then create additional methods within this, serving additional functionality to the administrator that wasn't previously available within the main User class.

If you're unsure about inheriting in OOP, it's a fairly simple concept. Creating a class without declaring it as final means you can create additional classes and inherit properties and methods from that class. The class from which your inheriting is then a superclass and the recently created class is a subclass. This sounds complicated, but simply put, you're creating a class that's taking properties and methods from another class as long as the properties and methods are not declared as private (which means only the class they're defined in can have access to them). Similarly, learning about encapsulation can be made difficult simply due to its name.

Encapsulation is basically the practice of setting visibility modifiers (again, a slightly more complicated sounding term) to your properties and methods. These can either be public, protected or private. The difference is simple - public properties can be accessed from anywhere, by referencing the class on the page where it's instantiated or within a subclass. Protected properties can be access within any subclass, but not from outside of the class and private properties can only be accessed from within the class they were defined in.

These rules, and the ability to define classes, properties and methods in these ways can sound rather odd. The key is to understand that this is rarely to alter functionality. You're not writing code that will directly affect how your website will function, but instead you're writing this code by design and conforming to a standard. This makes it a lot easier to reuse code and for different developers and yourself to work more efficiently on your website by giving clearly defined code, separated by logic.

12.3 But what about functions?

Functions can and should be used for their original purpose - providing functionality that can be used again and again. Although methods within classes are technically functions, you can define a file containing functions that you may commonly use that aren't related to any particular class. For example, you may have a function that calculates the VAT for a particular item's price based on a global configuration value, or something more general like calculating the area of a circle. Of course, these can be embedded within your defined classes, however bear in mind that it wouldn't make sense to say $user->calculateCircleArea($radius);

PHP has been criticised as not being a truly object oriented language because of its use of functions. Of course, PHP started purely as a procedural language and of course cannot remove functions that already exist for backwards compatibility.

12.4 Is it ok not to use OOP?

Procedural code is fine. If you still prefer to use procedural code then this won't affect the way your application works as long as you've carefully designed it and haven't made a mess of the source code so it's either unreadable or hard to work with. It's argued that because PHP isn't a purely object oriented language that it's not always necessary.

Let's take an example. You have a website that searches your database for articles. The user is presented with an interface and types some keywords, and is then forwarded to a page where they see a list of results. The files involved here would most likely be the landing page (index.php) and the results page (results.php). From this, we can pass keywords from an HTML form to our results page and pick up these keywords with the $_POST array. All that's required here is a simply call to the database (after some sanitisation and splitting of keywords, of course) and then the results are returned as, perhaps, an array. You then have a predefined template to display results and the associated information and you simply loop through this list. You may decide to add pagination or a dropdown list so the user can specify how many results they see and this would be passed from a form and picked up and applied to either the query or the code you're using to loop and display results. If you're working with simply functionality like this you may not benefit from using OOP.

Of course it's ok to not use an Object Oriented approach when programming, however moving forward you'll see a lot more code written in this style. You also may find it harder to move to another language that depends on this style if you don't at least learn the basics. Remember that this style of programming is and should feel natural.

12.5 Examples in practice

Let's look at a couple of examples where you may want to apply this. The first will be a registration and login application and the second will be a generic database handler to return data quickly and easily. These are both useful and would more than likely apply to any website.

Database handler

What do we need to do here? The first is to create a skeleton outline of our class. Doing this initially makes it easier to grasp what you need to start doing within each of the properties and methods you need to create.

```
class Database {
	private $_link, $_result;
	public $count = 0;

	public function __contruct($server, $username, $password, $db) { }
	public function query($sql) {}
	public function result() {}
	public function count() {}

}
```

We could add more functionality, but let’s keep this short. When we instantiate this Database class we want to pass in the server, username, password and database. This could come from a configuration file and would then establish a connection to our server. We also have three properties, link (the link to our database connection), result (the result of the last query performed so we can access this from other methods) and count, which is simply the number of rows returned by the last query.

```
class Database {
	private $_link, $_result;
	public $count = 0; // can be accessed by subclasses and outside of the class if \
required

	public function __contruct($server, $username, $password, $db) {
		$this->_link = mysqli_connect($server, $username, $password);
		mysqli_select_db($db);
	}

	public function query($sql) {
		$this->_result = mysqli_query($sql, $this->_link);
		$this->count = mysqli_num_rows($this->_result);
	}

	public function result() {
		if ($this->count >= 1) {

			$rows = array();

			for($x = 1; $x <= $this->count; $x++) {
				$rows[] = mysqli_fetch_assoc($this->_result);
			}

			return $rows;

		} else {
			return false;
		}
	}

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

}
```

This is generic functionality to connect, query, count results and return rows. Obviously it's an example, but looking carefully it should make perfect sense. Anything larger than this may be difficult to absorb, but don't worry, it's unlikely to relate to your website. Remember that starting small and building up as you go is fine. If it works, you're on the right track and can improve and tweak as you go.

Warning!

Remember working with OOP doesn't excuse you from security. You need to implement security as you would at any other time.

12.6 User class

You may need something to handle users and then expand this to add different functionality for special users, like admins. This is extremely easy and makes a lot of sense using an object oriented approach. Let's look at an example class:

```
class User {
        protected $_id;
        public $data;

        public function set($id) {
                // identify user so we can use $id in other methods
                $this->_id = $id;
        }

        public function get_data() {
                // get user data from database and store in $data property
        }

}
```

A short snippet that doesn't do much, but that's ok, it's only an example. If we wanted to create an admin we may want to add special functionality to an additional class to allow other methods, but keeping the original methods. We still need to access admin data! The reason we used a protected property is so we can access this $id within any sub classes.

So, we could do:

```
class Admin implements User {
        public function get_users_banned() {
                // return the amount of users this admin has banned
        }
}
```

We can also access get_data, etc. by doing something like:

```
$admin_data = $admin->get_data();
echo $admin_data['username'];
```

This is due to the fact that all public and protected properites and methods are available to us as we've inherited the User class.

12.7 Grasping OOP

It can be difficult to grasp the concepts of object oriented programming in PHP due to the time most PHP programmers have spent writing procedural code. If you're experienced in something like Java, writing object oriented PHP code shouldn't be too much of a struggle.

The best way to grasp this style of writing code is simply to do it. Think to yourself that the next application you build will be written like this and you can find help and support along the way to improve and extend the functionality. A great way to practice is to look at any classes you're currently implementing within your project if any. Let's say you have a class that allows you to read Excel files. Open it up, have a scan and see how it's written, but of course don't rely on looking at other's code.

It's also a good idea to look at some popular, well-built and expressively written frameworks and look at how they work.

12.8 Recommended reading

Although this book extends into very advanced topics, I highly recommend PHP Object-Oriented Solutions by David Powers. This gives a good basic overview of the fundamental parts of OOP in PHP and extends to more advanced topics.

The PHP manual is obviously somewhere that can be useful when learning OOP with examples and information about the structure of the language.

13 Databases

Your database will almost always be the fundamental part of your application. Careful planning of the structure, data types, relation between tables and table engine used is extremely important. It's often a great way to also start thinking about the overall problem of planning your application. Visualise your database tables and data, and you should be able to gauge how your code should start to work, and how you may interact with your data.

13.1 phpMyAdmin

This is an extremely popular tool for easily creating and managing databases, tables and data. phpMyAdmin allows you to interact with your database with a GUI (Graphical User Interface) and do almost everything you'd be able to do within a console. If you haven't already guessed from the name, it's built on PHP, so go and have a look at the source code. The great thing about phpMyAdmin is it's free, open source and very powerful. phpMyAdmin is also easy to use in terms of setting it up on your server. As long as you know your MySQL server's default username and password you can download, extract and point your browser to its location. From here you can log in and start managing your database, permissions and take a look at other information regarding your MySQL server.

Please be aware that by doing this on a publically available environment you're giving another point of entry to your database. This means you should be extremely careful about where you place it. It might make sense to only use it during development and remove it temporarily when your website is live, and only use it when you need it. Of course, this isn't always possible so you may need to keep it permanently. That's fine, but:

1. Choose an extremely strong password. The longer, more complex and varied it is, the better.
2. Don't keep it in an obvious location like /phpmyadmin.
3. Keep phpMyAdmin up to date whenever a new release is available. Unless you've changed your configuration file, you can just copy over the new downloaded version and get going again. This will help patch up any security vulnerabilities.

phpMyAdmin will be available if you're using a server with cPanel or Parallels installed, which will be protected behind this environment, so may offer a better level of security and ease of access.

13.2 Planning your database

Regardless of whether you prefer to start working on paper or start creating your database tables directly, you should start planning out the tables and fields you need straight away. This will start to get you thinking about what you can eliminate and what you may need to add along the way. For example, do you need a field for a user's first name and last name, or could you have one field for their full name and use PHP to split the name on the server side? Of course, you could also split this within your query, but in terms of performance this wouldn't be ideal. Some decisions can be rectified easily later on by running queries to chop and change data, however other decisions may not be able to be rectified so easily. On the other hand, don't get too tied down with perfecting your database tables, you're more than likely going to be able to change them later. However, do give future thought to your structure when planning.

Data types are equally important, and there are so many to choose from, which can sometimes be overwhelming. For the basics, something like a unique user ID could be stored as an int. Likewise, if you were only storing a number 0 through to 9, you would choose a tinyint (a one bit integer). These small decisions will ultimately lead to a faster, better designed database that has been appropriately designed around the data it holds. Remember, data types can easily be changed. If you're unsure, it's best to choose what you think would be suitable and then research what you've chosen later if you're worried about performance or your overall database design.

Including an index within your table will ultimately lead to faster performance, as long as other considerations have been taken into account to address speed. Keeping a field as an index, for example, an auto incrementing user ID will mean that querying this table will be faster, as a unique pointer will have been set up. This is common practice when creating a table, as you'll need a unique identifier for each user anyway. However, don't think you're just tied down to integers for unique, indexed columns, you can also choose other fields to index. We'll discuss this in a bit more detail when we look at speed.

On the more advanced side of database table setup, you should choose an appropriate storage engine for the tables you create. These have a massive impact later on down the line depending on how you're interacting with your database. For example, if you ended up with hundreds of thousands of records within a table and needed to search the table, you may have chosen an inappropriate storage engine for searching the table efficiently, e.g. you may have chosen InnoDB instead of MyISAM which allows for full-text searches.

Lastly, it's a good idea to populate your database with some dummy data. On the other hand, you may find it easier to build the functionality to insert data first, and this also largely depends on the kind of website you're building, and whether this data will ever need to be output to a user. Personally, I've always found it easier to insert the dummy data, output this within the application and then build the functionality to insert. This gives the joy and satisfaction of almost seeing your application work. Again, if your application isn't input/output driven, you may not find the same satisfaction. It's up to you which way suits you personally and what makes sense.

13.3 PDO

PDO (PHP Data Objects) is a flexible way to communicate with your database. It provides a way to connect using database drivers (one being for MySQL) and is essentially a unified way of database connection and interaction. This means that you can switch between databases and your website will still be able to communicate with your database providing the table names remain the same. PDO also offers powerful functionality in OOP, which makes it great to build into your code. It also offers transactional support which means that operations can be rolled back providing your database supports this, making operations within your database more efficient.

13.4 Choosing how to communicate with your database

I've assumed that you're using a MySQL database, as it's an extremely popular database solution. Best of all, it's free and very powerful. PHP has much built in support for MySQL and makes it extremely easy to connect, query and retrieve results in a variety of ways. You may have seen or used the MySQL extension which includes functions like mysql_connect to connect to your MySQL server, mysql_query to send a query to the server and mysql_fetch_assoc to retrieve a result set in an associative array. These groups of functions are called extensions simply because they're not part of the PHP core set of functions and are instead part of the PHP extension framework. The first thing to mention is that PHP advise against the use of this extension and instead recommend the newer MySQLi extension. This provides exactly the same functionality as you'll have previously seen albeit improved a great deal, hence the 'i', which stands for 'improved'. The MySQLi extension set is available both procedurally and object oriented, so they can be used in either scenario. Let's take a look at the difference between these two when querying:

```php
// procedural
$db = mysqli_connect(server, username, password, database);
$result = mysqli_query(query);
$assoc = mysqli_fetch_assoc($result);

// object oriented
$db = new mysqli(server, username, password, database);
$result = $db->query(query);
$object = $result->fetch_object();
```

The procedural example shows connecting, querying and returning an associative array of our results (which would, in practice, need to be done in a loop if there were more than one row returned) exactly the same way we would with the mysql extension. This is fine, because you're doing exactly the same thing, but with the improved extension.

In the second example, we instantiate a new MySQLi object which gives us access to some methods and properties defined within this class. We use the query method to query the database with an SQL string and then we return the result set as an object. This is interesting, because with the previous example we return the results as an associative array, which you may be used to. So, why do we need to store the return value of the query method within the $result variable? The answer is that it returns the mysqli_result object, which then contains the fetch_object method for us to obtain a result set. To examine the returned object, you can make use of the get_class_methods function or get_class_vars similarly for properties. For example, we know that the $result variable contains the returned mysqli_result object, so we print_r on the get_class_methods function, which returns an array of methods from whatever class you pass to it.

```
print_r(get_class_methods($result));
```

These are extremely simply, meaningless and useless examples, however they demonstrate the effective nature of using MySQLi with OOP. It allows you to be a lot more expressive with the way you work with your database and it's returned data after queries. If you're working procedurally, it makes sense to use MySQLi procedurally, and of course the same for OOP.

13.5 Interacting with your database

Once you've set up your database, tables and fields and have populated your tables with some dummy data, the next step is to start thinking about how you'll interact with this data. This vastly changes depending on what type of website you're building, but it's a good idea to start writing code to do what you need. Let's say you're building a forum - it's obvious that you need to read and write posts, so start with building up the functionality that will do this. You'll need to think about how to output certain areas of the site, like the forum categories, topics within categories and posts within each topic. This may seems daunting at first, but actually it's very easy. Let's look at this as a more step by step process:

1. To list your categories, you'll need to output them within a list. You'd query the database for all categories, output them wrapped in anchor (a) tags and have the link something like category.php?id=x, where x is the category ID from the database (this would more than likely be your primary key, set up as an auto incrementing integer). Bear in mind that SEO isn't used here, so we're using the raw format of the file and GET variable we're passing.
2. When the user clicks this link, they'll be directed to category.php, where the ID will need to be read. There are important considerations here, like security, as this data can easily be changed by the user. You may also consider categories that don't exist, but this could be incorporated into the main query that pulls topics from this category. Now, to pull in topics, you'd have a separate topics table. Each topic would have the category ID field set to the category it belongs to, and therefore you could query this table based on the category ID that's been defined within the URL. Now it's just a case of outputting the topics as you have the categories, and wrap

them in anchor tags again, linking to a page - perhaps topic.php?id=x. When you think of linking everything up like this it becomes much easier to think about how to transfer your user through to different pages, using GET data.

3. From here, it would just be a case of listing posts from the posts table that relate to the particular topic (again, we've specified this in the URL). This is the basis of most applications. It means you can build one template and use it for everything.

We'll be touching on security later, but as a quick note, and as we've already briefly discussed, it's extremely important that you take security into consideration. Absolutely anywhere that a user can modify data being sent to your server needs to be considered as it can be manipulated in a way that allows an attacker to take advantage of this. It also helps to include commonplace checks, like whether a particular topic exists, in case a normal user either purposely or accidentally changes the URL, and specifies something incorrectly, or a topic is deleted and a message needs to be presented to the user.

Of course, there is a lot more to creating a fully functional forum, but we'll be discussing various other example applications along the way so we can start thinking like a programmer and address various other problems.

The main thing to think about is that the fundamental part of many dynamic websites is just grabbing data, showing it to the user and allowing an administrator or user to post data for later retrieval. It's the cycle that once mastered properly, will make building dynamic websites a lot easier.

13.6 A Guestbook example

Although guestbooks aren't really used anymore, they represent the fundamental functionality for the majority of dynamic websites.

So, let's take a look at the thought process that would go into building a simple guestbook, where users could visit a page on our website to leave a comment. This functionality could be easily extended to blog posts, forum topics and much more. The key here is learning how to think about setting up tables so they can handle what you need to do. Once you've grasped this, you're free to explore greater things.

Let's start with thinking about what fields we need for storing information. A user will enter their name, email address (optional) and a message. But, that's not all we need to store. We want to know what time they made the post and we also need to keep a unique field for indexing purposes to help speed up retrieval of data, and identify each post uniquely. So, the fields we need are:

- post_id
- name
- email
- message

- timestamp

But what about data types, lengths and default values and required values? Our post_id will obviously be an integer, our name, email and message a string and we'll be storing the timestamp as an integer. So:

- post_id (int)
- name (varchar)
- email (varchar)
- message (text)
- timestamp (int)

Notice we have varchar and text. Varchar will require a fixed length whereas text doesn't, although it's advisable to give one. The int fields also don't require a fixed length, although for the timestamp we could give one and assuming you're not going to receive millions of posts, we can also limit post_id. Below is the amended fields with a brief explanation as to why we've given these the lengths we have.

- post_id (int 5) - This limit would still allow for 9999 posts. More than enough for now, but can be changed in future if this limit is approached.
- name (varchar 25) - This can vary dramatically, but 25 is more than enough characters to store the average first and last name. We've included a little more in case a large name or middle names(s) are included.
- email (varchar 85) - An email address is unlikely to be more than this, but emails can be up to 1024 characters.
- message (text 1000) - On the front end, we'll be limiting characters count to 1000 characters, we'll also be checking this within PHP. However, it's good practice to also define this within your table.
- timestamp (int 10) - Currently, and for years and years to come, a timestamp will be 10 digits.

So, we've got the basis of our table and have everything fairly precise. But what about our field contents? We want them all to be mandatory except email, so we should define this. You'd need to include the NOT NULL property when defining these fields, and leave the email field as normal, so this isn't defined as required. This isn't required, but is advised.

All that's left now is to write backend code to start interaction. Remember that it's important to include validation and security when querying a database where user data is submitted or properties are defined when retrieving data (such as how many posts to display per page, etc.).

Why have we bothered to define all this when we can just lump it all into our table and not declare specific types, sizes and whether a field will be null? Well, of course you could store a username in a field with a text data type, but you'll be ruining the design of your tables. You could also choose

30 characters for a varchar field for a username, but why would you when you might be validating within PHP to only accept 20 character maximum? Your database ties into your website and most likely cannot function without it. If you're storing information and querying for data, design your database carefully and the transaction will be more harmonious and save you time in the long run.

14 Generic functionality and configuration

Generic functionality defined patterns that allow you to re-use code for a variety of purposes.

It sounds boring, but generic functionality and somewhere to define configuration within your website will save you hours, perhaps even days in the long run. Consider a website that outputs articles on several pages. Perhaps you have a main news section, a featured article section and for each author on your website you list a short list of their most recent articles on their profile page. Are you going to build the functionality to output articles for each instance? Certainly not!

14.1 Global configuration

Configuration within your website is extremely important, particular as it grows and may become unmaintainable. Global configuration, in short, means somewhere you can specify values that may be used once or more than once throughout your application. The reason for this configuration is that it makes it easier to change functionality from one place, normally a configuration file. Although, this can be from a database which may prove easier if you've set up an administration area, from which these values can be changed.

Let's first take a look at flat file based configuration, how we may create a configuration file and how we make use of our values. One of the easiest ways to create configuration within a flat file is simply to create a new file, create an array within this file to store your configuration settings and then include this file where you need (almost always globally). The reason we use an array is because this allows us to easily create configuration groups which house different configuration options. Let's take a look at an example, which may be defined within a file called config.php.

```
$GLOBALS['config'] = array(
        'mysql' => array(
                'host' => 'localhost',
                'username' => 'billy',
                'password' => 'N&G8>di4ab0x',
                'db' => 'site'
        ),
        'articles' => array(
                'initial_count' => '10'
        )
);
```

Here you can see an associative array, with each inner array containing a group for configuration options. You'll also have noticed the variable we're applying this array to is $GLOBALS['config']. This isn't necessary, and may perform slower if you were to do it this way, particularly if you had a very large array, although the speed difference would usually be unnoticeable. If you're declaring configuration within a class, you may want to make this global to a class so you can reference this only where your class is instantiated (and presumably then where you'd need these configuration options as part of your website functionality).

Now you've defined your configuration options, you'll need a function to deal with returning the values based on what you pass to this function. This could be a standalone function, or be part of a class as a method, depending on how your configuration array is defined. An example of a function that deals with returning values may look something like this:

```
function conf($group, $option) {
        return (isset($GLOBALS['config'][$group][$option])) ? return $GLOBALS['config'][\
$group][$option] : false;
}
```

So, this will return the value requested based on what's passed to $group and $option. This is a very simple example, but what if you wanted to create arrays within your main groups? You could of course add another argument to your function, like:

```
function conf($group, $first, $second) {
```

But, this is still unmaintainable. Just think what would happen if you wanted an even deeper collection of arrays! For this, you may need to consider a better way to address this, and we can make use the func_get_args function to help us. Here's an example of how you may implement a function to read any part of your global config array, any level deep:

```
function config() {
        foreach(func_get_args() as $arg) {
                $value = isset($value) ? $value[$arg] : $GLOBALS['config'][$arg];
        }
        return $value;
}
```

This function defines no arguments to be passed in, but we use func_get_args (returning an array of arguments passed through to the function) and loop through this. We then check if this value is set, and if so, move to the next value. We eventually return this value. This means we can make use of the function like this:

```
config('articles', 'homepage', 'initial_count');
config('articles', 'article_page', 'initial_count');
```

14.2 Database based configuration

If you don’t require as many groups of options, you can choose to store them in your database. This is also useful if you’re building a backend for administration and need to change website settings from here. Be aware that this option may also be slower, as reading options would require a query on every page where options are required to be read. These could be cached, but could still be slower and if you don’t have caching in place this would not be feasible. To set up a configuration table within your database, you simply need to consider key/value pairs. So, a table may contain the following fields:

- option_id
- option
- value

You could still group data here, perhaps using an underscore for groups. For example, if you had options for articles, you may have an option like articles_initial_count and assign a value to this. You can probably already see how this may start to get messy and would become unmaintainable with many configuration options. Unfortunately, this is just part of the table structure, and would otherwise mean you’d have to create many tables for different configuration options. This isn’t a good idea, would impact speed and would generally be a mess and hard to maintain. If you do choose this option, how do you go about pulling configuration values from your database? One query, either globally or defined within you class (perhaps within your __construct method in your core class) will then allow you to return an associative array of this result and then start to access settings. We won’t look at any specific examples of how to set this up, as this could take many forms depending on how everything has already been set up. Storing configuration data in a table may also be more secure as it’s not prone to being revealed as a flat file as easily. Just ensure that this method of getting settings is absolutely necessary, considering the negatives we’ve already looked at.

One last consideration is that you’d be unable to store MySQL connection information within your database table, simply because you need these values prior to connecting to your database! You should store these in a configuration file to then connect to your database. From here, if you’re using your database to store your configuration, you can then build the functionality to retrieve it.

14.3 Generic data output

As with the example above, you may have areas of your website that output the same thing, in the same format, only based on different settings each time. We’ll get straight into an example of this, continuing with the idea of outputting articles within the same website, but varying slightly each time. We'll assume a design has already been built for the listing of articles. This could be a series of div elements that would be output for each loop of your records, outputting data for the current article.

The first thing you'll need to assess is exactly what will vary throughout these lists of articles. Assuming what we've discussed above will happen, these will be:

1. Result limit (amount of article to appear)
2. The field to sort by
3. Order of sorted data (ascending or descending)
4. Ability to specify a particular field to filter in (e.g. a particular user).

Once we've established these settings, we build a method to take arguments, which we'll later process within the method.

```
public function articles($filter, $sortBy = 'timestamp', $sortOrder = 'desc', $li\
mit = 25) {
        // nothing here yet
}
```

At first glance this is very simple, and it'll also save us a lot more time later on, as it's more maintainable. Notice the default values for if these arguments aren't passed.

So now we come to actually building some of the inner contents of the method. This isn't a full scale solution so we'll just build something very simple. We're also assuming that there is database functionality to query and return results, so this example won't work straight out of the box. Also note I'm using SELECT *, which should be avoided, and you should specify explicitly the fields you want to retrieve. I've done this simply to save space.

```
public function articles($filter = null, $sortBy = 'timestamp', $sortOrder = 'DES\
C', $limit = 25) {
        $sql = "
                SELECT * FROM `articles`
                " . $this->escape($filter) . "
                ORDER BY `". $this->db->escape($orderBy). "`
                " . $this->db->escape(strtoupper($sortOrder)) . "
                LIMIT " . (int)$limit;

        $this->db->query($sql);
}
```

From this you'll now need to output your data with your chosen database handler and within your written markup. Below is an example of how this may be done, very basically. You may have other details to include, but this generally shows looping through article results with the most basic of data.

```php
// continued
$this->db->query($sql);

foreach($this->db->rows() as $article) {
        echo <<<HTML
        <div class="article-list">
                <h2>{$article['headline']}</h2>
                <p>{$article['teaser']}</p>
        </div>
        HTML;
}
```

Notice we're also using the heredoc syntax to output the HTML, which we looked at earlier.

We've now created a method that allows us to output a list of articles with the chosen markup, with the ability to specify configuration options. You can add more configuration options to the argument list to suit your needs, and you will more than likely need to do this at some stage. Before adding anything else to the argument list, consider how you might reduce the amount of arguments by combining options into an array of data and then exploding or looping through it within the method. All this takes is careful planning beforehand, however if you do build in functionality that doesn't make sense, or is difficult when you come to implement it, you'll soon spot the difficulty and will be able to make changes. And as always, if you're passing arguments directly into a query string, ensure you're escaping and formatting any data that potentially lead to SQL injection or data inconsistency.

Also remember, if you notice that you're repeating any code, you could probably avoid this!

14.4 Translations

At one point or another, you may need to offer your website content in other languages. There are many considerations when using translations, as you need to know how deep you'll need to translate. Will this be site text only, or will you need to localise article content too? You may provide one article with the content separated into many records, all in different languages to be output based on the locale chosen. We won't be talking about how you can provide your users with the option to switch language, but we'll be looking at the options we have for the backend translation and implementing this, including a look at how you may translate non general site content like articles.

Let's firstly take a look at how you may go about translating site text, so that a locale is provided by whatever means, which would then go and fetch site text based on this. We're not going to be looking at many examples here, as there are many variations to the process of translating site text. We're going to be looking specifically at gettext, PHP functionality that allows you to output text based on locale, but also with the added benefit of being able to generate translation files (with a .po extension) that contain your website translations, by scanning through all of the files that make use of your gettext output. This gives the benefit of being able to write your code as you normally would output everything, for example:

```
echo gettext('Hello') . ' {$username}';
```

The gettext function is what does the magic of translating this text, so you would have 'Hello' output in different languages when the locale is switched. Bear in mind you have to do the translations yourself, so don't try to implement this where you haven't generated language files and translated everything. You may find it tedious to write code using the gettext function for everything you need to translate, so PHP provides a shorthand function for this, a simple underscore. In this case, the above snippet of code would look like this:

```
echo _('Hello') . ' {$username}';
```

This is much better as I'm sure you'll agree, and makes for more natural reading when scanning through your code.

So what now? You've written your code with the _ function, outputting all this text, but now you need to generate the files that will be read depending on which locale has been selected. This will then fetch the associated text and output. The steps to doing this are fairly simple as gettext provides utilities to sweep through your code and find any text wrapped within the function you've chosen to use (either _ or gettext). You'd then hand these files to a translator, or translate them yourself and place these files where they can be read by gettext. There are many resources that explain this process and exactly where they need to go, so we won't waste space listing everything out here. However, you now know that you can use gettext to easily translate site text.

14.5 Translating other site content

You may have other site content like articles that will require translation, if not now, into the future. The ability to translate these will also mean the content will need to be written in the language it's intended for, and then somehow marked with the locale for the language. For example, if you have a database table for articles, you may keep the 'body' of the articles in a separate table, and mark each record with the locale. Your main article table would contain a body_id field that references body content stored in the table articles_body, for example. This means that a field, locale, can be incorporated into the articles_body table. So, when querying the articles table for an article, you'd join this to the article_body table and pull in the content. This could then be based on the fact that a locale matches up. You could even fall back to the default language version if the specific locale's body content isn't found. This makes it really easy to then take the locale that is currently set and pull in localised content. For example, if you're already using gettext to translate site language, you may have a variable declared that changes based on the locale selected:

```php
$locale = (isset($_COOKIE['locale'])) ? $_COOKIE['locale'] : 'en_GB';
```

In this example, the locale is set based on a cookie. If this isn't defined, it defaults to en_GB. This can then be used with gettext, but can also be incorporated into queries that return localised content. Remember, the above example uses content that can be set by the user, so remember to escape values like this before passing them to queries.

Implementing these methods early doesn't mean you have to start translating your content to other languages straight away, but it means you're all set up to provide content in other languages later on. Imagine building your website with vast amounts of text and content without considering this beforehand. If demand for your content in other languages increases, you'll have a lot more work to do to turn this around if not initially implemented!

15 Commenting code

Commenting code is an extremely underutilised practice. Why? Because more often than not a programmer will assume that down the line, they'll know exactly what the code is, what it does and how to change the code if required. This is absolutely natural, however many of us have become victim to opening a file full of code that we wrote days, months or perhaps years ago, only to find that we have a much harder time understanding it than we anticipated.

If you've ever opened up someone's code and not understood it as a whole, but know deep down that you understand the fundamental parts that make it up, then this is exactly what you'll encounter when you open up some uncommented code of your own.

Recommending that programmers comment their code isn't enough, because a comment could be anything. When we're talking about commenting, we're not talking a little note here and there, we mean a full explanation of what something does, why it does it, how it does it and what could be improved if required. Of course, this extent of commenting would be silly for a line that simply output some contents, but if these contents were contained within a variable for example, it would still make sense to make a quick note of what it's actually outputting. This would make it much easier to scroll down the application, see this output building up and then quickly being able to spot where it's being output.

Commenting helps the author understand their own work when it's reviewed at a later date, but also for other developers. As we've already mentioned, other people may join a project at a later date, and commenting makes it a lot easier for someone to quickly scan an application and gain a good understanding of the code and what each part does in relation to the code as a whole. This of course depends on knowledge of the language, but you'll be surprised that giving a well commented file to someone learning a programming language can actually really help teach it.

Another good reason to comment brings us to the very top of a file's code, the meta information about a file. This can be very useful to display meta data such as when the file was created and modified, the author of the file and a brief overall description of what the file does. You may also see information here about how the file can be used. For most of this meta information, a text editor can be set up to automatically insert or update this information. So, every time you save a file, the note to say when the file was last updated could be automatically changed.

So, why is it important to include author information and the date of creation or modification? With regards to the author information, it makes it a lot easier to get help with the file within a team if you know who created the file. In open source software, you may need to contact the author with important information, such as a discovered bug, security flaw or recommendations. Put simply, the date will tell someone how old the file is, which could be used to flag any files that may need to be reviewed for deprecated functions. Functions or methods will have been deprecated for good reason, so reviewing old files will mean that an application can be kept up to date easily.

16 PHP Documentation

The PHP documentation is an absolute goldmine when used properly. This doesn't mean that you should open it up and use whatever you read, as some examples have been specifically created to demonstrate methods that shouldn't be used. You'll also find a host of user posted information that should not just be plucked out without proper research and thought.

16.1 Finding what you need

You'll find everything you need within the PHP manual, but finding it can sometimes can be tricky. A Google search for what you need to do (e.g. split a string by a particular character) will usually return what you need as the website is indexed fairly well in Google. As an example, if you needed to know how to split a string by a particular character, you could search "PHP split string by character". At the time I'm writing this, it currently returns the explode function within the PHP manual as the first result, which does indeed split a string by a particular character. Although successful, there is another function that splits by characters and regular expressions, preg_split. This may work better, for example if you needed to split by one or more spaces. The explode function will not let you do this, and therefore you'd be stuck if you were none the wiser about the preg_split function. PHP has introduced a 'see also' section to their manual which will allow you to explore other functions or topics that may be related to what you're looking at. As you may have guessed, within the explode page the first 'see also' result is preg_split. It's always worth browsing and checking out alternative or extended features before using what you've had returned as the first result.

16.2 Deprecated functions

Also known as 'functions that shouldn't be used any more, but can't be removed from the language', deprecated functions should be avoided and replaced where they have been previously used. Of course, these can't simply be removed as an upgrade of PHP would mean that thousands of applications would simply cease to work. For this reason, they are deprecated and another function is introduced, or another function is recommended to be used. The PHP manual will tell you if a function has been deprecated and will give you the alternative use, so ensure you look out for this when looking in the manual.

Be aware that not everyone looks out for deprecated functions and keeps up to date with the manual, so for example, a recent blog post that uses ereg_replace (to replace content within a string using a regular expression) doesn't mean that it should be used, as this was deprecated as of PHP 5.3. It's good practice to look up every function you're using when referencing code from another source, as this could leave you with speed or security problems within your application. Or, a replacement function could simply provide you with the functionality you need that wasn't previously available.

Remember, PHP includes a list of deprecated functions with each release, so review this and check if you're using them, and subsequently go ahead and replace them with newer functions.

16.3 Cautions

When browsing the PHP manual, you may find user posted comments. These contains code posted by the public and although somewhat moderated, the code contained within the posts may not be ideal for your application or in general. When referencing any code or advice posted here, make sure you seek advice from elsewhere before simply using it. You could also get involved by actively participating in the feed!

You should also bear in mind that simply plucking functions from the manual may not suit the environment you're working in, may not be available in the version of PHP installed on the server or may not work well within the code you've already written.

16.4 Browsing for help

It's easy to browse for help with any popular programming language, and PHP is no exception. A quick search for almost any PHP related question will return vast amounts of resources with tips, code snippets and general advice. The problem with this (as with searching for any information online) is that anyone can post something, regardless of experience. When searching for help, always question and research any advice or code you've found and if you're unsure, ask within a well-established community where you're sure experts can give you advice.

One of the best ways to get help is to be part of a community where more than one person can offer advice. You may get several responses, but those who respond to you may question each other and discuss best practice, speed or security concerns. This means you get many points of views.

Actively participating in any community can be extremely rewarding and should help you learn a lot more about PHP and web development in general.

17 Speed

At times you may find that your website is running slow, particularly when running queries that return large amounts of data or are inefficiently constructed. In this chapter we'll take a look at some speed considerations, including what you shouldn't be doing when writing code, particularly in terms of your database, and how you can start to speed up your website.

17.1 A method for testing your page server load time

Your server side page load time is the time passed since the code starts to be interpreted to when it was processed fully. This means you can store the time at the very start of the code and subtract this from the time at the very bottom of your code. Remember, this means the very first lines of your code. Add the following line:

```
$startTime = microtime(true);
```

This will get the current timestamp represented in microseconds. We pass true to return this value as a floating point integer (e.g. 1.375).

To finish, all you'll need to do is a simple calculation at the very bottom of your code which will output the result:

```
echo 'Page loaded in ' . microtime(true) - $startTime . ' seconds';
```

Run your page and you'll see how long your page took to process server side. Remember, this doesn't include the rendering time of your HTML, CSS or JavaScript, that's different. You'll need to use browser tools to monitor the page load time of any requests from your page. We'll discuss these next.

17.2 HTML, CSS, JavaScript and other HTTP Requests

Some would argue that the size of a CSS or JavaScript file doesn't matter too much, as the contents of the file is almost always cached by the browser being used to access it. This is true, however the initial load time of an unnecessarily large file doesn't make sense when steps can be taken to dramatically reduce the size of it, through carefully writing code, minifying the contents of these files and serving them compressed. Each of these steps can reduce a file significantly and can noticeably reduce your page load time. Considerations also need to be taken into account when serving content to mobile devices with poor connection speeds too. We also need to consider the amount of markup on a page, as this needs to be downloaded. Even if you have caching mechanisms in place, this won't have any effect on the time it takes to re-download the markup every time a page is requested.

Reducing HTML/CSS/JavaScript file size

We'll talk about CSS and JavaScript first, and then talk about the quality of HTML making a difference. CSS can be poorly written, resulting in duplicate styles, unnecessarily long selectors and/or chained selectors, not making use of shorthand style rules and including style rules for specific browsers. The list goes on.

Unfortunately, writing clean CSS comes with experience, but there are some steps you can take to push you in the right direction. There are various standards written on CSS that are good to follow as part of a project. You might also find using

There are a variety of plugins available in different browsers that allow you to monitor the use of CSS within pages you're viewing. I won't mention a specific plugin, as there are a few available through different browsers. Search around and find something that suits you.

You can also run your CSS through the W3C CSS validator. This will also pick up on any invalid CSS and let you know. Probably the best way to deal with CSS is to think about each selector as you write, and the process of writing good styles will eventually become natural. For example, let's say you're targeting an element, .name within an unordered list housed in an overall person container, .person. You could do:

```
.person ul li .name { }
```

Or, you could simply write:

```
.person .name {}
```

This of course, is only if you don't need to be too specific as to which elements you're targeting. Spending time going through your entire stylesheet and checking for small amends you can make like this one can save you a lot of bytes.

And, the same goes for JavaScript. Your code can be condensed in such a way that you reduce the amount of code you write. This is slightly trickier, and you generally need to know a lot more about JavaScript to be able to refactor it in this way.

Finally, you can minify CSS and JavaScript very quickly and easily, which condenses it into an almost unreadable state, usually based on a single line with variables, functions and more replaced with smaller identifiers. Minification is a quick and easy way to reduce the file size of your CSS and JavaScript before you place it within a production environment. Be aware that on occasion, minifying your code can result in problems with its functionality. Good minifiers provide levels of magnification that can be controlled, so you can specify how deep the minification should be. The last thing you should do is simply minify and go live, ensure every page is tested thoroughly first.

17.3 Loading resources from a CDN

A CDN (Content Delivery Network) is an infrastructure built to serve cached content from a location nearest to the person requesting the data. CDNs usually provide different levels of caching to suit your needs.

Let's take jQuery as an example. You could upload the latest version of jQuery to your server, include it and it would be downloaded to a user's computer, and cached by the browser for the next time it's requested. Now think what would happen if you serve the file from a CDN, such a Google's or jQuery's own. The file path would look something like http://code.jquery.com/jquery-1.8.2.min.js or http://ajax.googleapis.com/ajax/libs/jquery/1.8.2/jquery.min.js. If a user has visited another website that uses the same version of jQuery (extremely likely) and their browser has already downloaded and cached it, it won't need to be downloaded again when they visit your website. This will reduce the need for an additional HTTP request and therefore you'll be closer to a faster website. Delivering content from someone else's CDN isn't the only way you can serve files. Files specific to your website such as logos, any uploaded images, video files and other resources can take advantage of a CDN. It's more than likely that the delivery network will be faster than your server at delivering files efficiently and therefore will help to speed up requests. This is particularly useful for large downloads or streaming video content.

17.4 Rendering time

We'll only touch on this because we're mainly looking at server side speed. This is still important because what you output with PHP will affect rendering speed within the browser. Let's say you were outputting forum posts. Carefully choosing the way to structure the markup within a loop will vastly reduce the amount of HTML you output, increasing the speed in which your PHP is processed, and also the rendering speed of your browser.

Bear in mind to also avoid inline styling, which is bad. Also avoid inline JavaScript, and ensure you've set up your page so your browser can cache external CSS and JavaScript effectively. Loading resources like CSS and JavaScript inline (or lumped on the page) means they won't be able to be cached, resulting it larger download sizes and effectively slower load times.

If you need more help with this, use a well-respected tool like Google Page Speed[1].

17.5 Limit queries

The first thing to analyse is how many database queries you're running per page, relative to how much data you're returning. If you're querying in loops, this is bad. If you're querying more than once to retrieve data from two or more tables where data is linked, this is also bad. The more queries you're running per page, the slower your page will be. This is because your website will send a

[1]https://developers.google.com/speed/pagespeed/

request to the database, wait for a response and will then read what has been sent back. Each request generates network packets that will slow down your page. The answer to this problem is to get better at SQL in terms of joining data from multiple tables within the query so fewer requests are required.

Let's say you're returning a list of posts by a user. The table defines the user by a user_id field, and the username isn't present as this is stored in the users table, not the posts table. The bad way to do this would be to loop through the posts, and then place a separate query within the loop to retrieve the username based on the user_id. You'd be generating so many queries, if not now, then into the future when a result set grows. You'd be able to easily return this data in a different manner. Let's take a look at an example. Our tables are posts, and users.

```
SELECT `posts`.`title`, `posts`.`timestamp` `users`.`username`
FROM `posts`
JOIN `users`
ON `posts`.`user_id` = `users`.`user_id`
ORDER BY `posts`.`timestamp`
DESC
LIMIT 10
```

So, we're returning all the data we need, plus an additional field, username, which comes from the users table, being matched by user_id. This is only one way of joining, but we can also use joins like LEFT JOIN and RIGHT JOIN depending on how your data may sit in either table. In this case, when a post is made, we know that a user will exist in the users table, which will be the user who made the post.

17.6 Optimise queries

You may also find that particular queries are performing slowly. Basic queries pulling data from a couple of tables shouldn't be a problem, but that also depends on how much data you're gathering. You may be familiar with the SELECT * syntax, and may be currently doing this to pull data from a table. Selecting all field data from a table may not be necessary and therefore doing this may be giving you unused fields, thereby slowing down your query unnecessarily. This can be particularly slow if you're joining and then subsequently pulling all data from the joining table, when in fact you may only need to pull a username, like in the example we've just looked at. If you only need to return the username of a user, only select that data. Selecting all other data within a table isn't going to help you in terms of speed, and also gives a clear indication as to what data you're retrieving when you look at the query.

Defining which fields you need to pull isn't the only way you can optimise a query. You may also be joining in such a way that your queries perform slower or using slow MySQL functions within your query.

17.7 MySQL or PHP functions?

You can easily perform functions within queries to concatenate, format and organise data returned. However, you can also do this within your application layer. You may find that a particular function has been optimised well within MySQL than it has in PHP, and you can measure the speed difference to determine what's better to use. It'll be a lot better to test this yourself instead of relying on others to tell you what's better optimised. This is due to the fact your data set or processing may be different or their advice on a particular method may be outdated or inaccurate. Using the method stated at the start of this chapter should be enough for testing this with large pages and large data sets.

Normally, this kind of thing doesn't affect speed noticeably, but that depends on how extensively it's being used. If something has been optimised to perform better within MySQL and you need to work with a lot of data, do it there. There are some cases when pulling data from a query and then working with it in PHP can have its advantages. If in doubt, runs some small tests to see what performs better.

17.8 Searching a database

It's common to need to search a database for data, particularly if your website relies on data to be pulled back from a user search. Using the LIKE keyword can have a drastic effect on speed if not used properly, or used in the wrong situation. If you have a small collection of rows in a table and need to allow a user to search this, LIKE may be suitable as long as the data isn't going to grow and grow in volume. If you do end up with a lot of data here, your website speed is going to decrease as time goes on and more rows need to be searched. For large sets of data, performance time can be terrible. As a real life example, searching for a single keyword within 100,000 forum topics returned an average of 20 seconds. You absolutely can't afford for a page to take event 3 seconds to load, let a loan 20 seconds.

So, is there an alternative solution to this? Well, yes, there a few things that can be done to reduce the amount of time for queries like this. We'll look at two. The first we'll look at is basic, and the second slightly more advanced.

Add a fulltext index

A fulltext index can be used on a table with a MyISAM engine type and can dramatically speed up searching for words within a table. You'll need to add a FULLTEXT index type on the field the search will be performed on. A query may look something like:

```
$sql = "SELECT `id`, `headline` FROM articles WHERE MATCH (headline) AGAINST ('{$\
keyword}')";
```

Use a search server

Two popular search servers are Lucene and Sphinx. Lucene is slightly more complicated to get started with so I'd recommend Sphinx if you're new to this. This basically takes over the search process and returns relevant information from the search term provided. Because MySQL isn't greatly equipped to handle text search, a solution like this is perfect as the process is passed on to this type of software. Typically, the data will be ingested into the software and will then be searched.

17.9 Looping

Looping can cause massive problems as we've already seen within queries. Using PHP functions within queries that have poor performance can cause speed issues. You may also find speed issues if you're looping through very large sets of data returned from a query or within a file, e.g. a large xml feed.

You may have also come across the problem of looping infinitely. This may be easy to change, as you can alter the condition within a loop. If you're using a variable in the loop's condition that is likely to change, ask yourself whether it could cause an infinite or large loop at some stage.

17.10 File reading

Reading files can pose a problem if using out of date methods or reading and processing large files. Reading large files will always be a problem, but you can avoid reading a large file at once by reading it in chunks. There are several methods you can use to do this, depending on how much control you have over the data.

Also ensuring you're not using deprecated methods will ensure optimum speed. For example, fopen should not be used, but instead file_get_contents due to speed changes. The latter function is optimised per operating system, so you know you'll get the fastest writing available to your server. This is just one example, but it's always best to check the PHP manual to avoid using slower, deprecated functions.

You can also limit the amount of data you read in from a specific file. So, reading a 2mb file doesn't mean you have to access the entire file if you don't need to. You'd simply specify the maxlen argument within the file_get_contents function. Simple decisions like this add up in terms of overall speed.

Saying this, it's doubtful you'd need to read from a file where you could store the data in a database, so think carefully about where you're storing and retrieving data from.

17.11 Limiting user defined values

If your website allows users to change values such as how many posts are displayed per page, this needs to be taken into consideration with speed. If a user can change this value, then they can set this value to anything. So, in short, any data that can be defined by your user needs to be limited. To do this, a simple IF statement or ternary operation when injecting this value somewhere will allow you to set the defined value to a default if the value defined is too high. Ensuring these checks are in place for anywhere data can be manipulated will mean you can avoid excess bursts of data being processed and affecting your server speed.

17.12 Caching

For high load websites where many visitors are accessing the same information again and again, it doesn't make sense to serve the same content by making a request to your database. This is where caching comes in. Caching allows you to store data that would otherwise have to be generated by the server and therefore reducing on load to the database. This of course wouldn't be cached forever in all cases, and your cached data would have to be refreshed at some point to provide the updated version of the data. Caching is particularly useful for data that will be accessed time and time again by everyone, or for data that just doesn't change very regularly.

We've already briefly discussed CDN caching, but there are ways to cache directly with PHP. We'll take a look at some of the options and a brief overview of each one, as your requirements for caching will certainly vary depending on your website.

PHP Accelerators

There are several accelerators that can be used in conjunction with PHP to cache data either file side or in memory. This includes caching entire queries (although MySQL has its own functionality to cache queries too) and can dramatically speed up access to data that is requested time and again by lots of people - particularly if the data is unlikely to change for a long time, in which case it can be cached for a large period of time. Some website data can be cached forever, meaning that it will always be read from a cache unless flushed. Flushing data will simply generate content and place this back in the cache. If, for example, your menu is generated from content within a database table, you could probably cache this for a long time, or forever. Only when you make a change to the menu would you need to flush the cache and pull through the new changes.

Memcached

A popular and somewhat easy to set up caching solution for PHP is memcached[2]. Although not exclusively designed for PHP, memcached is an effective solution to cache in memory and is widely documented with many examples. Setting up something like memcached isn't too tricky, and with the documentation and perhaps a little browsing around for help, you could be up and running with memcached with a couple of hours. There are then several techniques that can be used to cache different aspects of your website, including caching data retrieved from queries (you're not actually

[2]http://memcached.org/

caching the query, you're caching what is returned from it), PHP variables and of course, any output like HTML. If you're new to memcached, it's best to try memcached in isolation instead of trying to directly tie it in with any code you already have. Trying some quick examples will let you slowly get to grips with using it and will help you start to understand how to check what's cached and what isn't. As mentioned earlier, memcached isn't exclusive to PHP and can be used with other languages.

eAccelerator

Another option is to use eAccelerator[3]. eAccelerator is a general purpose optimiser that can be very easily installed and will get to work straight away at caching files either on disk or in memory to speed your pages up '1-10 times' as claimed by eAccelerator. There is a certain lack of control with eAccelerator, which means it's not ideal if you need very specific caching rules.

APC

Similarly, APC (Alternative PHP Cache) can be used, and will be bundled with PHP 6 (not yet released at time of writing). APC is very easy to use and works similarly to memcached whereby you need to define what to cache and when to retrieve it. APC is fully documented with examples in the PHP documentation, so this can be referred to while developing.

17.13 Caching AJAX requests

AJAX requests to a server should be cached. Remember, an AJAX request is simply an asynchronous call to a file using JavaScript - meaning the file is access without a page load required, and these should be cached so the browser can retrieve the file from storage instead of having to re-download the content again. If you're using a JavaScript library like jQuery, this happens by default when using the ajax method, but when caching isn't required you can turn it off. Turning off caching will append a timestamp to the end of the file being requested within the query string. It may look something like:

articles.xml?_=1359464930724

This means the file path will essentially change with every request and therefore the browser will almost always re-download the content. This is only required for when you're requesting files that contain real time information and will change frequently.

[3]http://eaccelerator.net/

18 Security

Security is often looked past when writing any code, particularly when communicating with a database like MySQL. The reason for this is that it's extremely easy to ignore escaping data ready to be used in queries, blindly outputting data from a database table and quickly setting sessions or cookies without researching the security considerations around these. We'll start by looking at the biggest and most exploited, SQL injection. We'll then move on to look at some less common problems, but equally as important.

18.1 Why no website is secure

When you're building anything, you should be aware that it will never be 100% secure. This is due to the fact security vulnerabilities can be exploited several different ways and new vulnerabilities are always being discovered. If you've secure against something one way, there is a likely chance that it will be able to be exploited via another method. There is a concept called defence in depth, which helps but will still not render your code 100% secure. Let's say you've secured your website as much as you possibly can from SQL injection and have taken precaution to connect to a MySQL server using secure login credentials. This is absolutely fine, however what happens when a security vulnerability becomes known for direct access to the server? When someone connects to your server and gains access to your data, how could you have known this was going to happen? The answer is you simply couldn't, however defence in depth would mean that you would have encrypted any sensitive information and therefore would be useless to an attacker, e.g. passwords and credit card information.

Don't worry though, it's not something to completely panic about, despite the fact it sounds depressing. If you take care to secure your website as much as you possibly can, it will be difficult for the majority of attackers to breach your security.

18.2 Security first

This is so important to mention, because many people consider security as an afterthought. It's easy to do this, so if you're the kind of person that thinks like this, don't worry. It's natural to develop something and promise to yourself that you'll fix it up later. As a developer you absolutely need to get into the habit of thinking about security as you develop. This isn't just a case of using mysqli_real_-escape_string in the odd place (a MySQLi function that strips data of potentially harmful characters that may assist SQL injection), it's about looking at your website from every angle. What if someone changes this GET value? Can this form be spammed? How can I provide an additional layer of protected when a user is performing a critical action? Unfortunately for most people, the important of security is something that comes after an attack has taken place or they've written some code that's been exploited. It's difficult, because without being an expert in PHP security it's sometimes hard to be fully aware of exactly how secure your website is. There are a host of resources and books that cover security, but remember it should be tailored for your code and not blindly applied.

18.3 What else?

You may have heard of SQL Injection and XSS attacks, but what else do we consider security? A popular attack that often brings websites down is something called a DOS (Denial of Service) or a DDOS (Distributed Denial of Service) attack. This consists of large amounts of traffic being sent to a server until it eventually can't cope. Remember that servers have limited hardware and software capacity and in essence operate like a home computer, so can suffer from these kinds of attacks.

Malware is also a concern. Are you letting people upload files to your server? Are you protecting against the kind of file that can be uploaded properly? If not, it's possible for worms, viruses, etc. to be uploaded or hostile code to be executed on your server. A single PHP file could wipe your entire public directory, leaving you with nothing.

18.4 Back up

If you're scared already, I apologise! One way you can ease the pain of potential data loss is to back up, and back up regularly. This obviously isn't an excuse to not implement good security. Backing up will mean that things can be restored in the event of a security attack and allows you to focus on fixing what went wrong and where. Remember that a database compromise can potentially reveal sensitive user data, so backup is simply a data loss precaution, not just something to fall back on. There are several ways you can back up your file and database data, and this can be automated and data can even be sent to a remote location. If you're running something like Parallels or cPanel there are automated backup processes that allow you to send data to a remote FTP location. Useful!

Backing up database data is slightly more difficult as you'll almost always need to write some code to do this for you, unless you can find an application that will connect and store this data for you.

It's probably recommended to back up once a day but of course this depends entirely on your website and what you need to back up, and how often data is written to your database or file system. It also depends on how often code is released to the live server as part of development, as you'll want to back this up too.

18.5 SQL Injection

Simply put, SQL injection is when an attacker crafts user input in such a way that it causes an SQL string to perform an unexpected and undesired query on a database. These can be deadly, with the ability to delete entire databases in some circumstances.

All data (whether specified by a user or not) that is used to form part of an SQL string, should be escaped. By escaped, we mean alter this data to ensure that it isn't able to perform an SQL injection. There are many different ways to do this, but we'll start with the most used example to explain how SQL inject could work, and then look at how we can protect against it. Consider the following query:

```
SELECT `user_id`, COUNT(`user_id`)
FROM `users`
WHERE `username` = '{$username}'
AND `password` = '{$password}'
```

This will return a count (1 or 0) as long as the username is unique (which we'll assume it will be), telling us whether a specified username and password match, meaning a successful login. We'll assume a plain text password for now, so we don't get into encryption discussion. When the user provides a username and password, the query may then look like the following:

```
SELECT `user_id`, COUNT(`user_id`)
FROM `users`
WHERE `username` = 'ashley'
AND `password` = 'lawnmower'
```

Albeit being a weak password, this looks fine, and if the user 'ashley' did exist with the password 'lawnmower', this query would return 1. We may then go on to set a session containing the user's ID as we've also specified this in the query. The danger here is that if this query was modified by SQL injection, it could return 1 regardless of whether this username and password combination had been found or not. Now, if we were to enter:

- The username as ' OR ''= '
- The password as ' OR '' = '' AND `user_id` = '25

Then the SQL string would look like this:

```
SELECT `user_id`, COUNT(`user_id`)
FROM `users`
WHERE `username` = '' OR ''= ''
AND `password` = '' OR '' = '' AND `user_id` = '25'
```

Let's evaluate this by hand. The query is checking that a username is equal to an empty string or an empty string is equal to an empty string (evaluates to true). It then requires that a password is equal to an empty string, or an empty string is equal to an empty string (which also evaluates to true). We then have an additional AND clause specifying the user_id must be equal to 1. This is something that will need to be guessed, but in reality isn't that hard to. In effect we have two statements that have evaluated to true, and therefore the entire WHERE clause is true. We would now be logged in as the user with user_id 25.

This is a very basic example of SQL injection through user authentication. However, this isn't limited to where a user specifically defines data. Watch out for values sent as query strings in URLs, posted as hidden form data or within cookie or session data. For example, you may be sending a variable via a URL query string to show a certain amount of posts per page. It's likely that this would be placed directly into the query using the LIMIT clause and therefore can be manipulated, directly from the URL.

If you have hidden fields within a form then these need to be taken into consideration also, as it's easy for an attacker to view and modify these. With cookie data, you may set cookies for particular settings. This could be a cookie to store a user's particular setting, such as which page they were last looking at in an index of articles. Assuming you'd use this in a query with the LIMIT clause, this would need to be escaped also. These are a few examples of data that can be manipulated by users and some of the ways these values may be used in an attack, but pay attention to your own website and the data you're passing to queries.

We'll now look at how to escape data. We already know that single quotes can interfere with the flow of a query, so how do we escape these? The answer is to use the built in functionality from your database extension or think about the data you're inserting and cast it to a particular type. There is another important consideration relating to PHP settings which we'll also discuss - magic quotes.

However you're connecting to your database, you'll find a method to escape data. With the old mysql_ function extension, the function required is:

```
mysql_real_escape_string($string);
```

Previously there was mysql_escape_string, which should no longer be used. Take a look in the PHP manual to find out why. For MySQLi, which is recommended instead of the old MySQL extension, you could do this one of two ways, depending on your application style of development. This is the procedural and object oriented way.

```
mysqli_real_escape_string($string); // procedural
$db->real_escape_string($string); // oop, where db is the connection to your MySQ\
L database
```

For the majority of cases, this is enough to escape user input so you're protected against SQL injection. The functions we've looked at above, however, can be used unnecessarily when we're using certain data types. Let's say you're inserting an integer into a query. You wouldn't need to use an escaping function because you know the data type and can cast this variable to a specific type:

```
$user_id = $_GET['user_id'];
$sql = "SELECT * FROM `users` WHERE `users`.`user_id` = {$user_id}";
```

The above simply concatenates the query and the supplied user ID from the global GET array. This user ID could be submitted as part of a form or supplied directly from a query string in a URL. Note, we're not checking whether this user exists, or performing any other validation, simply to water down this example. Looking at our earlier examples, anything could be passed to this page and could form a dangerous query. If we know that the user_id field is of integer type, then we know it has to be supplied as an integer. So, to escape this data we can simply do:

```
$user_id = (int)$_GET['user_id'];
```

This simple addition will ensure that this variable is cast to an integer, removing any floating points or characters. For example, if the user ID was supplied as '1a', our casting would leave this as 1. If it was supplied as 'a1', it would be 0. If we supplied it as 1.8, it would simply round down (note, this doesn't round up) and leave 1. We've now cast a variable to a data type that we were expecting. Casting doesn't have to just be for integer values. You can also cast to boolean, float, string, array, object, and can also use unset to cast to NULL. Casting in PHP is known as type juggling. As PHP is loosely typed, it doesn't require casting when comparing or manipulating variables, such as concatenation or mathematical operations. Strictly typed languages like Java do require casting.

18.6 Magic Quotes

We'll now look at magic quotes, what they are and what they do, and how to control this functionality based on your server settings. It's important to establish that magic quotes has been deprecated since PHP 5.3 and actually removed in PHP 5.4 onwards. This was for very good reason, considering that magic quotes for many developers is something that has been ignored or not properly understood.

Basically, magic quotes is a feature of PHP that automatically escaped data passed to a PHP script. The plan here was to escape data globally so that anything passed to a query or anywhere that may pose a security risk was protected. Take the following line of code as an example:

```
$sentence = $_POST['sentence'];
```

If magic quotes were turned on, and we passed "It's a really sunny day, let's all go outside.", we would end up with "It\'s a really sunny day, let\'s all go outside.".

The danger with magic quotes is that it can be turned on and off within the php.ini file, the central file for a server's PHP installation. This means that assuming that magic quotes has escaped all data on one server may mean that transferring to another server will leave the code open to SQL injection. Because some users may not be able to control their php.ini file through restrictions imposed by their hosting company, this would either mean that the entire code would have to be checked and data escaped, or a user would simply ignore it, which would avoid the point of magic quotes. Other reasons for magic quotes not being used is due to the fact that all data passed (e.g. from $_GET, $_POST, $_SESSION) is escaped, and this data may not need to be escaped, meaning that processing power is used unnecessarily. It makes sense to only escape the data that is not to be trusted. This also means that data is escaped and passed back in this form. For this reason, if a user were to type a comment into a comment box on your website and hit return to see a preview of their data, they would see the escaped content. For this reason, you'd have to use the PHP stripslashes function to remove the slashes before displaying this data to the user. This is a performance problem as you're using another function without the need to. Overall, it made sense to remove this functionality from PHP, and rely on developers to escape the data by hand.

To check is magic quotes is turned on, you can simply look within your php.ini file for the following lines:

```
magic_quotes_gpc = Off
magic_quotes_runtime = Off
magic_quotes_sybase = Off
```

You can also turn off magic quotes by placing the following line within your .htaccess file if you don't have access to your php.ini file, or can't make changes to it.

```
php_flag magic_quotes_gpc Off
```

You can now develop as usual, and escape data when and where it needs to be. When you transfer your project to a live server, or switch server, you can rest in the knowledge that your website will be protected against SQL injection, regardless of the server's PHP settings.

18.7 Session and cookie security

We'll discuss session and cookie security together, and particularly in relation to user authentication. Typically, data identifying a user will be stored in a session, and could be initiated by a cookie if the user has chosen to be remembered (obviously, you'd never store data in a cookie like a user ID, as the value could then be easily manipulated within the browser). Using the most basic setup to store values in a session simply isn't good enough, and we need to employ several other steps to ensure we make our website more secure. By all means, this isn't going to be a detailed account into session and cookie security, but it should give you the ground to start to build on the security practices here and go and research further ways to secure your website.

When you store a value in a session or a cookie, this can be viewed by the user easily within a browser. At the time of writing, Firefox makes it particularly easy to view and modify session and cookie data. Try setting a cookie and a session on a page and open up the Firefox Firebug extension to take a look. You'll find any sessions or cookies from the page you're accessing under the cookies tab. Sessions are technically cookies, but are stored with a value that can be identified by the server, and then subsequently this value is looked up. If you've already examined with Firebug, you'll see this hash. This means that the value can't be read in plain text or manipulated meaningfully as the long string you see is only used to look up a server stored value. Cookies, however, store values in plain text and can be manipulated to potentially change the way code behaves. There are a also variety of ways that sessions can be manipulated and hijacked, so that an attacker can pose as someone else and do everything they'd be able to do if they'd logged in with their username and password.

Where are sessions stored?

Sessions are stored on the file system, where they hold the data you assign to them within your code. For example, if you were to do the following:

```
session_start();
$_SESSION['secret'] = 'tabby';
```

This creates a file with the following name within your PHP tmp directory. You can find out where this directory exists by either browsing around your PHP installation folder or taking a look in your phpinfo output.

sess_0613842845b85ca357ebe1d1eb0fb514

The last part of this filename is the session value within the browser. If you use tools like Chrome Developer Tools or Firebug to inspect these you'll see this hash. This file contains the value:

secret|s:5:"tabby";

If this file was somehow compromised, it could be used to set a valid session for another user and therefore take over an account.

So how can we make session storage more secure? A popular method is to store sessions within a database where access should be harder. PHP allows you to extend the functionality of its original session handler and from here you can store and retrieve sessions from a database. This is way beyond the scope of this book as it would involve listing a lot of code that would be non-specific to your website, however go ahead and look it up.

18.8 CSRF (Cross Site Request Forgery)

CSRF occurs when requests can be forged by providing a link which an unsuspecting user clicks on. This subsequently performs an action on your website that may have not been originally desired by the user. Let's take an example:

```
<a href="/deleteaccount">Delete your account</a>
```

The following link may be presented to a logged in user to remove their account from your website. In this instance, the user may simply be presented with a dialogue, allowing them to confirm their decision to leave your website. This may look like the following:

```
<p>Are you sure you want to delete your account?</p>
<a href="/deleteaccount?confirm=1">Yes</a>
<a href="/">No</a>
```

Clicking 'Yes' will redirect the user to /deleteaccount?confirm=1 and will process their account deletion.

Now, if a user is still logged into your website and they're browsing another website, they could be presented with the following link, either posted or hosted by someone:

```
<a href="http://www.phpacademy.org/deleteaccount?confirm=1">Click here for cute c\
ats!</a>
```

And your unsuspecting user has now had their account deleted on your website because you've not protected against CSRF. So, how do we protect against something like this? It's actually rather simple and can be implemented quickly along with the rest of your link/form:

The first thing to do is set a session value, with a random hard to guess string:

```
// combine the result of the microtime function and the user's ID and md5 hash it.
$_SESSION['csrf_token'] = md5(microtime(true) + $user['user_id']);
```

The fact we're using an md5 hash here doesn't matter. What's important is that this value can't be easily replicated. Using a simple string of numbers like '12345' as a CSRF token would be silly.

Now, output this token wherever you require protection:

```
<p>Are you sure you want to delete your account?</p>
<a href="/deleteaccount?confirm=1&token=<?php echo $_SESSION['csrf_token']; ?>">Y\
es</a>
<a href="/">No</a>
```

Or a form:

```
<form action="deleteaccount.php" method="get">
        <p>Are you sure you want to delete your account?</p>
        <input type="submit" value="Yes, delete my account">
        <input type="hidden" name="confirm" value="1">
        <input type="hidden" name="" value="<?php echo $_SESSION['csrf_token']; ?>">
</form>
```

All that's left to do is match up the session value with the value retrieved from the result of clicking the link or submitting the form. In your confirmation page you may have the following code:

```
if ($_GET['c'] === $_SESSION['csrf_token']) {
        // process user account deletion
} else {
        header('Location: /');
}
```

If the value doesn't match, we do something with the request. In this case, redirect. In situations like these, try to avoid outputting something like "Ha, got you!" or similar, as it gives an attacker more clues as to how your code is written (or unsuspecting users more confusion).

18.9 Include/require security

When talking about includes, I refer to using include, require, include_once and require_once, which all do more or less the same thing, just with minor variations. Consider an example when determining to show a user panel or admin panel to a logged in user. You may have a check to determine the access level of the user. For example:

```
if ($user->is_admin()) {
        require '/includes/admin.php';
} else {
        require '/includes/user.php';
}
```

This is fairly straightforward and to an untrained eye would seem insecure. The is_admin method deals with the check and will only include admin.php if true has been returned. We'll assume that is_admin is fully functioning and working as expected. Now, if an attacker were to enter http://localhost/includes/admin.php, they would have immediate access to any admin functionality as the functionality within the include is not protected. Now consider that the code above remained the same, but a change was made to the admin.php file to be included:

```
// contents of the admin.php include
if ($user->is_admin()) {
        /* admin page stuff */
}
```

When an attacker loaded up the URL above, they would be presented with a blank page. It's a good idea at this point not to include an else statement with a message like "Sorry, you're not allowed here!" as this is giving an attacker clues as to how your code works.

An alternative method of securing includes is to place them outside a publicly accessible location. This also means we don't have to reuse the is_admin method when including the file and within the include itself. Placing admin.php outside the public folder may result in code like so:

```
if ($user->is_admin()) {
        require '/var/secure/admin.php';
        // or, if you're on a windows server
        // require 'C:/secure/admin.php';
} else {
        require '/includes/user.php';
}
```

Bear in mind that the above example assumes there are no other security exploits in your application that may allow access to the admin.php file, such as the ability for an attacker to traverse through files.

18.10 File traversal

When creating dynamic websites that allow values to be passed either from a session, GET or POST variables and placed into a function that accessing files, you're automatically decreasing the level of security in your website. Consider an example where a user can upload a text document and then view it immediately. You may have functionality similar to the following:

```
header('Content-type: text/plain');
$file = $_GET['file']; // assume a value of 'notes.txt'
$dir  = 'uploads/';

if (file_exists($dir . $file)) {
        echo file_get_contents($dir . $file);
        // will include 'uploads/notes.txt'
}
```

As it stands, the above code is extremely insecure. We're assuming the contents of the GET value is 'notes.txt', but what if it were '../.htpasswd' for example? Yes, you guessed it, we've just traversed back a directory by including 'uploads/../.htpasswd' and output the contents of the .htpasswd file. You may also assume that if only uploading text files then you could change the code for added security:

```
header('Content-type: text/plain');
$file = $_GET['file']; // assume 'notes'
$dir  = 'uploads/';
$ext  = '.txt';

if (file_exists($dir . $file . $ext)) {
        echo file_get_contents($dir . $file . $ext);
        // will include 'uploads/notes.txt'
}
```

So now if we provide the value of '../.htpasswd' into the URL, we'll be including 'uploads/../.htpasswd.txt' which doesn't exist and therefore will not be output. However, on some servers a null byte can be used to ignore anything after the point where the null byte is inserted. A nullbyte is basically a control character representation of the value 0, and is specified as %00 in a URL.

This means that when we provide the value '../.htpasswd%00', the .txt part of the include is effectively ignored and an attacker has successfully gained access to the .htpasswd file contents again. If this all sounds daunting then don't let it. It's a simply case of sanitizing all user input and not trusting anything else other than what you expect. In this example we could change the code to:

```
$file = str_replace(chr(0), '', $_GET['file']);
```

This essentially strips out a null byte character. If this confuses you, run the following code:

```
var_dump(chr(0));
```

And you should see the following output:

string '�' (length=1)

Inspect the source code of the page you've output the above snippet on, and you'll find the following character hidden amongst the PHP output formatting which was previously the question mark character:

```
&#0;
```

And that's a null byte as an HTML entity!

18.11 Important considerations

When you're building any software, security should be built into your application from the start. However secure you think something you've written is, there's a chance that vulnerabilities may still exist. If in doubt, initially you should always assume that something along the line of your security process will go wrong. Let's look at an example in which we read in GET data and output a file based on this given value.

```php
$file = trim($_GET['file']);
$file_allowed = array('about', 'contact');

if (in_array($file, $file_allowed)) {
        $path = 'includes/' . $file . '.php';

        if (realpath($path)) {
                include $path;
        }
}
```

Here, we're taking several precautions.

1. Trimming the input using the trim function to strip whitespace from the left and right.
2. Checking our value is one that is expected by using the in_array function to only allow specified values.
3. Using the realpath function to see if a file actually exists before including it.

Without security precautions, the code would read as follows:

```php
$file = $_GET['file'];
include 'includes/' . $file . '.php';
```

18.12 Error reporting

You might be wondering what error reporting has to do with security. Take the following example. You have an /admin directory (not a great idea anyway) that contains an index.php file. This file contains a form to enter a username and password, and if correct, forwards the user on to administrate the website. Let's pretend we've recently changed our MySQL user password and haven't updated our code to reflect this. If the connection fails, you'd expect to see something like:

"Access denied for user: 'alex' to database 'php'"

Oops, you've now revealed the name of the username connecting to the server and the name of your database. This may not seem like such a big deal, but it's simply more that an attacker can add to their inventory against your website.

If somewhere along the line a query fails, you may see an error output like:

"You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'SELECT `admin_id` FROM `admin` WHERE `username =` ' at line 1"

This is of course an example, but entire SQL queries can be displayed revealing more about your database structure, depending on where the error appears in the query.

So how do we control error reporting? There are a variety of error reporting levels that we can set in our PHP configuration, and we can also turn it off altogether. This is recommended for production environments. You'll be interested in the following lines in your php.ini file:

```
display_errors = 1;
error_reporting = E_ALL;
```

In the above configuration, we're enabling error reporting and using E_ALL to dictate what kind of errors should be displayed. Simply changing the `1` to a `0` on the display_errors property will turn it off. You don't need to make any amends with the error_reporting property. You can also change the error reporting level, and you'll find some great documentation and examples within your php.ini file that'll tell you more about this.

18.13 error_log

The error log is by default stored within the directory of the file the error originated from and will be stored with the file name error_log (no extension). This is great for development, as you can pinpoint where something is going wrong, open up a log and inspect it for the errors that are preventing you from continuing. These logs however, are available for anyone to access unless you either disable their functionality or you specify that they be saved in a different location. For a development environment that can't be accessed by the public, this is fine to leave, however within a production environment you can take action and change the default behaviour of this. Head over to your php.ini file and look for the following lines:

```
log_errors = On
error_log = /path/to/error.log
```

It's preferable to store error logs outside of your public directory or if not, store within a folder where you can define no access using an .htaccess file. For example, an .htaccess file to deny directory to all files within a directory would look like:

```
Deny From All
```

Simple, now you're logging errors to one location that can't be accessed within the browser, but you can still view. Of course, you could always just turn error logging off!

18.14 Passwords

When you use hosting, you'll obviously use passwords to access the services being hosted. This includes any control panel you're using, your billing area, MySQL and more.

Passwords are often overlooked in security, and it's really important that any passwords that even remotely relate to your server are long and strong. You may not think that your billing area needs a particularly strong password. What's the most an attacker will do here? Pay a bill? Unlikely, but they could gain access to information relating to your account. What happens when they contact customer services to have your account removed?

18.15 You online

This brings us to my next point nicely, which is what you write and share about yourself publically online. Basically, the more information available about you online, the more likely it is that someone can impersonate you. This might involve someone breaking into one of your accounts by using secret information (e.g. "What was the name of your first pet?") to reset your password, which could easily be obtained from a social networking site you're on. Often overlooked, this is an important thing to watch out for.

1. Keep as much personal information offline as you can.
2. When entering answers to secret questions, use bogus information to deter people using this to reset your password.

18.16 What next?

Security is something that will always be an issue. Whatever the environment, you should always consider security and build it into your application. Security should never be an afterthought. The key is to never trust input, ensure you have validation present on all input even when coming from trusted sources and to sanitise before storing data.

We've not actually covered every security problem you may run into when developing, but we've covered the most common. Now you've seen the basics, you can at least apply these to your website and start to tailor more advanced and more extensive security into what you're doing.

18.17 Recommended reading

I highly recommend visiting the PHP Security Consortium[1] which is an ever expanding resource for security concerns around PHP. The founder has also written a book called Essential PHP Security[2] which is a great book if you're just starting to learn about PHP security or want to top up or clarify any knowledge you currently have.

[1] http://phpsec.org/

[2] http://phpsecurity.org/

19 Ideas

It's worth talking about ideas and intentions when building applications. The best way to learn for most people is to build and learn along the way, so it's important to be able to take an idea further. But what if you're struggling for an idea? You may want to start working on what will eventually become a product that forms part or all of your income, or just a website that you own and maintain. You may even want to start working on an open source project that you lead, while getting a community involved in active development. Either way, you first need to come up with an idea.

19.1 Coming up with an idea

Thinking about what'll make you the most money, or what will grow to be the biggest website will not get you far. Passion and commitment is key to building any website, and there are so many websites that haven't yet been built that if implemented properly with a dedicated team, could become a massive success. Most websites will have been built to solve a problem that either isn't solved elsewhere, or is solved but isn't implemented as well as it could have been. If you're wondering what to build next, you should consider building something that offers value to those who will use it.

A brilliant product is something that doesn't necessarily go with trends and offer tonnes of features to users. When building something, try and start with only what's absolutely necessary and see if it'll do. Often when building products people assume that the more features a product has, the better it is. This isn't true for the majority of applications. How many times have you struggled to get something done in Microsoft Word while finding something like Google Docs simple and refreshing?

19.2 Monetizing

Don't think about how you'll monetize your project. Of course, you may need to do so to be able to create a product that will generate revenue, if this is your goal. However, if you create value within a useful product, people will want to pay and you can decide on this later. This goes the same for advertising. If you're already planning spaces for advertisements you may not be thinking how best your user experience could be. You may be able to safely set aside a small amount of space comfortably without compromising your user experience, but in the initial stages don't worry about this space too much. It goes without saying that adverts shouldn't be the focus of your content. This concept could be written about within a book of its own, but generally you need your adverts to blend in and work with your content, not be so prominent that they're annoying.

19.3 Get feedback

Feedback is extremely valuable. If you're always seeing your website from your perspective, you'll never truly find what people want from it. Instead of asking just anyone, try to introduce your website or idea to people that will most likely use it. This way, you can gauge interest or usefulness, perhaps before you've even started the backend. You may start with a design and some very basic functionality, which will often be enough for a potential user to give you good feedback. Visuals are important, as these will engage your users and allow them to visualise themselves using the website even if it isn't yet funny functional. This will put them in the mind-set of actually using it, and they'll probably be able to give you more useful information.

19.4 Start small

Start small and you'll be able to build on something slowly and carefully. Adding, removing and changing features as you receive feedback from your users means you can tailor what you're building along the way to the people who will eventually be paying for your product or using your website on a regular basis. If you want to implement an idea but know it'll take you several days to design and code, start with the design and show users how it'll work. If they don't like it, you can make a better judgement on whether to scrap it or not. This can save you a lot of time and effort, and can help you tailor your website to your audience and their needs.

19.5 Recommended reading

I highly recommend reading REWORK[1] by 37signals, creators of popular project management software Basecamp amongst other software. There is a vast amount of extremely valuable advice in this book that I wouldn't wish to try and replicate here.

I also highly recommend The Lean Startup[2] by Eric Reis. This is another extremely valuable book with a concept that many businesses today have been built around. Reis has coined terms within the startup community that are now used within everyday business life.

[1]http://37signals.com/rework
[2]http://theleanstartup.com/

20 Examples

You've arrived at the fun chapter of the book where we'll be discussing the thought process that may go into development. If you haven't already started to think about how to approach the design, structure and functionality of just about any website yet, then hopefully this chapter will clarify this for you.

20.1 Building a forum

This is probably one of the most sought after applications to build, and can actually be relatively easy depending on how advanced you want or need to get. Most people will associate a forum with a complicated social platform packed full of features, so they fail to see what a forum actually consists of - content. Content in this form is extremely easy to display and store, and then the features of a forum can be built around this data, allowing users to interact either with forum topics, posts or other users. In this example, we'll be looking at the following parts of a forum:

1. Ability for users to create an account and sign in
2. Listing categories
3. Listing topics
4. List posts within topics
5. Ability to post a new topic and a discussion about how we may implement replies to topics

The above is what makes a forum. Obviously, we'll need to take user permissions into account, but this chapter isn't technical. It's simply to assist the thought process of building something like this. The steps you take to build forum software are very similar to 90% of websites. As we discussed earlier in the book, most websites are simply made up of the ability to display and store data.

User authentication

Your user authentication system can be built quickly and easily, but needs to be done properly and securely. It's at this point that you may consider using a framework or at least taking advice from an expert in this area. Remember, simply using a framework doesn't guarantee that what you're building will be entirely secure. If you've already built something, address the code and look for any security issues that may cause a problem. Escaping data for SQL injection often won't be good enough for general security, so you'll need to consider:

- Session security

- Cookie security (if you're implementing this functionality to remember users)
- Spam protection, so multiple accounts can't be registered repeatedly
- Login attempt protection so your login system isn't open to brute force attacks on user accounts
- Sanitizing user input (yes, even usernames, email addresses and other user data that may be output) to help protect against XSS attacks
- Strong password encryption
- Ensuring that passwords are salted so if hashed passwords are obtained at any point, it's a lot harder to crack passwords using rainbow tables
- A host of other security holes that may be present in your website

Category listing

Forum software usually lists categories that can be selected and will show topics within each category. When a user makes a post, they will choose a category to post to. As with most databases, we'll have a field in our topics table that references which category a topic belongs to, and therefore it makes sense to start with these, as they'll be shown on the homepage. You'd start with building your table, which would probably contain the following fields:

- cat_id
- name
- description

That's as simple as it needs to be, but can include a variety of other fields which you can always add to later. Looking at the above, this is all that's required to start populating the table and to output these in a list. We're simply not overcomplicating things. Now, however you're querying your database, you'd select all three fields and you're done. Your query may look something like this:

```
SELECT `cat_id`, `name`, `description`
FROM `categories`
```

If you need to order them specifically and think this may change in the future, add a marker to sort by. For example, you could add 'order' as a field and then do:

```
SELECT `cat_id`, `name`, `description`
FROM `categories`
ORDER BY `order`
DESC
```

So, if you only needed the 'PHP' category at the top, you could just add a 1 integer here. If you needed to continue to sort, you could have 'PHP' as 3, 'MySQL' as 2 and 'CSS' as 1.

Now you've queried, you'd naturally loop through the results, again, using the method you've chosen to do this. Let's say you were outputting within an unordered list, it may end up looking something like the following:

```
<ul>
    <li><a href="category.php?id=1">PHP</a></li>
    <li><a href="category.php?id=2">MySQL</a></li>
    <li><a href="category.php?id=3">CSS</a></li>
</ul>
```

You'd arrive at this output of markup by looping through and outputting the cat_id where the id= part of the URL is and the category name within the anchor element. The description has been omitted for clarity, however this could just as easily be added in. Bear in mind you may not have built category.php yet, and this doesn't matter. You should now have a vague idea of how this may work, where in category.php we need to pick up this ID that has been passed through. Remember earlier we discussed that for every topic created, it would contain a reference to which category it had been created within? Well, the query on category.php can now pick up all topics only assigned to that category.

Listing topics within a category

We have no topics and nowhere to even store topics, so we'll need to create the topics table first. As with most tables, let's keep this as simple as it needs to be. It's far too easy at this point to start wandering off and thinking about what other fields you need, so let's create only the field we need to make this work. We can add the rest later. The fields we need are:

- topic_id
- cat_id
- user_id
- title
- timestamp

We don't necessarily need the timestamp value stored, but it is commonplace to show when a topic was created and this also helps for ordering topics within a category. We have two fields that will reference other tables, and we'll use these later to join data. So, in our query we could join the users and categories table to return the category name and the username (amongst any other data required) of the user who posted the topic. We'll not get into this just yet, but let's instead take a quick look at how we may construct a query to output a list of topics, and we'll then look at the resulting sample output HTML. When we hit category.php we need to take in the ID passed to the URL, which would look something like:

```php
$cat_id = (int)$_GET['id'];
```

The cat_id variable hasn't been checked for existence and likewise we haven't checked that this category actually exists, so you'd need to ensure this is done. The query to follow would look something like the below:

```sql
SELECT `topics`.`topic_id`, `topics`.`title`, `topics`.`timestamp`, `users`.`user\
name`, `categories`.`name`
FROM `topics`, `users`, `categories`
WHERE `topics`.`user_id` = `users`.`user_id`
AND `topics`.`cat_id` = `categories`.`cat_id`
AND `topics`.`cat_id` = $cat_id
ORDER BY `topics`.`timestamp`
DESC
```

Returning all topics within one category isn't advisable, so would need to be broken up with pagination to avoid loading thousands of topics at once. This could be done with the MySQL LIMIT clause, but we won't discuss this yet as there are a variety of different ways we can paginate.

Assuming the user has visited category.php?id= 3 (our CSS category), the resulting output would be something like:

```html
<div class="topic">
        <h2><a href="topic.php?id=1">Help with CSS</a></h2>
        <div class="author"><a href="users.php?id=5">Billy</a></div>
        <div class="category">in <a href="category.php?id=3">CSS</a></div>
</div>
```

We have everything we need from the query, most of which is coming from the topics table (cat_id, user_id, title) but we've also taken the user's username from the users table and the category name so we can display it in a friendly manner. You may have noticed we're not storing any post contents within the topics table, and this is simply because we're going to create another table to house topic posts, including the first post when a topic is created. This means doing a similar things as topics, but this time linking posts to the topic they belong to.

Posts within topics

As mentioned, posts are stored in another table with a reference to what topic they belong to. This helps keep everything separate and avoids duplication of data. For example, if you were storing post text along with the topics, you'd have to duplicate the topic_title for every post, which is obviously unnecessary, a waste of space, slow, and unmaintainable. Instead, we create a posts table with the following fields:

- post_id
- topic_id
- user_id
- text
- timestamp

This is all we need to identify where each post belongs, who posted it and the ability to order the list of posts by the timestamp. So, when we land on topic.php?id=1, we need to show the posts within the first topic. We'll also need to show metadata such as the topic title, who posted each post and perhaps some count results like how many posts there are within the topic. We can achieve all of this with a single query. There is absolutely no need to query for each part of what we need to show, which is a common problem amongst developers. Too many queries will slow down a page. We'll assume that once we've hit the page, the relevant sanitization will occur on the GET value passed via the URL. The query we write may look something like:

```
SELECT `posts`.`text`, COUNT(`post_id`) as `post_count`, `topics`.`title`, `topic\
s`.`cat_id`,
`categories`.`name`, `users`.`username`
FROM `posts`, `topics`, `categories`, `users`
WHERE `posts`.`topic_id` = $topic_id
AND `topics`.`cat_id` = `categories`.`cat_id`
ORDER BY `posts`.`timestamp`
DESC
```

So we've now returned all the data we need, and we could return more if we like (such as user post count). We'll skip an example for this, as the markup may get a little too long and we've already seen what an example of output may look like with the categories and topics listings which are similar.

You now have a functioning listing of forum categories, topics and posts within topics. It's not been difficult at all, and the only thing left to do to make this a technically functioning forum would be to create page that allows users to post a new topic, or reply to a topic. It makes a lot of sense to keep the functionality to post a new topic and the functionality to reply to a topic the same (keeping the same code), as there are only a few minor adjustments that would be made to complete either action. For example, the URL for posting a new topic may be:

post.php

And the URL to reply to a topic may be:

post.php?id=25?reply

Where 25 is the topic_id the user is replying to. This could be built as a small asset and included into the topic.php page, as when replying the $_GET['id'] variable would already be present and this could just be flagged with the reply GET variable in the query string we've seen above. We'll talk about the post.php page now, and how everything will be handled. We're not going to build this as if it were to be include within topic.php, so we'll look at it in isolation as if it were on a page of its own.

New topics and replying to topics

In this discussion, we'll look more closely at security and validation of data considering with any application that takes and displays user input you should be very careful to do so. I'll assume basic markup has been created to deal with entry, and you understand how forms work by submitting data either to the same page, or to another page. In this case, the action attribute for your form should be left blank in order to submit the data to the same page. The fields contained within your form should be a category dropdown, topic title, topic body and a submit button. It goes without mentioning that your user should be logged in before being able to access this page.

You already know what the topics and posts tables look like, but here's a quick reminder of their structure:

For the topics table:

- topic_id
- cat_id
- user_id
- title
- timestamp

For the posts table:

- post_id
- topic_id
- user_id
- text
- timestamp

We already know that the category listing will show all topics from the topics table, allow the user to click through to them and then view all posts associated with that topic, so really all we need to do is populate the tables, making sure we sanitize any data going into this table. We'll bring some validation into this discussion by briefly looking at how we should validate and sanitize.

Once you've submitted your form, you'll need to pick up the category the user has chosen, the topic title and the post body (which will become the first post of this topic). The dropdown (HTML select element) should be populated using whatever function or method you previously used to list categories on the homepage. This will loop through the returned categories and output them within a select element, with each option element being the category and containing a value matching the category ID. Ensure you're reusing whatever function or method does this, as there is no need to duplicate the query. Your output may look something like:

```
<select name="category">
        <option value="1">PHP</option>
        <option value="2">MySQL</option>
        <option value="3">CSS</option>
</select>
```

When the form is submitted, you'll then be able to access the cat_id using $_POST['cat_id']. Of course, you'll need to introduce a function or method that checks that this cat_id is valid and exists, perhaps something like:

```
public function exists($cat_id) {
        $query = mysqli_query("
        SELECT COUNT(`cat_id`)
        FROM `categories`
        WHERE `cat_id` = " . (int)$cat_id);

        return ((int)mysqli_result($query, 0) === 1) ? true : false;
}
```

I've used non object oriented MySQLi functions within this method to simplify the example. There's nothing strictly wrong with this anyway.

Now, this will ensure that your category exists, so we can start to sanitize other data and check for its validity. We won't go through any code here, but let's take a look at the topic title and the first post.

- Topic title - this should have a maximum value which should be defined in your database table also. The data type should already be a varchar, forcing you to specify the length of the field. Let's say this is 50, which is a reasonable amount. Checking the length of $_POST['title'] isn't strictly necessary as MySQL will populate the table and just truncate (cut off) any data that exceeds 50 characters. But this will generate an error, you may want to stop this and let your user know. Applying a maxlength attribute to your form field will stop any characters beyond 50 (e.g. maxlength="50"), but this can either be changed, or data can be sent via an HTTP header or another form so is not reliable. Use the strlen function to determine the length of the title and return an error if it's too long.
- Post - This is the same deal, although you should have the data type set to text within your database table which will allow a very large value. It's wise to limit this in PHP, even if you allow large amounts of data. You could, for example, limit this to 10,000 characters. You could use the maxlength attribute once again here and alert the user if too much data is entered.

For both of these fields, there is absolutely no reason to trust the input. We've already touched on security but let's review what we need to do with these fields to ensure it's clean.

Use an escaping function/method native to the database you're using, such as mysqli_real_escape_-string for MySQL. This escapes characters like single quotes that could cause an SQL injection to take place within the query. Use the htmlentities function to convert any characters that may be interpreted by the browser to their entity equivalent. For example, this would change

```
<script>
```

to

```
&lt;script&gt;
```

which looks a mess in plain text, but will be shown as

```
<script>
```

within a browser. This effectively stops any HTML being embedded into the topic title or topic post.

You could also use strip_tags to remove any tags, but don't rely on this alone. Your users may also need to post content with tags in without them being interpreted by the browser, so won't want this content to be stripped.

When you're done clearing up data, you need to insert it. The first thing to do is insert the topic data into the topic table. This creates a topic record. We then need to populate the posts table with the first post data, linking it back to the topic_id we've just created. But, how do we fetch the ID of the last inserted topic? That's fairly simple and for most systems is perfectly acceptable to use. MySQL provides the mysqli_insert_id function and will return the integer of the last inserted auto_increment field (topic_id in the case of the topics table).

So, with that in mind, this code could be used to populate the topic and post table:

```
$cat_id = $_POST['category'];
$title = $_POST['title'];
$post = $_POST['post'];

/* IMPORTANT: Validation and sanitization goes here */

if (empty($errors)) {
        mysqli_query("INSERT INTO `topics` (`cat_id`, `user_id`, `title`, `timestamp`) V\
ALUES ($cat_id, $user_id, $title, UNIX_TIMESTAMP())");

        $topic_id = mysqli_insert_id();

        mysqli_query("INSERT INTO `posts` (`topic_id`, `user_id`, `text`, `timestamp`) V\
ALUES ($topic_id, $user_id, $post, UNIX_TIMESTAMP())");
}
```

I’ve used the MySQLi function set once again to demonstrate this as simply as possible.

So what next? The category that posted this topic was posted to will now show a topic with the title specified. It was also to show the username of the person who posted it as when we joined the users table earlier, and clicking through will loop all posts within this section.

Now it’s your turn to build the reply functionality. Remember, this could be built directly into everything we’ve done, as the only thing to vary is that it’ll only insert a new record into the posts table with the topic being replied to and the user posting the reply. Just remember everything needs to be checked for validity and you need to implement security measures. The good thing about combining the reply functionality into the new topic form is that you’re not duplicating code and therefore can focus on the validation and sanitization of data in one area, meaning you’re less likely to forget something in either place.

We’ve not looked at code that can be directly copied, but more thought about the process and links between data and how everything should work. The point here is that if you’re unsure about building such functionality, you may be able to give it a go and even if you struggle, this will help you learn.

20.2 Pagination

Pagination is something that can apply to anything. Any data that needs to be broken up into sections or pages can take advantage of the methods we’ll be discussing here. As this is slightly more technical, we’ll look at more code and how we can query a database efficiently as not to load too much data in at once. It’s easy to load in all records from a table and then pagination, but instead it’s quicker to load in only the records you need for the page you need to display.

To start, let’s take a look at a query that returns all records from a table, guestbook.

```sql
SELECT `post_id`, `name`, `message`, `timestamp`
FROM `guestbook`
ORDER BY `timestamp`
DESC
```

If we had 100 records here, it’s not going to significantly slow down, but it’s inconvenient for a user to see all records output on a page, not to mention that the size of the HTML document will increase, slowing down the page load time for the user. Let’s take a look at slightly modified query using the LIMIT clause and performing an additional query to select the total number of all records. Note that we could have used sql_calc_rows_found, but some tests show that it can perform slower, particularly depending on how your database is set up.

So, we do:

```
SELECT `post_id`, `name`, `message`, `timestamp`, (SELECT (`post_id`) FROM `guest\
book`) as `count`
FROM `guestbook`
LIMIT 0, 10
```

We've performed a subquery to return the count of the table, despite the fact we've limited the query overall result set to 10. Using the LIMIT clause, we've started the record set at 0 (the first record) and specified we want to return 10 records. So, we return the first 10 results and our count field could contain the value of 100. We can use the overall count to work out how many pages there are based on how many we're showing per page. The calculation for this is:

```
$pages = ceil($row_count / $per_page);
```

All the PHP function ceil does is round up, so, in our case:

```
$pages = ceil(100 / 10); // returns 10
```

This is obvious though, so let's take a look at an example that would mean ceil had some use. Imagine we had 68 records and wanted to show 10 per page. Without ceil this would be:

```
$pages = 68 / 10;
```

This leaves us with 6.8 pages. Clearly this isn't right, hence why we round up, to accommodate for the records that trickle onto the 7th page.

```
$pages = ceil(68 / 10); // 7
```

Now that hopefully everything makes sense, it's easy to see how this can give great flexibility. You could give the user control over how many pages they wanted to view, perhaps using a GET value:

```
$per_page = $_GET['per_page'];
$per_page = ($per_page !== 0 && ($per_page <= 30 && $per_page < $row_count)) ? $p\
er_page : 10;
$pages = ceil($row_count / $per_page);
```

And, of course, you could check whether the user defined value is suitable for the amount of records you have, but this isn't necessary as the code above would render anything useless (e.g. text values or a 0 value) or anything too high or larger than the record count to a default value of 10.

To list your pages, you could use a simple for loop to do something like:

```
for($p = 1; $p <= $pages; $p++) {
        echo '<a href="?per_page=', $p, '">', $p, '</a>';
}
```

This would create a list of anchor elements (links), appending ?per_page= onto the end of each URL, which could then be picked up using the code we looked at above. You may also want to include a check to highlight the page the user is currently viewing, perhaps by adding a class to the anchor. All you need to do is compare the value of $p within the loop to the current GET value, if it is set you'll know what the current page is.

Hopefully you've gained a lot from the above logic, but now we need to put it all together. This will vary depending on how and where you're implementing pagination. For example, you could be implementing it into a generic function or a specific method that returns data for a guestbook, perhaps. Either way, the following examples should give you a general idea of how to implement pagination.

If the URL was guestbook.php?page=1:

```
public function entries($page, $per_page) {
        $start = ((int)$page == 1) ? 0 : (int)$page * $per_page;

        mysqli_query("SELECT `post_id`, `name`, `message`, `timestamp`, (SELECT (`post_i\
d`) FROM `guestbook`) as `count`
        FROM `guestbook`
        LIMIT " . $start . , " . (int)$per_page);

        // resturn result set
}
```

Note that the MySQLi function set is used for clarity, instead of the object oriented solution.

We've now introduced this into a method, with a new variable, start. This will start at 0 if the page variable is set to 1, otherwise it'll be increments of how many results are to be displayed per page. The result set we return will contain the count field we generated from our subquery earlier and from this the total number of pages can be calculated. This is only a taster of code but there isn't much more involved than this. Also bear in mind that pagination will always require data sent by the client to specify what page they want to view. This means that it's extremely important to be careful to sanitize data being using with queries. In this case, because we're only dealing with whole numbers, I've simply cast these values to an integer before using them within the query.

To put what we've looked at into practice and to try building your own pagination system, it's a good idea to start with a database full of data. Once you've got the hang of it, you can then build it into your existing application or work with it so it's more reusable and a bit tidier and can be quickly and easily implemented anywhere on your website or another project.

20.3 AJAX Content

Once you grasp the basics of using AJAX (Asynchronous JavaScript and XML) either alone or using a library like jQuery, it becomes extremely easy to use. There are a few points that should be raised, however, including efficiency and security as well as the best way to minimise the amount of code written. In this example, we'll look at how you may start to include AJAX requests into your website to allow content to be displayed and updated without page refreshes, as AJAX requests are simply asynchronous requests to files. The nature of this makes it extremely useful to process PHP within these requests, which then means data can be either returned and displayed to the user, a user can update something and receive a confirmation message or you can perform periodic checks for new data (e.g. a chat window pulling in data every 10 seconds). The best reason to incorporate AJAX requests is to aid UX (User Experience), so don't use it unnecessarily just for the sake of it. At points when a user is posting a forum topic, for example, it's customary to see a user redirected to the topic they just posted and therefore an AJAX request might be unnecessary here. You may, however, incorporate client-side JavaScript validation to enhance UX. However, remember to include server-side validation in addition to this.

Whatever you decide to use AJAX for, the following examples will help you grasp this. We'll be using jQuery to perform requests simply because jQuery has been built with support in mind, meaning that it's quick implementation will work across the majority of browsers. jQuery also provides a rich set of options for AJAX requests and allows us to very easily return data and process it, include traversing through XML nodes (although not part of the jQuery AJAX method). You can find short examples of basic usage of the jQuery AJAX method at the official jQuery documentation pages.

Simple processing of data

The example we'll look at is submitting an email address that will be stored in a mailing list table within the website's main database. This will involve an email address field and a submit button, and will include full server side validation. We're doing the validation server side because we'll return an error that will be shown to the user instantly anyway. With websites with high traffic, you may want to include client-side validation here to save requests to the server, but it'll complicate things in this example.

We'll start by identifying what resources we need. We need:

1. The jQuery library
2. Our own JavaScript file
3. Basic markup
4. A PHP file to process and send the validated, sanitized email address to our database.

The jQuery library and your own external JavaScript file should be loaded into your page just before your ending body tag, and will look something like:

```
<script src="ajax.googleapis.com/ajax/libs/jquery/1.8.3/jquery.min.js"></script>
<script src="js/global.js"></script>
```

You'll notice the jQuery version we're pulling in is from Google's CDN. This almost always makes the request to the file faster and means it's more than likely cached for users that visit your website. Our own JavaScript file is located within the js directory of our server and at this stage should be a blank file.

We then need some basic markup on our page in order to display a field and submit button to our user:

```
<div id="newsletter">
        <input type="text" class="email" placeholder="Enter your email address...">
        <a href="#" class="submit">Sign up</a>
        <div class="message"></div>
</div>
```

Here we have a wrapper container, a field to enter an email address with HTML5 placeholder text to guide the user, and an anchor. We're using an anchor simply because styling this is a lot easier than submit button, and we're not using a form element anyway. Because the data isn't directly submitted to a page through the use of the default HTML form functionality we can leave this out. We also have an element that serves as a container for any data returned from our request so we can display errors or confirmation messages.

Now for the jQuery code. At present, the following is incomplete and only returns what the PHP file we're addressing outputs. This allows us to determine whether the AJAX request has been successful or not because we'll see the data output from newsletter.php to us within a JavaScript alert dialog.

```
$('#newsletter .submit').on('click', function() {
        var email_entry = $(this).parent().find('.email').val();
        $.post('../ajax/newsletter.php', {email: email_entry}, function(data) {
                alert(data);
        });
});
```

You can see we're sending the data to newsletter.php within the ajax folder of our root directory. We're traversing back a directory because this file is located within the js directory. Above, when a user clicks the submit button we store the email address with a variable in the scope of the event handler. This is then passed to the post method of jQuery and sent, named 'email'. After this request is sent, we see a callback function with the argument 'data' which contains whatever has been returned by newsletter.php, which we place into a JavaScript alert. We're using the jQuery post method and not the AJAX method, as the AJAX method uses post or get anyway. If we needed to provide more options we could have written the above the following way, although at this stage unnecessary:

```
$.ajax({
        url: '../ajax/newsletter.php',
        type: 'post',
        'success': function(data) {
                alert(data);
        }
});
```

Now we know that we're successfully sending the request to newsletter.php, create the below file in the ajax directory. The code we then need to output at least something is:

```
<?php
if (isset($_POST['email'])) {
        $email = trim($_POST['email']);
        echo $email;
}
```

You can see we're accessing the email value sent using POST. This is simply because all that the jQuery post method is doing is sending the data as an HTTP request, so is no different from using a form, it just does it without needing to refresh the page. We're also just simply using echo to output the email address value, trimmed (useful for validating later). This will be carried over into the data argument we used within the callback function and will be output using a JavaScript alert dialog we implemented earlier.

Now we're done, we can validate, sanitize and use whatever method we're using to insert into the database. We won't show the code to check if the email exists, or the code that actually inserts the email address, because it doesn't directly relate to the point of demonstrating AJAX. However, here's what you may do to validate, including some comments where action points may be required:

```
<?php
if (isset($_POST['email'])) {
        $errors = array();
        $email  = trim($_POST['email']);

        if (empty($email)) {
                $errors[] = 'Email address required';
        } else {
                if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
                        $errors[] = 'That\'s not a valid email address';
                }

                if (strlen($email) > 100) {
                        $errors[] = 'Your email address is too long';
                }

                /* apply a check here for whether the email address exists in the database alre\
ady */
        }

        if (!empty($errors)) {
                echo '<ul><li>', explode('</li><li>', $errors), '</li></ul>';
        } else {
                /* insert email address into database, remembering to sanitize appropriately */
                echo 'Thanks, we\'ve stored your email';
        }
}
?>
```

On the client side, you would then remove the JavaScript alert and replace this with something like:

```
$('#newsletter .submit').on('click', function() {
        var this_ = $(this),
        email_entry = this_.parent().find('.email').val(),
        message = this_.parent().find('.message');

        $.post('../ajax/newsletter.php', {email: email_entry}, function(data) {
                message.text(data);
        });
})
```

We've done some things slightly differently here, and introduced two new variables. We're caching the $(this) selector so it can be used within the callback function of the AJAX request (as otherwise it'll refer to the object returned) and we've also selected the message element so we can inject the output from our PHP file into it.

21 Hosting your website

There are plenty of ways you can get your website online, but this comes with the struggle of different web server configurations, different PHP versions to your local development environment and unreliable companies either in hardware or support. In this chapter we'll review some considerations for getting your website online and maintaining it while online.

21.1 Choosing a host

There are literally thousands of hosting companies who offer some form of web hosting. Many of these companies are extremely cheap, simple due to the amount of competition. I've seen hosting companies offering packages for as little as $1 a month. The problem with cheaper web hosting companies is that they're more than likely stored within a larger dedicated server purchased through another company. Reseller hosting is also popular, where a server is purchased and specifically designed to allow easy reselling of hosting packages onto other people.

For small, non-critical test applications this is usually fine, but even then you may find problems. The main problems are often security, configuration and speed.

Security

When hosting a website on a shared environment, you're sharing disk space and a file system with other users. If you're not protecting your website it will be potentially vulnerable to access by other users on this file system. Also, it's the responsibility of the server owner to update, patch and monitor the server and this often doesn't happen. If you're lucky, the software on the server where your website is being hosted will be updated by the company it's been sold by, however this may not the case.

Session security is weaker on a shared hosting environment due to the way that sessions are stored on the file system. When you create a session containing a user_id, for example, you're actually storing this on the file system to be looked up by the default PHP functionality. There is a way to increase the security here, by creating your own session storage functionality. Unfortunately, this means more work and is one of the prices paid for hosting your website on a shared environment. Of course, you could implement this functionality even on a dedicated server if you wanted to increase security further.

Configuration

The configuration of a server sometimes canâ€™t be modified. In some cases, you won't even be able to define configuration rules in an .htaccess file. Not only is this extremely frustrating, it also makes it impossible to increase your knowledge, run code that requires such settings and means you have no control over the server configuration if you need to change it for an important reason.

Speed

As we've already learned, you're sharing hardware with other users and therefore an increased load within one environment is more than likely to have an impact on yours. Do you really want inconsistent results when you're testing your website? If something is running slowly, it could be harder to pinpoint where the problem lies.

So, if problems exist with these types of hosts, what can you do? The loose rule is to choose a host with a good reputation, and more importantly good support. To find a good host, don't look for lists of good hosts online by rating. These have often be added due to promotional reasons or link backs between websites. It's best to gauge an idea of how good a hosting company is by recommendation from someone who has used and had good experience with a particular host. Try also looking in forums to see the general scoop on a particular company. If all else fails, load up their website and head for their live chat and ask them all the questions you need to. If they're honest they'll give you all the information you need.

When choosing a budget package, a VPS (Virtual Private Server) gives the advantage of a dedicated server in a shared environment and allows you to change configuration. These are more expensive than shared hosting by are more than likely going to serve you better now and into the future. VPS packages when installed with something like cPanel, allow you to easily create accounts and therefore can be used to host multiple websites. This is exactly what happens with some hosting companies, and exactly what you're paying for when you purchase shared hosting.

21.2 Webserver configuration

Whether server you're running, you may need to access your configuration. On a shared environment this is tricky and often you'll be left in the dark and won't be able to change it. On a VPS or a dedicated server you'll be able to control this, giving you the ability to change what you need to suit.

21.3 PHP configuration

When you're up and running with a host, you should learn more about the environment you're working in. You may want to add PHP extensions or check which extensions you're running. You can use a handy function to output all the information you should need regarding your PHP installation. Create a file with the following and run it.

```php
<?php
phpinfo();
```

Warning!

This code shouldn't be publicly available as it contains information about your hosting environment that could be useful to an attacker.

You should see a page output. Have a scan through and take a look at what it shows. You can also run this if you're working on a local environment to find more about your local server installation.

So what might you need to change within your PHP configuration? Well, for example, let's say large files aren't uploading and you absolutely need to allow larger file uploads. You could have this capped at 1gb within your upload validation and now you'll need to modify your PHP configuration to allow files of 1gb or less. Your php.ini file contains the configuration settings for your PHP installation. To find your php.ini file, run the phpinfo function as discussed above and you'll find the path to it.

For example, the two lines of interest for the upload file size are:

```
upload_max_filesize = 10M
post_max_size = 10M
```

So, changing these as nessasary will allow for larger uploads.

```
upload_max_filesize = 1g
post_max_size = 1g
```

Naturally the reason we have to change the maximum size for POST data is that we send files using this, usually through the method of a form.

Warning!

Don't set this value too high. It's highly unlikely you want to allow very large files to your application so it's better to cut a user off uploading a huge file if they bypass the limit restrictions.

21.4 Dedicated server

A dedicated server is what it sounds like, it's your own server that yours to pretty much do what you want with. Dedicated servers are often expensive, but for good reason - you're renting an entire server! Most packages are sold with some sort of control panel, either something like Parallels or WHM, although this is often charged additionally. These allow you to manage near every aspect of your server. You can also SSH to your server to perform actions you'd normally be able to do with a shell.

Don't think that using something like WHM is absolutely required. It might be easier for you to manage the server yourself, but if you're buying a dedicated server you're probably serious about your website. In this case, you might want to consider hiring someone to manager your server(s) for you. This will avoid the need to have to pay additionally for this bulky, often expensive software.

22 How to learn

This may seem a strange question, because everyone learns differently right? Of course, but the basic principal to learning a programming language is the hard way. The hard way is not knowing how to do something and spending time working it out. It's not asking for too much help, copying snippets of code or getting someone else to do it for you. If you do this, it's not your work and you won't learn.

Everyone learns differently, but within the programming world there is a massive trend of people simply copying. If poor code is copied and copied, nothing is learned. So how can you break free from not being able to work without reference to snippets of code? I find a really good method is to just try and spend time working on a problem. You may be used to working in an educational environment whereby you practice something, remove the original material and try and solve a problem or re-create something without referring to it. This works in so many cases! I'd highly recommend taking this approach to programming. If you have no idea of syntax, functions or structure of a language you may need to refer to resources online and this is fine, but there is a difference between researching parts of a language to looking at full source code. The PHP manual is a good place to start as it gives you information with official and user posted examples. Try and skip the larger examples though, as these are usually fairly complicated and probably won't apply to your website. The official documentation often provides smaller snippets that give you quick and recommend example usage.

22.1 Stick with it

If you've spent hours attempting to solve a problem this is good for learning, but stop. Sit back and relax for a while and think about the problem without the need for code. How do things relate? What makes sense? How should the process flow? Once you've spent time assessing the problem away from a computer, you'll be in a better state of mind to start writing code that does something useful. Write it down if you need to.

I tend to find the longer I experience a problem for the better I learn. This is because you're actively seeking a solution and once that solution appears, you're more likely to remember it. Learning is a process of solving problems, not trying to absorb code. Trying to absorb code does nothing but show you other people's solutions for other problems. Sure, you can look at code that shows you how to create a forum, but what happens when you want to add a specific feature to that forum? You need to know how to write code, not how to copy it.

22.2 Teaching

Teaching is a wonderful and effective way to learn yourself, especially when others correct you. You don't need to qualify to teach and help other people so join a forum and start answering people's questions. If you get something slightly wrong, it's more than likely that someone else will jump in and correct you anyway. Thus, you have just learned something new by attempting to help others. Being part of a community that offers help and support is such a great way to learn. You also have the added benefit of then being able to ask for help yourself, and as a respected and known member of a community you'll always have people that are willing to help you back, perhaps even those you've helped in the first place.

It's also a good idea to write tutorials, however basic. Writing tutorials means you can get feedback on alternative methods, such as faster ways of doing things. There will often be several solutions to problems and it's good to get used to learning how and why things perform better. Posting tutorials to websites where comments can be posted and read are great. Just be sure to listen to this advice and update your code accordingly, or others may find your code and put it to use. It goes without saying that when writing tutorials you shouldn't post reams of code without carefully explaining exactly what you're doing and why, as this creates very little value.

22.3 Books don't always help

Not this book, of course! This book is conversational and doesn't list loads of code. If we did, it would be outdated by updates within PHP and would be rendered useless or less effective at some point. Once a book goes to print it's difficult to recall all copies of a book and say "Hey, that function for reading files isn't recommended, you should use file_get_contents instead!". Reading a programming book full of code means you're unaware that things can change in the future, and I rarely see books that warn against potential changes in the future. Think carefully before buying a book and if you do, ensure it's extremely recent and if you're unsure, check in the PHP manual to see if something has changed.

22.4 It'll click

There is a phase that I've experienced and that I know many people have too, when something just clicks. Particularly with web development, this can feel like you've accomplished a lot. Once you learn how to do something, you find that this branches off and allows you to do much more than you originally thought you'd learned. This is usually true for database and application communication, where a lot of people are unsure about what tables they need to create and how they connect, query and output this information in a way they need.

I guarantee you'll get to a stage where you can plan and deploy any database driven website with confidence. You might then need to learn more efficient ways of querying for speed enhancements but at least you'll have built something. Being able to build something originally means you can quickly mock up a website to start to get people using or it show it to those who may be involved with it. You can then spend time tweaking and improving the product. This is a 'lean' way to work and is becoming popular, where investing a lot of time and money into a product initially is no longer desired.

22.5 Starting a project

Starting a project can be scary, but it all depends on how much effort you'll put into it. If you're not the type of person that likes learning about programming and has no desire to tweak, fiddle and change until something works or works better, it's probably time to step back and hire someone passionate to do it for you. People often mistake passion for the desire to learn something when they're actually only interested in the end result - the working product.

A project takes time, commitment and passion to survive and once you're in a position to learn better and know what to do when you're stuck with something, you can focus on what matter - the people who'll be using the product.

23 The end

We're at the end of the book and I sincerely hope you've absorbed all my advice and can use it to help you learn better and drive your project forward, whether it be personal, for business or for a company you work for.

Learning a language is often a long journey if you want to learn it well, particularly when languages, frameworks and specifications are changing all the time.

My aim was to alleviate the stress of diving into a project and I feel giving a general overview of a language, rather than a code based book, is the fuel to be able to go forward and get more out of programming.

This book is a work in progress

Because I've published using Leanpub[1], I can update this book at any time and those who have purchased it will have access to the latest version as if buying it again. If there is anything that I've missed or you think something could be explained better, please let me know. I'd be happy to update the book with anything you feel would better explain or extend on anything that has been discussed.

[1]http://www.leanpub.com