PHP MENTORS

Advice From PHP Experts around the world

Flávio Silveira

PHP Mentors

Advice from PHP Experts around the world

Flávio Silveira

This book is for sale at http://leanpub.com/php-mentors

This version was published on 2020-10-27

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.

Contents

Disclaimer

- My name on the book cover is just a convention. The authors of this book are all the names who answered the questions that I've sent to them.

- The book shows some ideas and opinions about:
  - 1 - IT, computers and software: software design, software architecture, programming languages, PHP and so on.
  - 2 - Also we have some topics outside of IT: life, diet, sleep, exercise and so on.

- What you read here reflects the ideas of the author or those whose ideas the author presents.

- The topics about IT and computers are not intended to express that everything will work everywhere. Please analyze and study the topics before you do anything that can drastically change your application, software or workflow.

- The topics outside of the IT field are not intended to substitute the analysis and the words from an expert in those topics.

- The author disclaim responsibility for any adverse effects resulting directly or indirectly from information contained in this book.

Introduction

It doesn't matter what's your knowledge or the stage of your career, I can guarantee you will find something interesting in this book.

What is PHP Mentors?

PHP Mentors Book is a set of questions with topics that can help a lot of developers out there (Not just PHP People).

To answer those questions I contacted more than 200 personalities and senior PHP people around the world that came up with the material that you will read here.

Me and this Book

I don't want to extend myself in the introduction, but to explain a little about this book, I need to tell you a bit about me.

Computers and programming were not my first choice in life. I was a musician, a good one. I used to play some difficult Prog Heavy Metal pieces like Rush, Dream Theater, Symphony X and also some Fusion stuff like Greg Howe and Dave Weckl band (MPB and others also, why not?).

But Computers were always present in my life. I did a website using pure HTML, before the internet bubble, to talk about electric and acoustic guitars. And I remember reading about PHP in some forum at that time.

Doing my own websites, using Flash, HTML, Css and JavaScript was a hobby. I was really focused on music and also started a degree in Music Therapy. But unfortunately or luckily, there were some very difficult times paying the bills as a musician, not impossible, but eventually I gave up (don't blame me).

One of my students at that time was the Technical Lead in a company here in the city, so we talked and he asked me to work with him. This guy taught me the first steps and the rest is history.

I have been in a lot of different roles in my career: Developer, System Analyst, Team Leader, Manager. I Worked in small and big companies. I worked with a variety of languages for Web, Mobile, Infrastructure, etc.

2017, August 23, 10:20am and my daughter was born. I spent the year focused on her first steps and after that in 2018, I decided to start something new in my career, but I didn't knew exactly what. So I

went back to what I really love, that is being a developer. After some good years as a leader/manager, I really wanted to get my hands dirty with code, so I started to look for a mentor.

While I was searching for someone, I remembered some words said by *Timothy Ferris* in his book *Tribe of Mentors*: *"Why have one mentor, If I can get a lot of them". Tribe of Mentors* is a book where Tim asked some awesome questions for some awesome people and edited all in a book.

I decided to do the same, but in a Tech - I.T. way, focused on the programming language that I love, **PHP**.

The Questions

With the questions I tried to cover the areas that have been challenging to developers. I ended up with the follow topics:

- Work station
- Development Environment
- What to study?
- How to learn better?
- CMSs and Frameworks
- Code Quality
- Docs
- Tests
- Logs
- Legacy Code
- Code Management
- Team work
- The Future of PHP
- Career
- What to do Away from IT to become a better person/professional?

We will talk a little more about the questions in the next pages.

The People

Until now, I contacted exactly 178 people (The book is a work in progress).

The names came up from people who collaborate and created for the PHP Ecosystem: core PHP people, PHP-FIG people, frameworks people, CMSs people, libraries people, Tools people, podcasts people, book authors, community leaders and also people who inspired me in my career before this book.

The Journey

At the beginning of the project I was very motivated. I was talking with a lot of people around the world about the book and the questions. A lot of them congratulated me for the idea and said **YES** to answering the questions and to help me with the book.

But, you know, time is money, and answering the questions takes some time for everyone.

In the journey of the book, I faced people:

- Who answered the questions
- Who gave up after saying yes
- Who said yes, but never replied again
- Who replied my first contact, but never replied me again
- Who said no
- Who never replied me

Then I almost gave up, because things were not the way I imagined at first. But, it's not because things are not the way you imagined, that they are bad or wrong.

I had a very good material from the people who answered me, and I really can't keep that in a folder hidden from everybody.

So, my big *THANK YOU* to the people that spent their time to answer me and bring this project to you until now: Julien Pauli, Dries Vints, Jeroen De Dauw, Chris Pitt, Denis Sokolov, Rob Allen, Michelangelo van Dam, Chris Hartjes, Junade Ali, Jonathan Wage, Woody Gilk, Evert Pot, Rafael Dohms, Alexander Makarov, James Titcumb, Matt Stauffer, Matthew Turland, Elton Minetto, Lorna Mitchell, Tobias Nyholm, David Laietta (WolfPaw), Paul M Jones, Adam Culp, Dave Stokes, Taylor Otwell, Chris Tankersley, Pablo Dall'Oglio, Michael Moussa, Matthew Weier O'Phinney, Michelle Sanver, Ed Finkler, Adam Englander, Adrian Cardenas, Kenta Usami, Steve Grunwell, Er Galvão Abbott, Joshua Ray Copeland, Cees-Jan Kiewiet, Mark Story, Bradley Holt, Eric Poe, Matthew Brown, Larry Garfield, Gabriel Caruso, Michael Heap, Jeremy Mikola, Tessa Mero, Samantha Quiñones, Adrien Crivelli, Gary Hockin, Eric Mann, Ian Littman, Pascal Martin, Pablo Godel, Anna Filina.

I hope this list increases for the next editions…

The Cover

The idea of the cover is to show that we can follow a lot of different ways in our life and career.

The art was inspired by M.C.Escher and drawed by Mark Lester[1].

[1] https://www.behance.net/marklestercelozar/info

Before Start

- The questions are long and they have some sub-questions in them to try to reach a more detailed explanation from those answering it and also to cover some similar topics. I wanted to include them together with the answers (to avoid getting things out of the context), but the book would be way too extensive. So, I summarized them. I summarized them differently for each person, depending where they set their focus on. This way, the reading flow will be better and less boring. Check the chapter called Questions to get the complete question every time you need.

- I decided not to include a resumé/about of the people who answered the questions for some reasons:
  - For you not to judge them by their history, company where they work, books, frameworks, libraries they've built or anything like that.
  - To avoid delaying the book even more, waiting for people to send it to me and then validating their profile.
  - To encourage you to search for them.

You will find the links for their twitter and github right after the name of each one. Also there is a list at the end of this book with all the names, twitter and github profiles of everyone who answered the questions. Check chapter called *Profiles* or check the list on our website: http://www.phpmentors.com/#mentors[2].

Please, Enjoy!

[2] http://www.phpmentors.com/#mentors

Paul Jones

https://twitter.com/pmjones[3] - https://github.com/pmjones[4]

What is your current Setup? Any specific method to increase productivity? Do you make notes or snippets while working? Do you listen to music while working? How did you get to this setup?

I think it's pretty spartan. I use a mid-2015 MacBook Pro with one external screen, Mac OS High Sierra (or whatever the latest OS is). I usually work in Sublime Text; occasionally I will swap over to PHPStorm to see how my work looks in an IDE, for those who are used to code completion.

The best way for me to increase my coding productivity is to *be alone*, in a space where I can get up and walk around and talk out loud without feeling the psychological pressure of other people. This makes me feel like I can focus more directly on whatever I'm working at.

Having a home office with attached bathroom at one end of the house, away from everyone else, with a couple of working desks, bookshelves and filing space, an easy chair and ottoman, windows and outside door, and a mini-fridge and microwave – that all makes it easier to get the solitude that helps me best, while allowing me to take breaks as needed.

I have a natural pomodoro cycle of sorts, in that I habitually get up and move around when I am distracted by side-problems. I don't use Agile or Kanban or other management approach for my side project, though I do sometimes set deadlines to help me focus on them.

Do you make notes or snippets? Absolutely. It helps me to write up a file called `woes` where I write down and "talk out" the problem I am facing, exploring different aspects of it and keep track of what didn't work and why. It helps me keep track over multiple days of where I was and why I was working in a particular direction.

Do you listen to music while working? Yes, I like the Groove Armada station from Soma FM.

How did you get to this setup? Trial-and-error, mostly.

How to earn your space when entering a development team?

The most important thing anyone can do, I think, is to recognize where they *actually* are relative to the other team members. You have to tell yourself the truth about this, which can be really difficult for developers who see themselves as more senior (or less senior!) than they actually are in relation to the other people there.

[3] https://twitter.com/pmjones
[4] https://github.com/pmjones

One does well to recognize the hierarchical status system that actually exists in all organizations; walking in thinking "we are all equal" in terms of knowledge, skill, and experience is a surefire way to make trouble. (I've done that myself before.)

Now, that does not mean you cannot ask questions and offer insights, even as a relative junior; but it does mean, as we were tolkd back in the military, to "speak from your rank." That is, if you are relatively junior, don't try to challenge seniors as if you are a senior. You're not.

Likewise, if you are a senior, you have a responsibility to guide juniors by making tough decisions, giving direct and explicit instruction, and taking responsibility for their direction. Nobody likes a junior who is self-aggrandizing, and nobody likes a senior who demurs from making hard choices.

The real trouble is when you are midway between junior and senior; where do you fit in? The best I've been able to figure is that you guide the juniors as if you are a senior, and defer to the seniors as if you are a junior. Doing so willingly, without resentment, is one path toward becoming a senior.

Do you monitor your code in any way? Any continuous tool for code quality control? Any Profiling tool?

The main thing I formally monitor is unit-test line coverage; in my library packages, I am for 100% line converage, and almost always achieve it.

Informally, and now by habit, I try to keep per-method cyclomatic complexity scores low by counting the number of loops and and conditionals.

In the past, I have used Code Climate to find "hot spots" of complexity in my code, but my discipline there has waned. As for profiling, I do very little; more often, I do end-to-end benchmarks of more coarsely-grained operations to see how one project or library matches up to a comparable offering.

We see many programmers that have a lot of experience moving to leadership/management positions. Is this natural? Is there an age to stop programming? Until what age will the market hire me as a developer?

I think it is natural, though not always wise. There is the Peter Principle, which notes that ones rises to the level of ones' incompetence; that is to say, just because you are very good in a particular role, does not mean you will do well at managing others in that role.

But I prefer this fictional exchange from *Star Trek 2: The Wrath of Khan*. We find that Captain Kirk has been promoted from Captain to Admiral, and now flies a desk for Starfleet. Seeing that Kirk is obviously unhappy about it, Spock remarks: *"If I may be so bold, it was a mistake for you to accept promotion. Commanding a starship is your first, best destiny."*

It is up to you, as a programmer, to determine where your best destiny lies; if it is management, so be it. But if you try management and it doesn't fit, there is no shame in a "demotion" back to being a developer. Indeed, it would be the best thing for you to do, to play to your strongest competencies.

How is the job market for seniors? First bad, then good. I have weeks or months of latency between contracts or jobs, during which I work on side projects or other things; at some point, either searches or serendipity will result in new works, which itself will last for weeks or months. Then it's back to downtime again.

If there's an age to stop, or an age past which you will not be hired, then at 48 years old I have not yet reached it.

Looking back on your career, what would you have studied more?

I would have studied people, marketing, and sales more.

I never really did understand other people as a kid, and have had to train myself how to work with them; some will say that I still need more work there.

More specifically, I would have focused more on charm and personability, if only to get more dates with women.

What to do when facing legacy code? Is there any way to modernize an old application?

These and related questions are the focus of my 2013 book, Modernizing Legacy Applications in PHP[5].

What do you do to learn? How to learn better/faster?

The first task is always "research" – internet searches, patience, and lots of notes are my friends here. The second task is to try doing it, whatever "it" is, on my own if possible or under tutelage if necessary. The third task is "review and refinement", generally with more "research" involved as part of that. Then, finally, publication of the result either as a package or as a blog post, even if it doesn't feel like a success, to expose it to outside criticism.

Some of the critique may be jealous noise; some of it may be insipid, careless, intentionally point-missing, or otherwise low-value; but, some of it will be honest critics asking honest questions, and there may even be truly expert advice offered. Ignore (or mock!) the noise-makers, and thank the senders of useful signal.

[5] https://leanpub.com/mlaphp

How's it going to be the future of technology in 2024 and 2029?

If I knew the answer to that question I'd be investing in it somehow, instead of writing here. 2024 is 5 years away; 2029 is 10 years. Think back 5 or 10 years ago to 2014 and 2009; could you have imagined then that things would be like this now?

By the easiness that PHP brings, many devs end up NOT studying basic concepts of computer science and etc. They jump straight to coding. Are these areas important? Would it help or not?

I think it depends on if you see your primary goal as "a product" or "a program".

If the programming work is instrumental only to some other near-term goal, and you don't actually care about the program for its own sake, then not-studying basic concepts might not matter to you in putting together your product. That may or may not come back to bite you later; in fact, it might actually help you get a successful product out the door sooner, attracting customers and money and fame. Then you're stuck with a hard-to-handle codebase, but you may have enough money to hire plenty of developers to throw at it.

On the other hand, if you care about the craftsmanship of the program itself, more or less for the doing of good work for its own sake, then you are going to want to study and apply those basic concepts at some point. You will learn them anyway as you make enough mistakes. But that may mean a longer product-development cycle, and maybe not as much money as fast as you would like.

How to deal with team code management? Who is responsible for committing? Writing tests? Who manages all that?

I would opt for Git or Mercurial instead of SVN these days. Past that, you need a reasonably good branch-and-merging strategy; that strategy will depend on the size and scale of project you're working with.

I have been lucky for most of my career to work with specialists: DBAs, sysops, devops, and so on. For deployments, then, I have been able to lean on sysops and devops folks to make that process either low-effort or much less burdensome than it would have been otherwise.

Regarding tests, I say if you are able to write them at the very beginning, then do so. I don't mean TDD per se, but to incorporate testing as one of the requirements for deployable code at the start of every project. That means every developer is also a test writer.

Unfortunately, most projects are more product-focused than program-focused, which means there are no tests. In those cases, I advise repurposing at least one existing developer to focus on writing

automated tests of any kind at all; generally these end up being systems or functional tests. You can then start iterating toward better tests once you have something in place; you can improve existing tests, but you cannot improve tests than don't exist at all.

How to improve the communication between IT and stakeholders? How to avoid miscommunication and task-rework problems?

The most-important thing I have found is for a knowledgable developer, or someone else with *"skin in the game"*, to particpate directly in the sales process, whether that process involves internal customers or external ones.

Sales is where promises get made and expectations are set; the developer can help to ground those promises and expectations in reality. This has the effect of making the process take longer, and in some cases redirects the effort away from the initial "ask" to something else entirely, but the end result is greater success for everyone involved.

What is the ideal development environment for you? What about production environments? Continuous delivery? Continuous integration? How to find and fix bugs quicker in production?

My ideal is, "whatever devops has set up to minimize the differences between development and production."

What about production environments? These are generally things I've been able to leave to sysops and devops folks.

Continuous delivery? Continuous integration? I am a huge fan of CI/CD systems because they increase release cadence dramatically. A manual quality-assurance cycle can be a terrible drag on throughput from development to production.

How to find and fix bugs quicker in production? Products like Sentry[6] are a great aid here. After you integrate it into your codebase, you can monitor production for all the errors you failed to track from lack of testing, and find new and interesting errors that would never have occured to you no matter how much testing you have in place.

In the PHP universe we have excellents CMSs, Frameworks and etc. While all of those help popularize the language and best practices, many developers get stuck to it and lose some of

[6] http://sentry.io

what it is to be a PHP developer. They become developers of a specific CMS. Do you agree with this view?

It's not a question of "if I agree with this view" or not – it is merely true that developers focus on what is in front of them. There's nothing wrong with being a WordPress developer, or a Laravel developer, or Magento developer, so long as you recognize that's what you are. The trouble comes when you say that those things are "the same as being a PHP developer" and attempting to equate them with being better or worse than some other kind of developer.

For myself, I like to be broadly familiar with many systems so I can help to diagnose problems, even if I cannot fix those problems. And for the people I hire, I have much preferred that they be PHP developers first and foremost, and framework/CMS/etc. developers a distant second. I have come to realize that if you know PHP, you can learn the basics of a framework quickly; but if you only know a framework, it takes a much longer time to learn how to work without it.

What is the optimal level of documentation for an application? Where to document business rules? Where to document architecture and code? Any specific strategy?

"Optimal" is one of those things that depends on a series of tradeoffs. One could say *"more is always better"* but at what cost? Or one could say *"just enough and no more"* but what would be enough?

So you need to talk with your users and figure out where their difficulties are, then write up documentation and training around at least those things. That writing will lead you naturally to documenting the prerequisites for them.

Where to document business rules? When possible, you want the business rules to be coded into the application using the language of the business itself. If a product owner describes a list of rules in normal everyday language, the classes and methods and other code should use those same terms. When you read the code back to the product owner, he should be able to recognize the terminology. That makes the rules self-documenting within the codebase.

Where to document architecture and code? Docblocks are the normal thing to do here, though I confess with PHP gaining better typehinting for paramters and returns, I find myself doing a lot less docblock work. With typehinting, and more descriptive naming of methods and parameters, you can make the code more self-documenting than it might be otherwise.

However, that doesn't help you when writing narrative documentation on how to use a package a library; for that, you still need to write manual read-mes, how-tos and tutorials.

Are tests always present in your code? Do you write tests before code?

Unit tests are always present in my own projects, but I build mostly library packages, and those are easier to unit test. I don't do TDD or write tests before coding, but I do write them as I finish major subsystems.

Coding is exploratory and design effort; once it works and is reasonably well-factored, I write tests to make sure it keeps working later.

What is the ideal level of logging for you? What are the best strategies for saving them?

I don't do a ton of application logging, other than for debugging in development and for database query profiling. Performance and server logs are the domain of sysops and devops, so I have not had to deal with them too much.

Error logging is critical, and is always on (though of course not always displayed). As for saving them, I don't – but the sysops and devops folks sure do.

Away from IT, what you do to become a better professional? Exercise? Good sleep hygiene? Diet? What gets you out of bed every day?

The things that I do away from my professional work are not so I can be a better professional, but so that I can be a better person.

Exercise? My exercise regime is weightlifting, specifically barbell training. I like it because it tells me the truth. Programming is a discipline filled with smart people, and while smart people are good at many things, the one thing they are best at is rationalizing their failures in a way that makes it not-their-fault.

When it comes to weightlifting, you can't lie to yourself as to whether or not you succeeded in your deadlift; either you pulled that bar up off the ground, or you did not. There is nothing else possibly at fault: it is something related to you directly, and not anything else.

Good sleep hygiene?

I sleep as much as my dog and my children allow. There is no honor or status in sleeping less for its own sake; sleep is happiness, you need it, and your brain needs it.

Diet? When I turned 30, I weighed 140 lbs. In four years, I gained 10 lbs; then another 10 in 3 years; then another 10 in 2; then another 10 in 1. By the time I was 40 I weighed 180 lbs, and my vanity demanded that I do something about my weight gain.

I had tried various diets, none of them working, and decided to try a low-carb lifestyle as a last resort. On that plan, I lost 20 pounds in 8 weeks, and got down to a manageable 160 lbs. I have managed to maintain that for eight years, eating steak and bacon and eggs and heavily buttered green vegetables.

What surprised me most was that my headaches disappeared. Ever since I was a kid, I had regularly gotten terrible cluster headaches over-and-behind my left eyesocket. When I say "regularly" I mean 2-3 times a week; I would go through a bottle of Excedrin in just a few weeks. But once I started

the low-carb eating, they went away; I attributed it to the stabilizing of my blood sugar and the reduced insulin reaction from not eating sugard and processed starches. As an accidental test of that hypothesis, I ate a box of candy at the movies one day, and within hours I had a blinding cluster headache again.

What gets you out of bed every day? Some days it is the joy of duty, some days it is readiness to begin.

Do you want to say something that no other question has brought to you and you think it is important?

The single-most important thing that I don't see developers talking about, is taking care of their finances. I am not an accountant or lawyer or other professional, so you cannot take this as professional advice, but as one adult to another:

1. Get some life insurance. I have seen too many people in the PHP world and elsewhere die unexpectedly, sometimes literally sitting at the keyboard working on a program. Their families were left behind with nobody to help them, no money for bills, and no way forward. Their lives are ruined for years. If you have life insurance, your family and loved ones will have something to help with financial burdens without having to find jobs or disrupt their lives any sooner than necessary. Shoot for an amount that will return 10 years-worth of income – or at least enough income to last until your children can move out of the house and get jobs of their own.
2. Pay off all your debt. The sooner you can stop making payments on everything, the sooner you can keep all the money you are working for. Sell stuff if you have to. Being out of debt is a freedom that you cannot comprehend, until you do it – and then it's the best feeling in the world.
3. Build up your savings. There will be a time when you don't have a job, and you will need to maintain your lifestyle while looking for new work. Or you will decide you don't like your job, and want to quit – having savings as a buffer will allow you to make that decision from a place of financial strength, rather than financial weakness. You don't need to have a lot; 3 to 6 months of expenses in cash, maybe more, will give you a lot of options. (It will be easier to build up savings if you are out of debt.)

Taylor Otwell

https://twitter.com/taylorotwell[7] - https://github.com/taylorotwell[8]

What is your current Setup? Do you make notes or snippets while working? Do you listen to music while working?

My primary development machine is a 5K iMac with 16GB of RAM. I don't use multiple monitors at this time but do use multiple "desktops" on the single iMac screen. I have one desktop dedicated to email and chat and the main desktop is dedicated to *"work"*.

The primary tools I use are Sublime Text, iTerm2, TablePlus, Laravel Valet, Todoist, and Apple Notes.

I typically use Spotify to listen to music while working. I listen to a lot of deep / progressive house while working.

How to earn your space when entering a development team?

The best way to integrate into a team is to do a lot of listening and learning. Before beginning my full-time work on Laravel, I only held two programming jobs. I was fresh out of college when I started my first programming job, so I obviously had no ideas to contribute.

Do you monitor your code in any way? Is this / would it be important?

I will occasionally use the basic *"phploc"* tool on the command line to get a feel for the total number of lines of code in a project, as well as the cyclomatic complexity. However, this is not a huge or important part of my workflow. I generally know which areas of a codebase I would like to improve.

Looking back on your career, what would you have studied more and what would you have studied less?

I don't regret not studying anything more or less but I do regret not spending more time programming in college when I had much more free time. I think my skills would have been much further developed and I perhaps could have started my own company sooner.

[7] https://twitter.com/taylorotwell
[8] https://github.com/taylorotwell

What do you do to learn a subject? How to learn faster?

Typically I just read as much about the subject as I can. This involved a lot of Googling. However, If I have friends that know about the subject that is the most productive and fastest way to learn.

How's it going to be the future of technology in 2024 and 2029? How will PHP be in this scenario?

PHP will survive but of course it is impossible to tell how many new applications will be written in PHP. I suspect in 2024 the overall web development scene will not be much different than it is today for many people. Most web applications are fairly boring, typical applications.

Regardless, there are so many applications written in PHP that it will likely cement its place as a *"COBOL"* type language into the far future. I personally have written and worked on COBOL programs that were written before I was born. I've also written new COBOL programs.

By the easiness that PHP brings, many devs end up NOT studying basic concepts of computer science and etc. They jump straight to coding. Are these areas important?

It likely depends on what types of applications you are building. The types of applications you would build that require intensive knowledge of operating systems and low-level computer operations do not seem to be typically written in PHP to begin with. So, if you are building the typical PHP application, I do not think expansive knowledge of these subjects is "required".

What is the ideal development environment for you? What about production environments? What is the best way to deploy apps between those environments?

I personally use Laravel Valet on my Mac, although sometimes use Docker for building and running applications that require software that is painful to install on MacOS. I deploy and manage all of my applications on VPS servers managed by DigitalOcean. It is likely the far future deployment story will be serverless. In my opinion, it is the true "end game" in dev-ops - it is simply a matter of the tooling and pricing being compelling for the average user. I am working to make sure that is the case.

In the PHP universe we have excellents Frameworks and tools. While all of those help popularize the language and standardize the structure, many developers get stuck to it and lose some of what it is to be a PHP developer. They become developers of a specific Framework. Do you agree with this view?

I don't agree that you should feel obligated to know everything about PHP before using a framework. Babies learn how to speak in sentences before they understand what a *"verb"* or a *"gerund"* is.

Are tests always present in your code? Do you write tests before code? When are the tests worth the time and when not?

I always try to write tests for any new piece of application code I am writing, especially applications I have developed in the last 3-4 years. Most of the time I write the tests after I code but some scenarios do lend themselves well to writing the test first, so I do that occasionally.

The tests are always worth the time if you are testing at the correct level of abstraction. Many PHP developers test at a very low unit level which in my opinion is typically not worth the time and leads to very brittle, over-mocked tests. I generally start testing at the controller level and allow the tests to hit a real database, either an in-memory SQLite database or a MySQL database if needed.

I can then fill in lower-level tests in areas that I'm not extremely confident about.

Away from IT, what you do to become a better professional?

All of these things are very important. I personally try to eat healthy foods and get 8 hours of sleep every night. I lift weights a few times a week, although I will note I don't find this particularly enjoyable.

I enjoy reading, and Stoic philosophy in particular has been very helpful to me. Marcus Aurelius' "Meditations" should be essential reading for basically everyone.

PHP, perhaps more than any other ecosystem I've seen, has a group of people that advocate very strongly for "community". However, I'm not sure that finding your primary community online is a good idea. Personally, I think it's better to find *"community" *in person if at all possible.