**Getting started as a backend developer: A beginner’s guide -** [Abdulbaki Suraj](https://medium.com/@codelikesuraj?source=post_page-----c8fb24854699--------------------------------) Jun 26, 2022

Getting started in anything these days is usually hard for beginners than people actually realize. With a huge number of resources on the web and easy accessibility from anywhere in the world, navigating the right path can be an exhausting task.

In this post, I try to get you acquainted with the backend development space, a guide on what to learn, and a few lists of resources to get you started.

**What is Backend Development?**

Backend is a term used to describe the unseen activities that happen in the background whenever any action is taken on a website. Those actions could include signing in, creating an account, accessing resources on a web application, purchasing an item in an online store, and so on.

Backend development also sometimes referred to as server-side development involves creating logic and code that interacts with a database or storage media stored on a server, a programming interface that exchanges information between the user on the frontend (the part of the website the user can see and interact with) and the database and then outputs necessary information back to the user.

**Who is Backend Developer?**

Now that we know what backend development is, we can say that a Backend Developer is responsible for managing all activities that occur on a web server.

**Roles of a Backend Developer**

* Writing server-side logic with a programming language of choice
* Processing data from user requests and performing actions on them
* Ensuring data is only accessible to users that are supposed to have access to it
* Fixing website bugs and errors
* Creating and managing databases
* Collaborating with the frontend to provide seamless communication between the frontend and backend
* Developing and integrating APIs to connect with other backend applications and services

**Getting started**

Before we dive head first into Backend Development, there are a few concepts and underlying architecture we need to understand, most of which form an integral part of whatever we are going to be discussing and doing in the future.

**Prerequisite Knowledge**

Having a basic knowledge of the underlying technology isn’t a waste of time. This usually proves useful in the long run and some of them include but are not limited to the following:

* **The Internet**: How it works, what HTTP is, how browsers work, hosting, etc.
* **Database**: Types (relational vs non-relational) and their use cases.
* **Servers**
* **Frontend**: Have a basic knowledge of front-end technologies and languages like HTML, CSS, and JavaScript. Trust me on this, understanding how your data will be displayed to and submitted by the user will help you when formatting input and output. It also helps in understanding the structure of the frontend and how to effectively interact with it.

**Pick a Backend Language**

Picking a language to start with is another daunting task but I will suggest starting with one and learning its perks as well as its limitation.

One good thing about learning a language is that when you decide the current one isn’t cutting it for you, learning another one won’t be hard as the knowledge of programming concepts such as loops, functions, variables, etc. of one language is transferrable to other languages with differences in its syntax and structure.

There are a few programming languages that are suitable for writing backend code but I will only be listing the commonly used ones in no particular order.

These include but are not limited to the following:

* NodeJS (JavaScript for writing server-side),
* PHP,
* Python,
* Ruby,
* Java,
* Golang and so on.

Amongst the above list, I started with and still use PHP as it runs on various platforms (Linux, windows, etc.) and it's compatible with almost all web servers used today (Apache, IIS). It's free to use and also easy to learn while supporting a wide range of relational databases such as MySQL which is another popular and widely used database.

After picking a language, learn basic concepts like variables, loops, functions, arrays, if-else, and switch statements and then once you have a grasp of these, take it a step further by learning OOP — Object Oriented Programming if the language supports it. This is not a must yet, but you’ll eventually need to learn that.

**Pick a Database Management System**

You have learned how to write code in your preferred language but you feel like something is missing. All the codes you have been writing up till this point probably won’t have persistent data, that is, the user inputs are stored temporarily. This is where a database comes in. It helps retain such data and provides them again on request for reading and modification.

A Database Management System (DBMS) is kind of like the software used in managing the database itself. This could further be classified based on the database model they support which can be **Relational databases** (usually made up of a series of tables that contains rows and columns and they use a Structured Query Language (SQL) for interacting with the data) and **Non-relational databases** (also referred to as NoSQL, doesn’t use the tabular form of storage used by relational databases).

For starters, go with a relational database and my preferred choice would be MySQL as it pairs nicely with my preferred language which is PHP.

It is not a must for you to go with these choices, your choice of backend language and database may differ from mine, just do a bit more research and find a pair that works for you.

**Learn basic CRUD operations**

You must be wondering what CRUD means by now, don’t worry it's nothing technical. It's just an acronym for the basic operations that occur in the backend which are

* **C**reate: receive data from frontend and save it to the database,
* **R**ead: fetch data from the database and output to the frontend,
* **U**pdate: edit and update data stored in the database, and
* **D**elete: delete data stored in the database.

There are a few tutorials online (videos and articles) detailing how to go about this in your language and database of choice.

**Learn authentication and authorization**

After getting comfortable with CRUD operations, how about we put some privacy into it. Up until this point, any user has access to all parts of your web application. In order to prevent this and allow only certain users to have access to certain parts, information, and actions on your web application, we need to add authorization logic to your web application.

This involves creating a login and signup code (authentication) that only allows users to have access to other parts of your application asides from the login and signup screen and then based on the type of users (whether standard or administrator) you restrict access to some other parts further inside your application.

**Start building projects**

If you’ve managed to persevere and get this far in your learning path, I can confidently say that you’re on your way to becoming a world-class backend developer. Since you can confidently write code to create, read, update and delete data from a database, and add authentication and authorization to your application, I think you can build the next Facebook now.

I am only kidding though, but with these, you can start with projects from as little as a Todo List (every beginner’s favorite) and then tune it up a bit by adding authentication and authorization in order to allow only the owners to view their own lists.

**Learn a framework**

You have been working on a few projects now and have begun to notice some repeating patterns, a typical example being writing login and signup logic, authorization, and writing logic to save and retrieve data from the database and they are beginning to get tedious.

Some of you might have even found libraries and packages online, others may have a reusable and customizable template they created to help them with these tasks, anyways, you are all trying to make your work easier and focus more on what matters right, and that’s where this wonderful thing called **FRAMEWORKS** come in.

A framework is a programming tool or set of reusable codes that come with pre-written components or solutions that can be customized in order to speed up development. Using a framework can save you a huge amount of time that would have been spent building things from the ground up and because of this, reduces the chances of errors.

Other reasons to use frameworks are:

* reduce code duplication,
* simpler testing and debugging,
* you can focus on writing code that is specific to your project (more like business logic),
* easily reusable,
* comes preinstalled with industry-standard conventions and security measures etc.

Frameworks are usually built on top of existing programming languages some of which include but are not limited to the following:

* PHP — Laravel, Symfony, CodeIgniter, Yii, CakePHP, etc.
* Python — Django, Flask, etc.
* NodeJS (JavaScript) — Express.js, Meteor.js etc.
* Ruby — Ruby on Rails, Sinatra, Camping, etc.
* Java — Spring etc.
* Golang — Fiber etc.

**Final notes**

One simple strategy that has always helped people to grow in this field is to learn by doing. Don’t get caught up watching tutorials, reading blog posts, and never actually taking your time to practice all you have learned, take time after learning to apply this by building projects. Most people make the mistake of learning a lot of theory without actually applying it.

It doesn’t matter if you picked up some dude’s code from the internet, take your time, read it, yes, I know what I said, read the code. Understand the purpose and what every function is trying to achieve then try recreating it by building your own.

You need not create an exact copy of the project but provided you understood what it was trying to achieve and the process that was taken to achieve then you’re good.

Take your time, understand concepts and keep on improving your skills. Join communities and reach out whenever you are having issues and let google be your best friend. Even senior developers still search for a few things once in a while.

There are still a lot more topics to discuss and expand on, but that’s not the purpose of this post. It's only but a guide as you will discover other topics and concepts I didn’t talk about here.