Do you still remember those days when websites were built in one text editor? HTML and CSS were all you needed to master to create them. It brings a smile to my face to think of those times. Why? Firstly, I imagine how easy website development once was (or wasn't it, since we have now UI editors?). Secondly, I remember how ridiculously the websites built that way looked, especially compared to today's ones. Nowadays, any piece of software, whether it is a website, an app or any other project, needs several pieces of technology to allow proper functionality and look. Web development processes evolved, app development evolved aswell. To understand them better, I would like to devote a few words to the stacks meaning in aspect of web applications development

What is a Tech Stack and what is history of Tech Stack?

A tech stack is the combination of technologies, programming languages, tools, and frameworks used to build and run an application. Applications can have different amount of items in their tech stack and overall complexity increases with every new added stack. There are 3 – 5 main groups of tech stack types, depending on how you try to group them. Those would be:

- **Frontend stack** – client-side part of the application. For example HTML and JS that is being run in browser.
- **Backend stack** – server side part of the application. Often besides logic here are included API connections
- **Database stack** – sometimes considered part of the backend stack
- **DevOps stack** – server setup, contenerisation, virtualization technology, automated testing and deployment
- **Web3 stack** – decentralized/smart contract layer. Sometimes considered part of Frontend stack or part of Backend stack

So as it seems it sometimes does not sound easy to assign certain part of the layer to specific tech stack. It could be a matter of dispute for example – is Redis part of DevOps or Backend? But really, that classification is not as important as actual relations between different stacks – but I will come back to this later. For now, another observation can be noticed that tech stack of web application has grown over time:

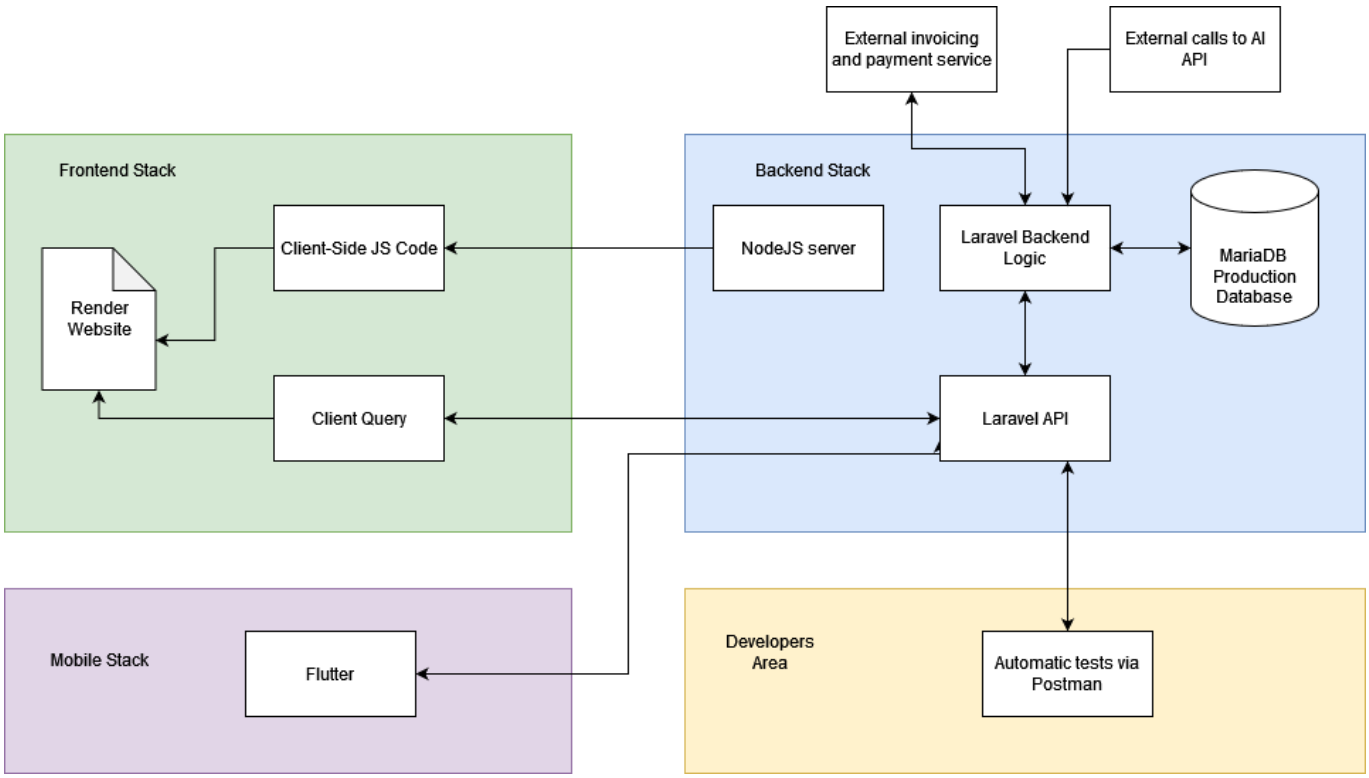
1. Frontend only website – this were the simplest websites written in HTML. One

might argue that you needed server (like Apache) to provide even for a simple HTML website, but on the other hand you could run it locally without server, just using your browser. So this included most often HTML and JS
2. Frontend + backend + server – now here is jump from one to three stacks. When need to have dynamic websites emerged, PHP was introduced. So now HTML websites to be dynamic could use PHP backend. But PHP backend requires a server to run. So you actually needed 3 stacks to run such application (I'm considering "server" equivalent to Apache in here).
3. Frontend + backend + server + database – now here comes the fun part, because when you allow to store and reuse data, you can do very wide amount of tasks – starting from simple "guest book" or user management, to very complex applications. And here it is worth noting that there are few most common setups for such approach, such as:
    1. **LAMP** – meaning Linux + Apache + MySQL + PHP
    2. **WAMP** – meaning Windows + Apache + MySQL + PHP
    3. **MEAN** – meaning MongoDB, Express.js, AngularJS and Node.js
    4. And list goes on, in any combination
4. But Windows and Linux, and even same distribution of Linux, could have different setup – for example different set of packages – so whole environment was sometimes hard to reproduce. So, to make application environment reproducible, and deployment process more scalable and automatic, you need another stack – DevOps – which containes at least contenerization (such as Docker with or without Kubernetes or LXC, also it's worth to mention Terraform) and CI/CD for automation. Now, if required, you could add to this stack hosting itself – such as AWS.
5. For our current analysis we could add Web3 stack aswell. This is fairly new topic, but alredy very complex and developed, because it provides decentralized connection between different users.Such servicecuold be for example Blockchain Wallet and Identity Layer.
6. And there is newest layer to be considered here – AI layer. It is not as new topic as you may expect – because we may consider beginning of AI somewhere in 1950s – but now the computing power and rapid advancement in simultaneously processed operations made it possible for everyday use. Some may argue that it would be part of the backend stack, but in my opinion it should be treated separately.

We can probably expect in the future that tech stack will become more and more complicated, but you can only imagine and guess what could be next. For example

AR, or maybe something that will originate from quantum computing. But for now, let’s close this list for the considered range – web applications.

Tech Stacks vs Frameworks and Tools



It must be explicitly said: tech stack is not the same thing as framework, although most common approach is using one framework for each stack. It makes a lot of sense, because then mutliple developers work in the same framework, which uses the same logic and “philosophy” across the stack.

It is also worth noting that while I have written about different Agile frameworks, they are NOT tech frameworks. As a reminder: Agile is a methodology based on a set of guiding principles outlined in the Agile Manifesto, so Agile frameworks are **project management** frameworks. Agile provides a way to manage the project and the development process, but it does not prescribe any particular tools, programming languages, or technical structures. It is compatible with various tech frameworks, languages, and tools, as it’s more about the approach to how work is planned, iterated, and delivered.Tech frameworks are sets of software libraries, tools, or components designed to help developers create applications more

efficiently, which provide a structured way to write code and handle common tasks in application development.

So fameworks in terms of tech stack can be for example:

- For frontend stack – ReactJS (which is most commonly used by Sailing Byte), Vue, Angular
- For backend stack – Laravel (most commonly used by Sailing Byte), NodeJS, Django, CodeIgniter
- For devops stack – Kubernetes might be considered a framework, ISPConfig aswell

There are also **tools and libraries**, which are not frameworks by themselves, but are often (although not necessarily!) considered part of the frameworks, but can be used in different frameworks aswell. As a non-technical business owner you mostly won't be interested at those at all, because they can but don't have to be used by developers to achieve a goal and we're starting to touch here programming part of the project. Such tools are for example:

- For frontend – Webpack, Babel
- For backend – Composer, Artisan
- For devops – Docker, Apache, Nginx, GitLab CI/CD

Tech Stacks and Developers Responsibilities

Let's get back to simpler, non-technical aspect of tech stacks. This is consequence of all of the above, but just to order knowledge a bit and ensure that we're on the same page, let's name it specifically. Depending on which stack developer specializes in, his position will be called respectively.

Full Tech Stack Developer

A full stack for web development covers the complete set of technologies needed to build web application for given project. So full stack developer is someone who can handle ALL technologies related to given project. As presented above, this can be quite a bit amount of technologies.

In Sailing Byte we do consider developers are more efficent if they are experts in one topic, but have general knowledge on other related topics. So we do not work

with full stack developers, but rather developers specialized in one stack, whom understand that stacks have to be connected between each other to form whole functioninig application. Such specialisation is called “T-Shaped people”.

Front End Developers

Front-end developers are responsible for all elements of a website that the recipient interacts with. This includes graphics, the appearance of animations, various fonts, and layout. One of the most important elements of web building that front-end developers manage is responsive web design. This constitutes what the website looks like on different devices. Front-end developers use such languages as HTML (yes, I know, HTML is rather data structure than language, but let’s not get into such details…) or JavaScript.

Back-End Developers

The code of the website is what the back-end web developer cares about. These web development specialists will take care of all server connections, page databases and functionality of the internal architecture of web pages. They most often work in such languages as PHP, Python, Java or MySQL.

DevOps

Such person in general is responsible for environment where application resides. This includes, but is not limited to, elements such as server architecture, server itself, deployment automation, virtualization, automatic scaling and so on. So again – it is position strictly conneted to technologies such as Docker, Linux, AWS, Kubernetes, LXC, GitLab CI/CD and so on.

Other specialists and developers

You can easly see the patter in here, and list goes on. To name just a few, you could think of **mobile developer** specializing for example in Flutter or **AI developer** specializing in different LLMs. But essentially idea stays the same – if there is specific stack used that will be developed within the project, then you will most probably need developer that specializes in that stack, knows frameworks and tools that are used in such stack, and is able to work with them in such way, that his “silo” will be efficiently communicating with other stacks.

Why is choosing proper Tech Stack important?

There are many reasons why a properly selected tech stack is important, such as high security, functionality or accessibility. However, from a business perspective, there is one that stands out – **scalability**. This is connected with the ability of your web application to handle an increasing number of users. After all, this is the purpose of any piece of software used in business: to attract more and more users. The software web developers build needs to be created with growth in mind.

And you may noticed, that organizing development Tech Stack silos sound like a separation, while your application needs to work **as a whole**. While separation might sound bad, it is actually quite helpful in terms of work oragnization, especially for larger applications. Stacks to communicate between each other will require some kind of communication between each other – for example between frontend and backend stack there may be API. This API can be standardized – and if standardized, it can be testable – so provides **testability**. Such tests would be black-box tests because we are not diving into code of frontend or backend itself, but we are just running tests if communication on endpoints is good both ways. This will be true for example for haedless applications, when backend provides API, that frontend connects to, and that API is tested on Postman, while also it is possible to connect new frontend (like Open API or N8N/Zapier automation or even mobile application) without recreating whole backend.

On the business prerspective, selecting proper tech stack also is important for further management. You woudn't want to invest in something that will become obsolete (or already is obsolete!) because that will mean a dead-end for application security, limited number of available developers, lesser ability to implement new features and so on. Essentially, for developmetn you will need a software house with experienced developers for selected tech stack.

Sample Tech Stacks for Mobile Apps and Web Apps

While there are hundreds of possible tech stack sets, some are used more often than others, and let's describe few of those, which are quite commonly used – with their variations respectively. While you may expected this section to be separate for web and separate for mobile, I will keep everything in one list in here – this is

because you can extend each application by using another stack. For example, you could add Flutter to LAMP stack provided that PHP delivers necessary endpoints.

LAMP and WAMP

Meaning – Linux/Windows + Apache + MySQL + PHP. Already mentioned in here two tech stacks, very similar to each other – they only differ with operating system. If written properly (and if not using OS specific functionalities), “AMP” part of the stack can be moved between those two. Another part of the stack worth noticing is “MySQL” – nowaday it is often replaced by MariaDB. But both MySQL and MariaDB are relational databases, and on some projects it is better to use non-relational databases, such as NoSQL. In that case you could call such stack LANP.

The MEAN Stack

The name consists of the first letters of the used technologies: **M**ongoDB, **E**xpress.js, **A**ngularJS and **N**ode.js. However, I like to think of this stack in a more informal meaning of *mean (good).* The whole stack can be written in JavaScript, and the code can be copied across the whole application. This makes MEAN Stack’s use as easy as pie. On top of that, it is also free, and open-source and produces apps characterized by high flexibility and scalability. MEAN is widely used to deliver web applications.

Mobile App using Firebase and Flutter

This stack leverages Flutter, Google’s cross-platform UI toolkit, to create native-looking apps for both iOS and Android from a single codebase. Dart is the primary language for Flutter, chosen for its ease in creating expressive UIs and high-performance applications. Firebase serves as the backend, providing essential services like real-time databases (Firestore), authentication, cloud storage, and analytics. Firebase’s serverless approach simplifies backend management, enabling real-time data synchronization and fast prototyping. For state management, Provider or Bloc can be used to ensure efficient state handling across the app. This stack is ideal for startups and small teams due to its quick setup, scalability, and flexibility in both development and backend services.

Mobile + Desktop App using Electron

In this stack, Electron is the core framework, enabling the development of cross-

platform desktop applications using web technologies such as JavaScript, HTML, and CSS. Electron allows you to build native-feeling desktop apps for Windows, macOS, and Linux by packaging a web app with a Chromium-based browser engine. For the mobile component, React Native or Flutter can be integrated with the Electron codebase to create a unified code approach across platforms. Node.js is used for backend processes within Electron, allowing the app to interact with the file system and other native resources. This setup can also integrate APIs or databases like SQLite for local storage or MongoDB Atlas for cloud database support, enabling seamless data access across devices.

Web + Mobile App using Laravel and AI

This stack uses Laravel, a PHP framework for building web applications with MVC architecture, for the server-side of both the web and mobile applications. Laravel simplifies authentication, routing, and data management with tools like Eloquent ORM for database interactions and Blade for templating. For AI, Python is commonly used to develop machine learning models, which are then exposed as APIs (e.g., using Flask or FastAPI) and consumed by Laravel. Vue.js or React can be paired with Laravel for a dynamic, interactive frontend, especially in the web version. Flutter or React Native can be used for the mobile app, connecting with Laravel's APIs to access data and AI-powered features, such as recommendation engines or image recognition.

Web3 Tech Stack for a Decentralized Application (dApp)

For example, a simple dApp for decentralized voting could use the following stack:

- **Blockchain Layer:** Ethereum for its consensus and transaction handling.
- **Smart Contracts:** Solidity and Hardhat for writing and deploying voting logic.
- **Decentralized Storage:** IPFS to store candidate information and voting data.
- **Frontend:** React and Ethers.js to build the user interface and connect to Ethereum.
- **Wallets:** MetaMask for user authentication and transaction signing.
- **Oracles (if required):** Chainlink to bring in any off-chain data (if applicable).
- **Off-Chain Services:** The Graph to index and retrieve blockchain data efficiently.

SAILINGBYTE

Your Experts in App and Web Development

If stacks meaning, site technicalities or website development seem a little too much, the Sailing Byte can give you a hand. Book a call to talk about your project or an already-existing website or app. We have years of experience and practice in developing software for our clients. Contact us, let us do our magic, and we assure you – you will not regret it.