Instituto Superior de Gestão e Administração de Santarém

Mestrado em Engenharia de Tecnologias e Sistemas Web

Understanding the complexity of creating a Codeless Web-Technology Platform

Paulo Constança

Docente:

Mestre Marco Tereso

Unidade Curricular: Dissertação

Santarém

Ano letivo 2021-2022

Abstract

The internet and its provided services have been growing exponentially since made available to the public. Advances in the processing power capabilities of computers, but also advancements in the form factors used to interact with the machines are also prevalent, and with the rise in popularity of mobile devices, the internet has experienced a very large growth both in users and content diversity. Web Technologies have also grown, with many popular platforms being written in these technologies due to its cross-device and cross-platform capabilities, ease of use, access, and programmability. Tools, programming languages and their supporting technologies have also been growing in functionality, allowing developers to not only in an easier, but also faster manner produce websites and web applications. Due to these advances, interest in developing software is no longer exclusive for computer programmers, and with the appearance of low-code and codeless platforms, tools which provide a user with little or no coding knowledge with the capability of developing a variety of different software, with the outputting result being what the base platform was designed to help conceive. Aligning these two interests, with this thesis we dive into the coding process to better understand how a computer works, how a software is made, the limitations and challenges the developer faces, and translate these processes by using modern web-based tools, into the creation of a codeless, web-based platform capable of creating the most typical data-information oriented software, and by doing so understanding the complexity behind the creation of software and all it's different use case scenarios.

Keywords: Internet, Web Technologies, Computers, Mobile Devices, Low-Code, Codeless, Programmer, Coding, Data, Information, Software

GLOSSARY

- Hardware Terms
    - AGP (Accelerated Graphics Port)
    - ALU (Arithmetic & Logic Unit)
    - APU (Accelerated Processor Unit)
    - ASCII (American Standard Code for Information Exchange)
    - ATA (Advanced Technology Attachment)
    - ATX (Advanced Technology Extended)
    - BIOS (Basic Input and Output System)
    - CPU (Central Processing Unit)
    - CU (Control Unit)
    - DIE (Integrated Circuit)
    - GPU (Graphics Processing Unit)
    - HDD (Hard Disk Drive)
    - IDE (Integrated Drive Electronics)
    - IPC (Instructions Per Clock)
    - MMU (Memory Management Unit)
    - mSATA (Mini Serial Advanced Technology Attachment)
    - PCI-E (Peripheral Component Interconnect – Express)
    - PSU (Power Supply Unit)
    - RAM (Random Access Memory)
    - SATA (Serial Advanced Technology Attachment)
    - SSD (Solid State Drive)
    - SoC (System-on-a-Chip)
    - USB (Universal Serial Bus)

- Software Terms
    - AI (Artificial Intelligence)
    - AMP (Accelerated Mobile Pages)
    - API (Application Programming Interface)
    - ASM (Assembly Language / Assembly-Source-Language (File))
    - BPM (Business Process Management)
    - CRUD (Create, Read, Update, Delete)
    - DLLs (Dynamic-Link Libraries)
    - HLL (High-Level Programming Language)
    - HTML (Hypertext Markup Language)
    - IDE (Integrated Development Environment)
    - LLL (Low-Level Programming Language)
    - MITM (Man-In-The-Middle)
    - ML (Machine Learning)
    - MLL (Medium/Middle-Level Programming Language)
    - MVC (Model, View, Controller)
    - OOL (Object Oriented Languages)
    - OS (Operating System)
    - PHP (PHP: HyperText Preprocessor)
    - PWA (Progressive Web App)
    - REPL (Read, Eval, Print, Loop)
    - SSL (Secure Sockets Layer)
    - TLS (Transport Layer Security)
    - UI (User Interface)
    - URI (Uniform Resource Identifier)
    - URL (Uniform Resource Locator)
    - UX (User Experience)
    - WIMP’s (Windows, Icons, Menus, Pointer)
    - WWW (World Wide Web)
    - WYSIWYG (What You See Is What You Get)

DEFINITIONS

- Application Programming Interface (API)
    - Set of definitions and protocols for building and integrating application software.
- Bug
    - Error, fault, flaw, erratic or unintended (for the software's purpose) behavior in software.
- Compiler
    - Computer program designed to translate a programming language source code into machine code.
- Computer Storage Capacity and Speed
    - Computer Storage is usually measured in 3 mainstream methods: Bits, Bytes or Rounded Units
        - Bits: Binary Language (Base-2) representation of a 0 (zero) or 1 (one);
        - Bytes: 1 Byte equals to 8 Bits
        - Rounded Units: 1 KB, also referred to as 1 KiB equals to 1024 Bytes. Due to manufacturers rounding methods, 1 KB is sold as being 1000 Bytes, creating rounding errors as the capacity increases.
    - Common Measurement Units:
        - Speed:
            - 1 Bps (Bit per Second; 1 Kbps (KiloBits per second), 1 Mbps (MegaBit per second) (…)
        - Real Capacity:
            - 1b (Byte), 1 KB/KiB (KiloByte/BibiByte), 1 MB/MiB (MegaByte/MibiByte), (….)
        - Rounded Capacity:
            - Same as Real Capacity, but every KB is rounded to 1000 and then converted to bigger units (AkinsIT, 2022).
- CRUD Operations
    - CRUD is an acronym for "Create, Read, Update and Delete", and refers to the process of manipulating data in permanent storage such as a database.

- CRUD Platform
    - Simple software where the main objective is to Create, Read, Update, and Delete information stored in a persistent database. An example of these types of software is billing processing software.
- Debug / Debugging
    - Process of identifying and removing potential logic errors (bugs) in a software code that can cause the software to crash or malfunction.
- Developer
    - Person or company who writes the software.
- Framework
    - Base structure used as a base to create software: allows a developer to build software faster by providing it with ways to skip complex programming tasks or by providing built-in tools to perform certain coding tasks, such as defining database relation models or accessing data (Laravel, 2022).
- Hardware
    - Physical components of a computer.
- Idempotent
    - Property where certain operations in mathematics and computer science can be executed multiple times without changing the result beyond the initial application (Cambridge Dictionary, 2022).
- Man-In-The-Middle (MTIM)
    - Type of cybernetic attack where the perpetrator redirects the original connection trough their systems and either steals the information or manipulates it while it's being transferred to the destination machine. Common Applications of these attacks are in IP Spoofing attacks, where the attacker disguises their machine as being the real application the user is trying to access or DNS Spoofing, where the attacker changes the DNS Query results redirecting the user to malicious pages of their choosing (NIST CSRC, 2022).
- MVC Framework
    - A Model, View, Controller Framework is an architectural pattern that allows for the separation of applications code into three main logical components:
        - Model:

        - Corresponds to all the data-related logic, such as ways of retrieving information or manipulating information in a database, including accessing this data's relation with other data.
    - View:
        - Component responsible for all the UI (User Interface) logic of the application
    - Controller:
        - Interface that acts as a bridge between the model and the view, allows all logic of incoming requests to be properly processed and directed towards the different operations (such as updating an entity in the database) (Laravel, 2022).
- Programming Language
    - Written or visual language constituted by sets is a set of commands, instructions, and other syntax represented by strings or graphical program elements used to instruct a computer by being translated into machine code.
- Progressive Web App (PWA)
    - Web application designed to look and behave as a mobile application. These rely on the use of service workers, manifests, and other web-platform features in combination with progressive enhancement to give users an experience on par with native apps (Britannica Dictionary, 2022).
- Software
    - Computer software is constituted by instructions code written and designed to guide the computer to perform a specific task (Britannica Dictionary, 2022).
- Server / Client Machine
    - A Server machine is a computer responsible for providing other machines in the network with services and information. By contrast, a client machine is the machine which requests a server said services or information. The client machine can also be referred to as "End-Point" (IBM, 2021).
- WIMP's
    - Design and concept approach of developing and interacting with computer software. It consists in software designed with the principle of having Windows, Icons, Menus and Mouse Pointer to interact with the user (Interaction Design Foundation, 2022).Web Development

- Web Development, or Web Programming, refers to the writing, markup, and coding process of a browser-rendered website or application. This process involves the creation of the web content to be executed by the web client and server, including scripting and network security (Ombeni & Thomas, 2016).

- World Wide Web (WWW / WEB)
  - The WWW is also commonly referred to as "The Web". The "Web" term comes from the fact that websites are a series of interconnected pages, connected between each other in a "Spider Web" like-form (Ombeni & Thomas, 2016).

TERM ORIGINS

- Computer Bug
  - The first computers were massing-size machines, with many of their components being eye-sight visible. Due to this, external factors could easily interfere with its operation. On September 9, 1947, Thomas Edison and its team reported the first known "computer bug", where a literal animal (bug) was found in Harvard University's, Massachusetts, Mark II computer, preventing it from operating correctly (National Geographic Society, 2022).

- World Wide Web (WWW / WEB)
  - The WWW is also commonly referred to as "The Web". The "Web" term comes from the fact that websites are a series of interconnected pages / computers, connected between each other in a "Spider Web" like-form (Britannica Dictionaries, 2022).

INDEX

TABLE

FIGURES INDEX

INTRODUCTION

Since the creation of the first mechanical computer, the Babbage Difference Engine by Charles Babbage in 1822 (Jacobson, 2019), and more recently with the first electronic computer, the ENIAC (Electronic Numerical Integrator and Computer), designed by John Mauchly and John Eckert and completed in 1945, computer technologies have grown exponentially over the last decades, with processing power and transistor sizes growing by each new generation of processors released. Moore's Law states that every two years the number of transistors inside a computer processor should double, and this has been mostly true ever since this observation was made by Gordon Moore in 1965 (Mack, 2011), with the latest to date commercial processor with the most transistors inside being Apple's ARM based M1 Ultra boasting over 114 billion transistors in an area of 432mm2, or little over 4cm2 (Shilov, 2022).

This growth in processing power, is accompanied by lower power consumption, allowing mobile form factors to grow ever more powerful without having to resort to exotic battery capacities and technologies (Papermaster, 2015).

The device market is both divided by form factors, that is, different devices with different usage types, such as a computer operated by mouse and keyboard, a smartphone operated by touch-screen, and many more (Crucial, 2022), by operating systems such as Microsoft Windows, or Apple's MacOSX (Stephen, 2017) and fundamentally by CPU microarchitectures, or rather, the "machine language" a processor operates with, such as ARM or x86-x64: this last point is very important, as programming a native application for a desktop computer, powered by Intel's x86 or AMD's x86-64 architecture is very different from programming for Apple's ARM silicon due to fundamental differences between both the CPU architecture but also the operating system's programming code running over the hardware (Michael, 2019).

This is where Web Technologies, the core subject for this thesis, are introduced: Web Technologies, due to how high-level they are and how, by design, device-abstract they are, allow multiple CPU architectures and operating systems to run the same application natively, without having to re-compile or re-code the entire application for a specific architecture, operating system or even device form factor, but this versatility does come at a cost: performance (Mozilla Developers, 2022).

In the present document, to properly introduce web technologies and how a codeless programming platform can originate from using these technologies, we will dive into the very basic aspects of what makes a computer work, passing through all the fundamentals, such as what is software, how software is developed, what are programming languages and a lengthier introduction to the web-technologies that power today's internet and how all these technologies can work together to allow a non-programmer to create a purpose-specific software without having the required knowledge about how to code in web technologies languages.

To introduce the ideas in a gradual and simplified way, this document is divided into five main sections.

Beginning with a Hardware Introduction section, which includes an introduction to the basic functionality of the hardware which composes a working computer, and followed by a Software Introduction section, which introduces the reader to the various types of software which operate in a modern computer and how software is developed by explaining the software creation cycle.

Entering a more technical but simplified approach, the reader is introduced to the Programming Languages allowing him to understand the abstraction layers added by each generation of programming languages and how it impacts both the coding process but the performance of the resulting application: it also introduces the user to the various types of programming languages and derived complements which exists to help the developers coding process.

After a solid understanding of how the computers, software and programming languages work, the reader is then introduced to Web Development, along with some of the most web-centric technologies, giving more insight on how a web application functions, as well as the concept of "Progressive Web Applications", a more recent concept where the traditional Client-Server methodology is blurred to allow Web-based applications to behave more like native applications.

Finally, using all the above technologies combined culminates into the final section, the Codeless Platform Concept is presented, whereby using a conceptual codeless platform as a basis the reader is shown how a codeless platform could be created to allow users to make a task-specific software suited to their needs.

In a less traditional way, the reader will also be gradually introduced to the state of the art of each technology introduced, to help fundament the knowledge introduced during each section, finalizing the document with the conclusion of what this thesis tries to improve upon:

solidifying the concept of creating a codeless platform using only web-based technologies, with the document finalizing with an overall journey conclusion and future work possibilities.

INTRODUCTION TO COMPUTER HARDWARE

Computer hardware is the definition given to identify the physical components that constitute a computer, with these components being classified as being internal or external.

The internal components usually reside inside a metal housing (case) to which most people generalize by addressing it as "the computer". This housing is typically built following various standards to allow a variety of many different internal components to be installed, being examples of such standards the ATX *(Advanced Technology Extended)* standard, developed by Intel® in 1995, which determines the physical dimensions the Motherboard of a computer must have to fit a specific computer case (Intel Corporation, 2006).

External components, such as Mouses, Keyboards, Printers, Monitors, Speakers, and many others also use specific standards to communicate with the motherboard of a computer.

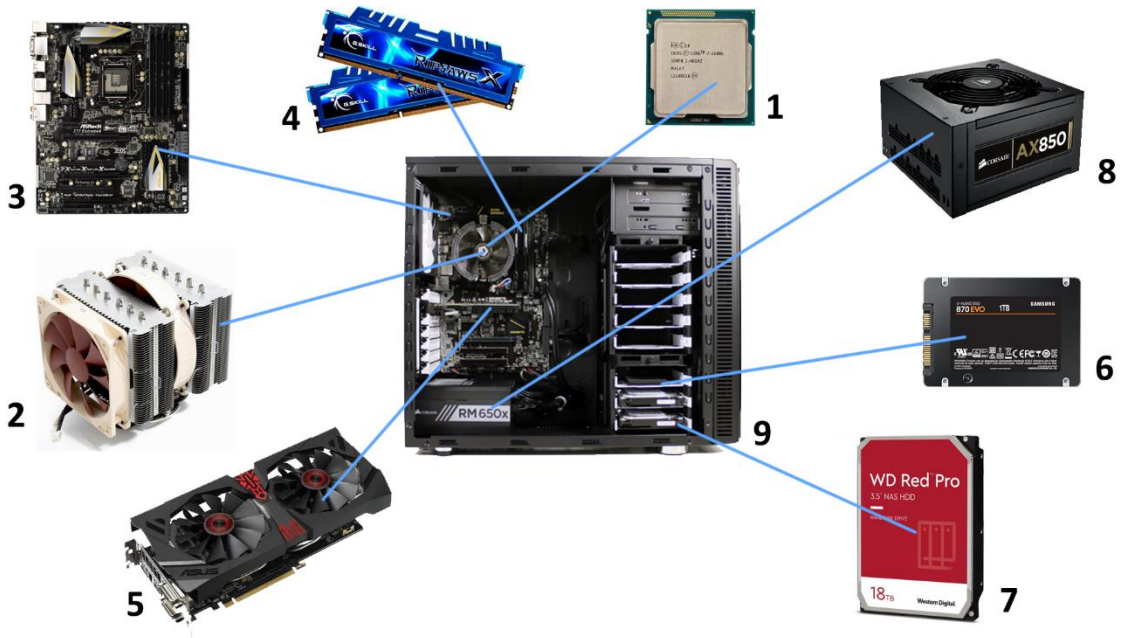


Figure 1 - Internal Computer Hardware Examples

To understand the importance of standards, let us address the typical hardware which composes the average modern day personal computer (identified in figure 1):

- [1] CPU *(Central Processing Unit)* – Responsible for most typical processing operations

- [2] Cooling Components:
    - FAN – Circulation of Air
    - Cooler – Metal part used to dissipate heat from components
- [3] Motherboard – Central component which interconnects all other components
- [4] RAM *(Random Access Memory)* – Stores temporary information necessary for operations
- [5] GPU *(Graphics Processing Unit)* –Graphic processing operations
- [6] SSD *(Solid State Drive)* – Non-Mechanic Storage Medium
- [7] HDD *(Hard Disk Drive)* – Mechanic Storage Medium
- [8] PSU *(Power Supply Unit)* – Stabilizes and transforming AC power into DC power for the computer components
- [9] Case – Houses all computer components inside (CHope, 2021).

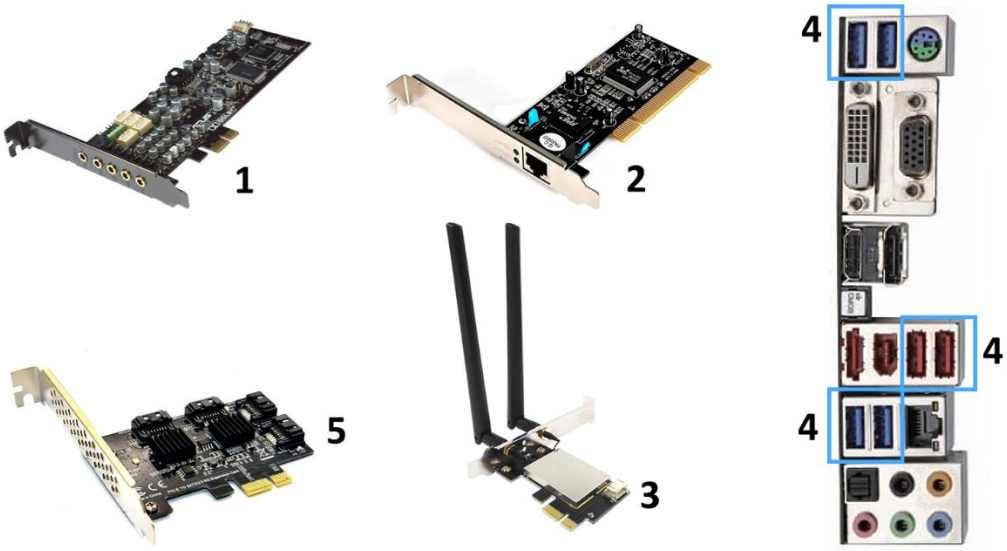


Figure 2 - Add-On Components and motherboard I/O Interface examples

Due to the evolution of the hardware, many components which are also available as add-on card are nowadays integrated directly into the motherboard such as (listed in figure 2):

- [1] Sound Card – Responsible for processing sound
- [2] Network Card – Wired communication with other computers in a network
- [3] WI-FI *(Wireless Network)* Card – Wireless communication with other computers in a network

- [4] USB *(Universal Serial Bus)* Controller Cards – Standard connector, allows hot-plugging of external hardware, such as Pen Drives, Mouses, or Keyboards.
- [5] 6Hard Drive Controllers (CHope, 2021).

Other components worth of mentioning are the hybrid-solutions where a single computer component has integrated multi-purpose silicon, such as:

- *APU (Accelerated Processor Unit)* - Constituted by a CPU and a GPU in a single chip;
- *SoC (System-on-a-Chip)* – The evolution of the APU, this chip contains not only the CPU and GPU, but also many other controllers, such as the Storage Controller and Network Connectivity controller

This variety of different components can only operate due to the implementation of standards. These dictate not only the physical dimensions a specific but also ensure the connections both physically and electrically are compatible, allowing the operation of these devices seamlessly.

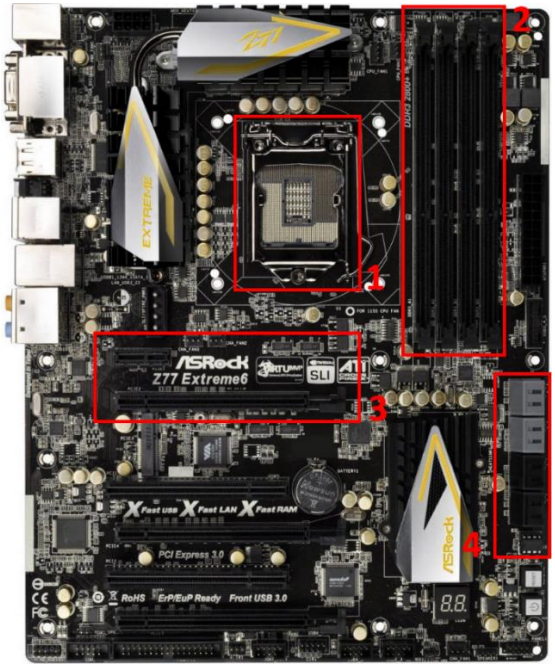


Figure 3 - ASRock Z77 Extreme6 Motherboard

Being the Motherboard the central piece of the modern computer, many standards are included on this component. A typical modern desktop computer motherboard has at least the following standards, identified in figure 3:

- [1] CPU Socket - Depending on brand and generation, a motherboard has a specific slot for housing the computer's CPU.
- [2] Memory Slots – Depending on the generation, allows the installation of, for example, DDR5 *(Double Data Rate - Generation 5)* RAM DIMMS *(Dual In-Line Memory Module)*
- [3] PCI-E *(Peripheral Component Interconnect – Express)* – Interface Standard used for high-speed internal component interconnections, such as GPUs, Network Cards, or High-Performance SSD Drives (using the M.2 *(M dot Two)* Standard).
- [4] SATA *(Serial Advanced Technology Attachment)* – Used for hard drive data transfer communications (CHope, 2021).

Many standards were also lost to newer and better suited options. Some examples of deprecated standards are:

- AGP *(Accelerated Graphics Port)* – Dedicated communications interface for Graphics Cards, was deprecated in favor of the more standard PCI-E standard
- IDE *(Integrated Drive Electronics)* – Once present in every computer, along with the ATA *(Advanced Technology Attachment)* was used for communications between the motherboard and the Hard Drive Disks. This standard was deprecated in favor of the newer SATA standard, and more recently, the high-performance PCI-E based M.2 connection standard, which replaces the older and less popular *mSATA (Mini Serial Advanced Technology Attachment)* (CHope, 2021).

INTRODUCTION TO THE BINARY LANGUAGE

The current electronic, silicon-based processors (CPUs) work based on switching its transistors states: a transistor is like a gate: when current flows through an open gate, the CPU considers it as being a “1”, and if current does not pass it’s a “0”: with this simple duology of operation being commonly known as the “Binary Language”. The smallest data size a computer can store is a “Bit”: a 0 (zero) or 1 (one). Further explanation on how the CPU itself works will follow in the next section (University of Rhode Island, 2005) (Simon, 2019).

The transistors of a CPU, by having only two physical states, open or closed, required a specific language to be worked upon, and this is where the Binary Language is applied.

The binary language is a Base-2 number system, and was invented by Gottfried Leibniz in 1689, many centuries before the first transistor-based computer was developed. This numbering system, contrary to the traditional decimal, base-10 system, uses only two digits: 0 (zero) and 1 (one) (Britannica Dictionary, 2022).

The reason the Binary Language is used in today’s computer systems is derived from the binary’s language simplicity and proximity to how the transistors themselves work, giving a perfect, simple way of representing a transistor state (Awati, 2019). The capacity a computer possesses in both memory and drive space is also measured based on how many bytes it can store (Awati, 2019).

A common mistake is to confuse “Bits” with “Bytes”: 1 Byte of information is constituted by 8 bits, meaning a number constituted by 8 “zeros” and “ones” originated in the binary Language (for example, the letter “A” can be converted to “01000001“, the binary representation of “A” in the ASCII table) (Transcend Corp., 2022).

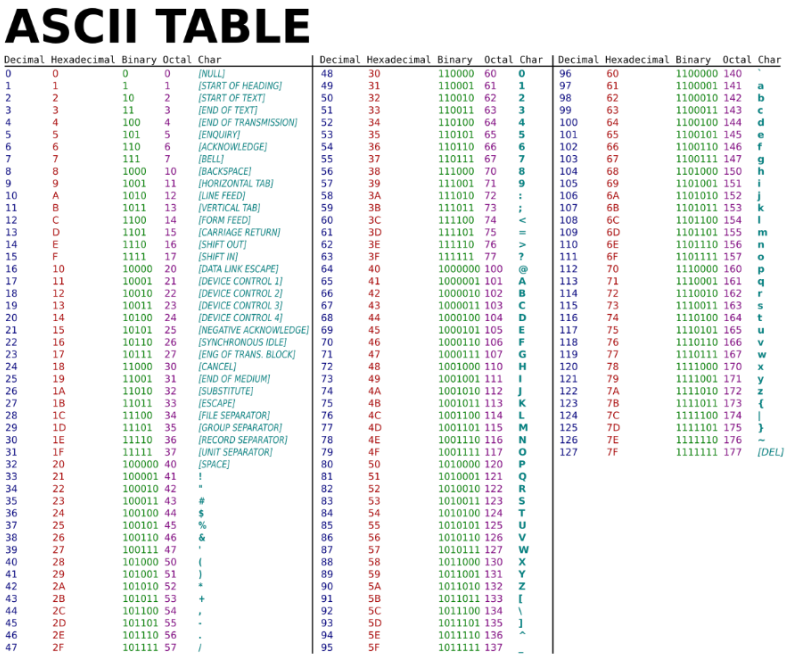
Another very common mistake induced by the manufacturers is the real capacity of a component: a 500 GB hard drive disk contains only 465.66 GB of capacity: this real capacity is often referred to as GiBs “Gibibytes”. The reason behind this difference is that manufacturers round 1 KB, which is 1024 bytes, to 1000 bytes, with the error growing exponentially with how much more bytes a component can store (Transcend Corp., 2022).

1TiB Representation (Real Capacity)	1TiB Representation (Rounded Capacity)
8 796 093 022 208 Bits	N/A
1 099 511 627 776 Bytes	N/A
1 099 511 627.776 KB (KiloBytes)	1 073 741 824 KiB (Kibibytes)
1 099 511.63 MB (MegaBytes)	1 048 576 MiB (MebiBytes)
1 099.51163 GB (GigaBytes)	1 024 GibiBytes
1.09951163 TB (TeraBytes)	1 TiB (TebiByte)

Table 1 - Conversion of 1 Terabyte of information to Bits, Rounded capacity, and real capacity.

To facilitate the comprehension on these differences, let us look at table 1, we can observe 1 Terabyte of information represented in the various methods of capacity measurements. Speed of transferring data is usually not measured in "Bytes per Second" but instead in "Bits per Second", meaning, a 1 Gbps (Gigabit per second) connection speed should be roughly equivalent to 125 MiBps *(MebiBytes per Second)*, or 0.125 GiBps *(GibiBytes per Second).*

Another very good example of how computers interpret data and how this data translates into characters or decimal numbers, is by observing how the ASCII (American Standard Code for Information Exchange) code table works.



ASCII TABLE

Decimal	Hexadecimal	Binary	Octal	Char
0	0	0	0	[NULL]
1	1	1	1	[START OF HEADING]
2	2	10	2	[START OF TEXT]
3	3	11	3	[END OF TEXT]
4	4	100	4	[END OF TRANSMISSION]
5	5	101	5	[ENQUIRY]
6	6	110	6	[ACKNOWLEDGE]
7	7	111	7	[BELL]
8	8	1000	10	[BACKSPACE]
9	9	1001	11	[HORIZONTAL TAB]
10	A	1010	12	[LINE FEED]
11	B	1011	13	[VERTICAL TAB]
12	C	1100	14	[FORM FEED]
13	D	1101	15	[CARRIAGE RETURN]
14	E	1110	16	[SHIFT OUT]
15	F	1111	17	[SHIFT IN]
16	10	10000	20	[DATA LINK ESCAPE]
17	11	10001	21	[DEVICE CONTROL 1]
18	12	10010	22	[DEVICE CONTROL 2]
19	13	10011	23	[DEVICE CONTROL 3]
20	14	10100	24	[DEVICE CONTROL 4]
21	15	10101	25	[NEGATIVE ACKNOWLEDGE]
22	16	10110	26	[SYNCHRONOUS IDLE]
23	17	10111	27	[ENG OF TRANS. BLOCK]
24	18	11000	30	[CANCEL]
25	19	11001	31	[END OF MEDIUM]
26	1A	11010	32	[SUBSTITUTE]
27	1B	11011	33	[ESCAPE]
28	1C	11100	34	[FILE SEPARATOR]
29	1D	11101	35	[GROUP SEPARATOR]
30	1E	11110	36	[RECORD SEPARATOR]
31	1F	11111	37	[UNIT SEPARATOR]
32	20	100000	40	[SPACE]
33	21	100001	41	!
34	22	100010	42	"
35	23	100011	43	#
36	24	100100	44	$
37	25	100101	45	%
38	26	100110	46	&
39	27	100111	47	'
40	28	101000	50	(
41	29	101001	51	)
42	2A	101010	52	*
43	2B	101011	53	+
44	2C	101100	54	,
45	2D	101101	55	-
46	2E	101110	56	.
47	2F	101111	57	/
48	30	110000	60	0
49	31	110001	61	1
50	32	110010	62	2
51	33	110011	63	3
52	34	110100	64	4
53	35	110101	65	5
54	36	110110	66	6
55	37	110111	67	7
56	38	111000	70	8
57	39	111001	71	9
58	3A	111010	72	:
59	3B	111011	73	;
60	3C	111100	74	<
61	3D	111101	75	=
62	3E	111110	76	>
63	3F	111111	77	?
64	40	1000000	100	@
65	41	1000001	101	A
66	42	1000010	102	B
67	43	1000011	103	C
68	44	1000100	104	D
69	45	1000101	105	E
70	46	1000110	106	F
71	47	1000111	107	G
72	48	1001000	110	H
73	49	1001001	111	I
74	4A	1001010	112	J
75	4B	1001011	113	K
76	4C	1001100	114	L
77	4D	1001101	115	M
78	4E	1001110	116	N
79	4F	1001111	117	O
80	50	1010000	120	P
81	51	1010001	121	Q
82	52	1010010	122	R
83	53	1010011	123	S
84	54	1010100	124	T
85	55	1010101	125	U
86	56	1010110	126	V
87	57	1010111	127	W
88	58	1011000	130	X
89	59	1011001	131	Y
90	5A	1011010	132	Z
91	5B	1011011	133	[
92	5C	1011100	134	\
93	5D	1011101	135	]
94	5E	1011110	136	^
95	5F	1011111	137	_
96	60	1100000	140	`
97	61	1100001	141	a
98	62	1100010	142	b
99	63	1100011	143	c
100	64	1100100	144	d
101	65	1100101	145	e
102	66	1100110	146	f
103	67	1100111	147	g
104	68	1101000	150	h
105	69	1101001	151	i
106	6A	1101010	152	j
107	6B	1101011	153	k
108	6C	1101100	154	l
109	6D	1101101	155	m
110	6E	1101110	156	n
111	6F	1101111	157	o
112	70	1110000	160	p
113	71	1110001	161	q
114	72	1110010	162	r
115	73	1110011	163	s
116	74	1110100	164	t
117	75	1110101	165	u
118	76	1110110	166	v
119	77	1110111	167	w
120	78	1111000	170	x
121	79	1111001	171	y
122	7A	1111010	172	z
123	7B	1111011	173	{
124	7C	1111100	174	|
125	7D	1111101	175	}
126	7E	1111110	176	~
127	7F	1111111	177	[DEL]

Figure 4 - ASCI Table (Wikimedia, 2022) [1]

[1] Simplified ASCII table, provided by Wikipedia, Wikimedia Foundation, Creative Common Rights Reserved.

Represented in figure 4 in the previous page, we can observe how specific actions and characters a computer uses are translated across multiple mathematical based systems, and how it translates to human actions and / or characters.

Other types of encoding tables are used, providing more characters and actions, with the most popular at the time of writing being UTF-8 (8-Bit Unicode Transformation Format), a newer, more extensive character encoding table system which allows a much broader symbology representation in binary (Mozilla Developers, 2022).

CPU (CENTRAL PROCESSING UNIT)

Having a basic understanding of how a computer operates allows us to focus on the core piece hardware component responsible for processing most software calculations.

Modern day CPUs have billions of transistors, and by switching the state of each one of them, allows a CPU to execute and perform various tasks and calculations (IBM Corporation, 2022).

From a simplified standpoint, a CPU is constituted by a CU *(Control Unit)*, ALU *(Arithmetic & Logic Unit)* and a MMU *(Memory Management Unit)* and operates by processing input instructions and outputting the results (Both, 2020).

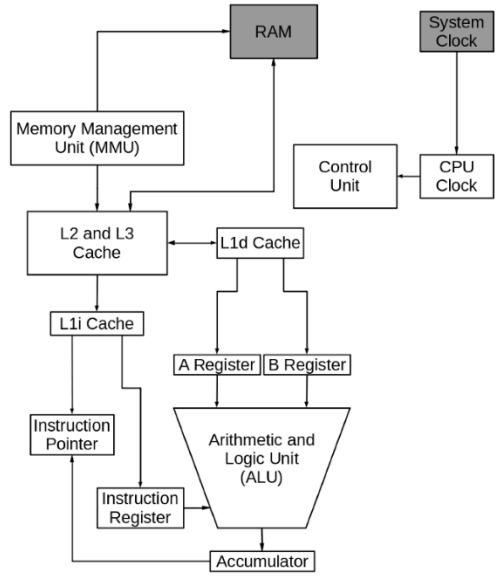


Figure 5 - Simplified Conceptual diagram of a CPU (Both, 2020)

Using the diagram present in figure 5 to help visualize the process, we start at the ALU: this part of the CPU is responsible for performing arithmetic and logical functions in a computer, such as adding or dividing numbers. The ALU works by being fed with data from two registers,

A and B, alongside with instruction data, which tells the ALU what to do with the input date, with the result of the operation being sent to the accumulator (Both, 2020).

The Instruction Register and Pointer specifies the location in memory containing the next instruction to be executed by the CPU, with this new instruction being loaded into the instruction register from the memory location into the instruction pointer (Both, 2020).

Another important aspect of the processor is its cache memory. Cache memory is an extremely fast yet small memory that resides very close to the CPU itself (the "CPU DIE *(Integrated Circuit)*", usually in square or rectangle shape, this piece contains the entire integrated circuit of the CPU). Although the CPU's cache memory is very small, having only few Megabytes worth of capacity, due to its extremely physical proximity to the CPU and fast read/write speeds, enables the processor to be fed with data to process and store temporary in-progress data calculations physically closer, removing the delays resulting from the CPU waiting for the data flow to and arrive from the main system memory (RAM) (Both, 2020).

To access the main system memory, the CPU uses the MMU *(Memory Management Unit)* to access the contents of the system memory. This component of the CPU used to reside in the motherboard itself as an independent chip in older processor generations, having been moved inside the CPU itself in modern CPU architectures (Both, 2020).

System memory is very important, as all ongoing tasks are loaded and worked on by the CPU directly in this memory. Modern computers often have at least 8 GB of system memory, allowing for smooth multitasking by the user (Yee, 2021).

All system components remain synchronized and working properly thanks to the CU (Control Unit) and CPU clock speed, allowing the components to work in a compatible rate allowing data to flow through the system properly (Both, 2020).

There are many other important aspects a CPU contains: modern CPUs have more than one core. with this core is nothing more than the processor described previously. Having multiple cores allows a computer to handle more tasks concurrently, thus allowing multitasking or parallel calculations to be performed by its multiple processors. This can have a huge impact in operation conclusion times, especially in heavy work tasks, such as mathematical calculations, video rendering or image manipulation (IBM Corporation, 2022) (Both, 2020).

A very important aspect of a processor is its operation frequency: in recent years, manufacturers have been pushing for higher and higher operating frequency's, but comparing two processors using this aspect only is inefficient, as two processors using two different

underlying microarchitectures can deliver very different performance results: this is due to the IPC *(Instructions Per Clock)* differences between microarchitectures, where certain processor manufacturing optimizations can result in a CPU being able to deliver higher performance in the same workloads than another CPU (Intel Corporation, 2022). This aspect is very important to keep in mind as we dive into the "Web Applications" world, where a single CPU core performance is very important for faster operations (Patrizio, 2022).

CPU MICROARHITECTURES

The capabilities of a CPU depend on its ISA *(Instruction Set Architecture).* Different ISA's are optimized and directed for very different use cases, with some being directed towards power efficiency and others to workload efficiency (ARM Corporation, 2022).

Desktop and laptop computers nowadays mostly use the "x86/x64", with the "x86" part being originally developed by Intel® in 1978 with the release of Intel®'s 8086 CPU, and the x64 (AMD64) instruction set being developed by AMD (Advanced Micro Devices) in 1999 with the release of AMD's Opteron Line of CPUs, the first to include this instruction set (Triggs, 2022) (Intel Corporation, 2017).

Smaller, more portable devices, such as Smartphones, Tablets or PDA's usually use a different ISA called ARM (Advanced RISC Machines). In contrast to the x86/x64 ISA, ARM focuses on a reduced instruction set to operate, enabling CPUs based on this ISA to be much more power efficient than x86/x64 CPUs (Triggs, 2022).

The ISA is not the only important aspect of a CPU microarchitecture, and proof of this is Intel®'s long lived 14nm fabrication process, where very measurable IPC improvements could be observed when the different generations of CPUs where locked at the same operating frequency for comparison.

The fabrication process of a CPU determines how large are the pathways between the transistors: the smaller the paths, the faster information reaches and the less electric resistance there is, allowing CPUs to not only work faster but also consume less energy (Intel Corporation, 2012).

The CPU architecture also determines the amount of memory a computer can access: The first mainstream computers used 32-bit based processors, which only allowed them to access 4,294,967,296 memory addresses (or bytes), which translate to roughly 4 GiB (Yale University, 2006).

Although the first Desktop 64-Bit compatible CPU, released by AMD in 2003, 64-bit platforms only became mainstream around 2009, and are prevalent in today's computer processors (Mashey, 2009).

INTRODUCTION TO COMPUTER SOFTWARE

A computer software is constituted by instructions code written and designed to guide the computer hardware to perform a specific task, with software being divided in two major types: system software and application software (Britannica Dictionaries, 2002) (Halwai, 2021).

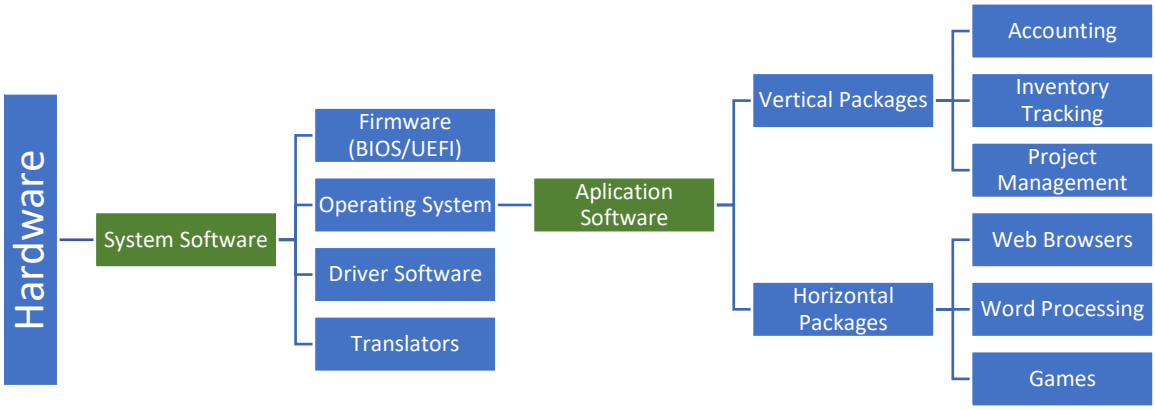


Figure 6 - Visualization of Hardware to Application Software hierarchy

Following the scheme observed in figure 6, the computer's operation stack begins at the hardware, which operates on the previously introduced binary language.

System Software is the base software which allows a computer to operate its hardware. This type of software can be further divided in two subsets: firmware software and operating system software (Halwai, 2021).

Firmware software is low-level code written to initialize a computer, such as the BIOS *(Basic Input-Output System)* software. BIOS software is code written specifically to initialize a motherboard's computer hardware and is executed when a computer is first turned on. This software was present in most older computer systems, having been replaced with the newer UEFI *(Unified Extensible Firmware Interface)* standard, which improves on various aspects, such as allowing larger HDDs to be booted from (Halwai, 2021).

The BIOS/UEFI is present in the computer's motherboard itself and is not the only firmware code the hardware relies on to function: most hardware components have their own set of written code instructions to help it operate, such as the GPU, Network cards or RAID cards firmware (Halwai, 2021).

These pieces of low-level software code help the computer's OS *(Operating System)* communicate with the hardware. The OS is part of the System Software stack, as this software is used as a base to help support all other programs to be executed. The OS has the function of serving as being a bridge between the hardware and application software, while providing the later with simplified ways of interacting with the hardware. Complementing the OS, Driver Software is also installed and helps the OS better understand the capabilities of the computer hardware, allowing for more complex operations to be executed by application software (Halwai, 2021).

All software is written on a specific programming language and needs to be translated into binary-code. This process is commonly called the "translation" software which refers to the compiler / interpreter or assembler software used to perform the written code translation, to be introduced further in the document (Smith, 2022).

Application Software are the programs a user typically interacts with when using a computer, be it word processing or a game. This type of programs can also be divided into two segments: Vertical Packages and Horizontal Packages (Wattanajantra, 2020).

Vertical Packages software aims to assist the user perform tasks in a specific industry segment, providing the user with tools to help, for example, manage a restaurant, guide a construction project, keep track of the processes a hospital has, such as inventory or appointments (Wattanajantra, 2020).

INTRODUCTION TO THE SOFTWARE DEVELOPMENT CYCLE

With the selected tools, it is time to begin applying them towards the objective of creating a prototype CRUD platform creator. Before beginning, it's crucial we understand where each of the technologies selected, with some tools often being used in multiple phases of the conception.

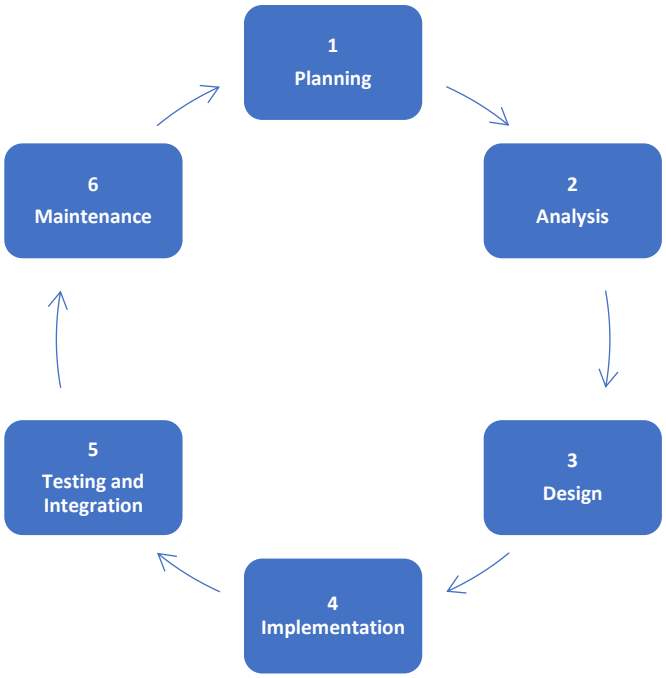


Figure 7 - The Software Development Cycle

Following figure 7, we can observe the repeating cycle a typical software project passes through and when applied to the currently being introduced concept platform, we can see phase 1 and two was already touched upon in previous topics.

Focusing on the Software Development Cycle itself, this cycle can repeat as many times as necessary to improve the software's functionality and better steer it towards the necessary objectives, and this cycle can be both applied to a personal project, a company project or a company-client participated project (Big Water Consulting, 2019) (T & S, 2013).

The typical, summarized steps this cycle consists in are:

1. Planning
    - The first phase of the project begins with analyzing an idea or necessity. This process is evaluated by all parties (client, industry experts, programmers, and any other interested parts) to get a proper understanding of what the end goal is (Jevtic, 2019) (Altvater, 2020).

2. Analysis
    - Also known as defining requirements, the idea is further processed to understand what the software needs to accomplish to achieve the goal: it is in this stage that technologies, such as programming languages, frameworks, and other system requirements are evaluated before entering the development phase (Jevtic, 2019) (Altvater, 2020).
3. Design
    - Better known for being the Design Specification phase, everything is compiled and presented to the client with emphasis on visual representations to allow less tech-literate parties to visualize and better understand the proof of concept of how the project is being intended to be designed around of. Usually more than one approach is presented to the client, and it is perfected until it is ready to be signed into a contract (Jevtic, 2019) (Altvater, 2020).
4. Implementation
    - This is effectively the Development Phase: the project leaves the paper and starts gaining form: the front-end designs and code are created alongside all the logic and programming code in the back end. Further requirements for the project to work are also prepared and tested, such as the hardware requirements (be in on-site or provided by cloud services) and supporting software are prepared to receive the application being developed for internal testing purposes (Jevtic, 2019) (Altvater, 2020).
5. Testing and Integration
    - This phase usually blends slightly with the previous one, where after the project reaches a certain maturity, testing beings to address potential problems from growing and delaying the development cycle. In this phase the software is fully tested for defects, inadequacies, inefficiencies, and any other type of anomaly that might affect its normal usage, with the results being reported back to the front-end/ back-end developers to be addressed (Jevtic, 2019) (Altvater, 2020).
6. Deployment and Maintenance
    - Once everything is tested, and fully validated by the client and QA teams, the software reaches the “Production Phase” and can now be implemented for the client to use. After starting the production phase many projects begin a new cycle, where suggestions and feedback received by the client is received,

analyzed and, following all necessary steps introduced before culminates into another development cycle (Jevtic, 2019) (Altvater, 2020).

This process, although simplified in these steps, is very complex, and requires teams to coordinate and organize their work using scheduling techniques, such as the previously referred SCRUM methodology, where specific tasks are split, assigned, and executed in smaller time-frame windows (SCRUM.org, 2022).

In a medium sized company, a small group is responsible for specific tasks, with these groups being coordinated by a Project Manager to help them interconnect and remain synchronized. Each group has their specific area, and a typical configuration usually has the following teams:

- Support for the hardware / base software necessities
- Developing the Back-End Logic
- Developing the Front-End Design and Logic
- Compile the progress and presenting it to the client in regular meetings
- Testing the alpha milestones and client milestone releases for errors
- Coordinating all teams with one another (Burak, 2022).

The entire process is very complex and requires many hours from each and every one of the forementioned teams: for small projects a lot of these phases can seem insignificant, which is why most times in small companies, where a Full-Stack Developer works, most tasks fall directly to the person programming the software, but as the project complexity scales it becomes important to divide tasks and allow certain persons to deal with specific tasks to achieve the end results not only faster but with a higher quality product (Khanra, 2016) (Burak, 2022).

INTRODUCTION TO PROGRAMMING LANGUAGES

Computer Software is written by using a programming language, with this language being later translated into binary-code the hardware can understand (Laseur, 2021).

A programming language is a means of communication between the developer (the creator of the software) and the hardware, serving as communication bridge between the two with the help of a translator, and works very similarly to how humans communicate: a Portuguese person can understand the Portuguese language and to communicate with a person who only speaks English a translation process needs to be performed, either by the person itself or using another person as an intermediary (Smith, 2022).

The translation process for both scenarios plays a big role in how the other person, or by contrast a computer, understands a given communication, and in computer software's case, the higher-level a language is, the harder and more complicated this process tends to be, resulting in slower compute times to perform an identical task (Smith, 2022).

The process of instructing a computer has evolved over the ears, being possible to be executed in three major categories of approaches to instruct a computer:

- Coded Languages
- Low-Code Tools/Languages
- No-Code/Codeless Tools

The difference between these three approaches relies on the necessity of the developer to learn and write a programming language (WatEletronics Corporation, 2021).

Coded Languages rely on a developer to know how to instruct a computer using written text (WatEletronics Corporation, 2021).

Low-Code Tools/Languages and No-Code/Codeless Tools rely on more visual interactions, such as visually connecting components and structuring the software interfaces using the WIMP's (Windows, Icons, Menus, Pointer) concept and design approach, applying methods such as YSIWYG *(What You See Is What You Get)* to their UI/UX *(User Interfaces / User Experience)*, working very similarly to how a word processor software works in helping a user write documents, allowing the developer to visualize the contents being created and laying out logic visually instead of in written code (Shawn, 2021) (Merrian Webster, 2022) (Lamprecht, 2022) .

HIERARCHY / GENERATIONS OF HARDWARE PROGRAMMING

Hardware programming, particularly the closer to the hardware's "Binary Code" language the developer programs in, can be a complex task, therefore programming languages are divided into various categories, separated by "abstraction layers".

These layers are categorized by generations of programming languages. These generations are based on the abstraction from the binary code a specific language provides the developer with: the higher the generation, the easier it is for developer to rationalize in more human-like approaches instead of machine-like approaches.

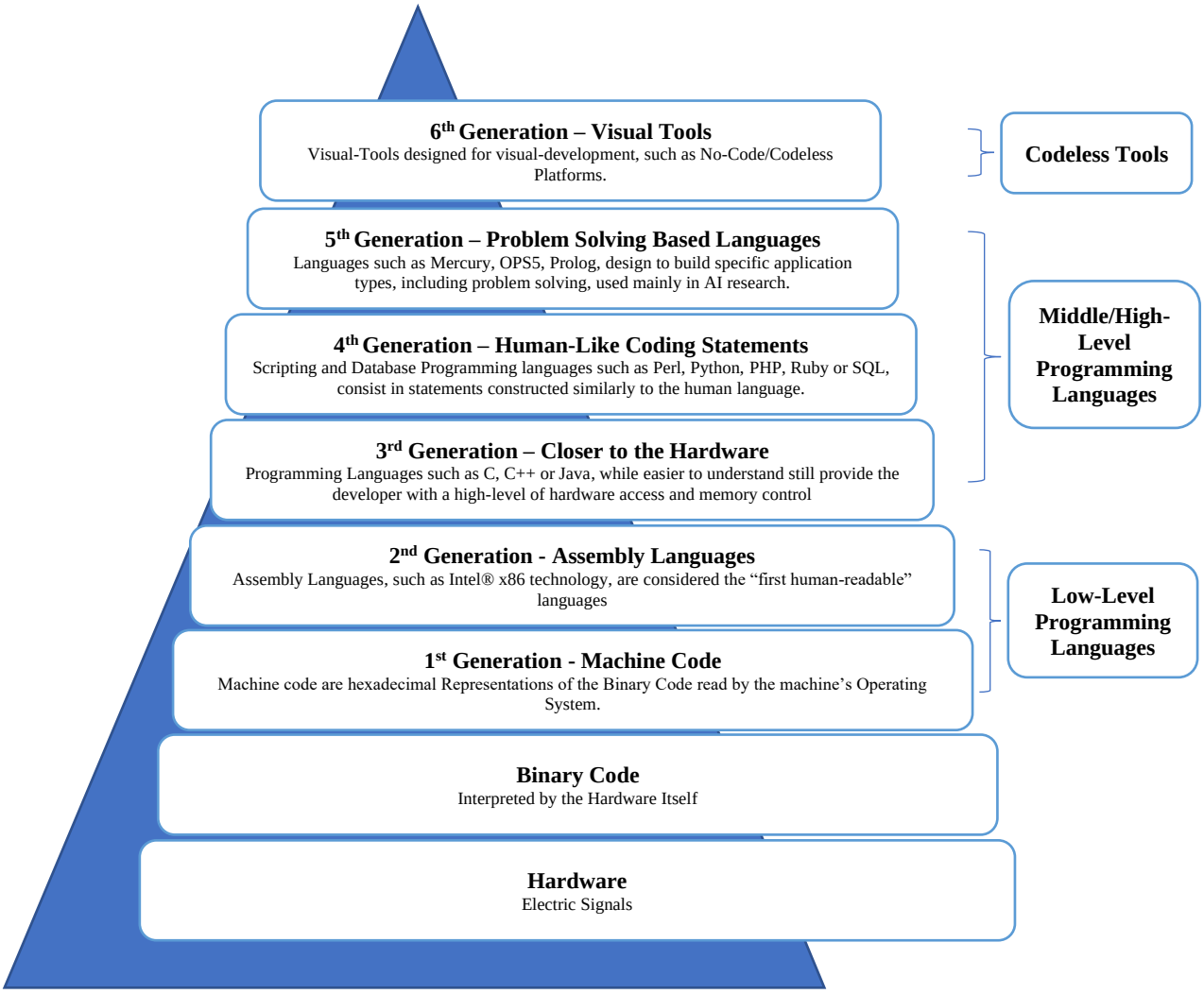


Figure 8 - Hierarchy (Generations) of Programming Languages.

In figure 8 these abstractions layers, or generations, are laid out in a pyramid-style graph. This pyramid is organized from bottom-to-top, with the bottom layer being the hardware itself, the first generation of programming languages is found in the second layer with the following layers representing each generation until the top.

The higher the level in the pyramid, the closer the written code language / programming tools usage is closer to human-ways of thinking and addressing challenges, but depending on the programming language or tools used, it has the potential to convert into slower programs due to extra instructions being introduced by the compiler (translation layer) (Moreau, 2019).

Represented at the base of the pyramid is the hardware itself and its electrical signals. These signals are generated based on the binary code the computer receives to interpret and execute the various tasks provided by the first-generation language code, the "Machine Language" (Moreau, 2019) (Verma, 2020).

The "Machine Code" is composed of hexadecimal representations, which are interpreted by the machine's Operating System itself. This code is then converted into the Binary Code the hardware can interpret. This first-generation language is considered to be the first "Human-Readable" language in the programming languages hierarchy (CHope, 2019).

The first humanized, written language, the second-generation evolution of the programming language, is the "Assembly Language" (also known for its file extension abbreviation, *.ASM) (University of Texas, 2015).

Assembly is a very low-level programming language (LLL) designed to communicate instructions with specific computer hardware and direct the flow of information and is not typically used by developers to program a machine as it is a very hands-on / close to the hardware programming language. Examples of this language are the x86 assembly language, commonly used in Desktop and Laptop computer processors made by Intel® and AMD® (Scott, 2009).

The third-generation programming languages, commonly addressed to as Middle (Medium) (MLL) and High-Level (HLL) languages, provide the developer with a more humanized way of writing code, while still providing a great level of control over the hardware's behavior and memory utilization. Programming Languages such as C and C++ are the most notorious examples of this generation of languages. Depending on the language used, third-generation languages can be considered MLL or HLL, depending on how they abstract the developer from directly managing the hardware and memory (Saylor Academy, 2022).

One of the main optimizations done to abstract the written code from the machine's logic, aside from having a and written syntax closer the human language, are removing the necessity of manually manipulating memory addresses (Monteiro, 2022).

Some of the MLL languages can also be used to write the Operating System Software itself or Hardware Drivers, with all other Application Software running on top of these (Munoz, 2022).

HLL have an even stronger abstraction from the machines code, further removing the necessity of managing memory and simplifying the overall process of writing code, but overall sits close to the MLL in terms of usability. These languages usually generate executable files, and its code needs to be pre-compiled before becoming usable (Monteiro, 2022).

Fourth-generation programming languages are intended to provide developers with a more humanized way of writing code by providing human-like ways of writing statements to be processed by the computer, also known as "scripting languages". Scripting languages code is usually interpreted and executed in Runtime (on-the-fly), with runtime meaning the code the developer wrote is executed and assisted by the programming language foundations, such as libraries, frameworks or other tools used to assist in the process of abstraction of the written code. These are usually easier to learn languages, are commonly associated with the creation of dynamic web applications or process automation scripts, such as writing code to execute a specific hands-on user task automatically (for example, mass-renaming files) (GFG, 2022).

Fifth-generation programming language designed mainly for specific problem-solving tasks, and usually is written by using constraints rather than using an algorithm written by the developer. Examples of these languages are the OPS5 and Mercury languages, with both being directed greatly towards AI (Artificial Intelligence) applications or research (GFG, 2022).

The last generation, sixth generation of programming languages is currently gaining traction, and is known for its Visual and No-Code approach of creating applications. This new generation is composed of tools designed to help developers quickly put together various types of applications, usually having task-oriented objectives, such as creating a game or creating a web page. These tools are designed to use different close-to-human ways of thinking and interacting design and concept approaches of helping a developer create the applications tools (P., 2022) (OutSystems Corporation, 2022).

Low-Code tools are not represented in either the 5^{th} nor 6^{th} generation layers, as it is more of a transitioning approach between these layers, with the difference between Low Code and No Code/Codeless tools being the fact that Low-Code tools allow the developer to write in the tool's own programming language to surpass obstacles and limitations provided by the visual aspect of the tool (OutSystems Corporation, 2022).

CATEGORIES / TYPES

A programming language can use many different types of approaches and can belong in many categories of Programming Language Types depending on how it was designed. These categories determine some of the aspects and or characteristics a specific language or tool provides the developer with.

Some significant examples of some of these types can be found bellow in the following table 3, with the most prevalent types explored in this case study highlighted in color green, or with the " * " symbol on its type name.

This table will also be very important for the next chapter, Coded Languages, where it can be used to understand the difference between some of the explored Code-Based Programming Languages.

Language Type	Description	Language Examples
Array Languages	Mathematically oriented, these languages organize operations on scalars to apply transparently to vectors, matrices and higher-dimensional arrays (Lin, 2019).	Wolfram Language, BASIC, GAUSS
Assembly Languages*	Assembly languages are equal to machine languages, although there may not be a direct 1 to 1 correlation between individual statements and instructions, making these instructions appear unreadable by humans (Scott, 2009).	x86, x86-64, ARM, MIPPs,
Authoring Languages	These languages are used by authors to create tutorials, websites and other interactive programs (IG Global, 2022).	DITA, Lasso, TUTOR, Lasso
Concatenative Languages	These languages are print-free, meaning all expressions denote functions and the juxtapositions of expressions denotes function composition. This replaces the function application, common to all other programming styles, with function composition as the default way of building subroutines (Concatenative.org, 2022).	Factor, Forth, Joy, PostScript
Constraint Languages	Declarative type language where relations between variables ar expressed as constraints, where execution proceeds by attempting to find values for the variables which satisfy all declared constraints (Jilani, 2019).	Claire, CHIP, ECLiPSe
Command-Line Languages*	Also called "Batch languages", they are used for job control and mostly for configuration purposes (Browne, 2004).	DOS, sh, Windows PowerShell, GNU bash.
Compiled Languages*	Although in reality every language needs to be compiled or interpreted, these types of languages are usually pre-processed using a compiler which converts the code into an executable file and accompanying files (such as DLLs (Dynamic Link Library) files for the Microsoft programming languages) (Smith, 2022).	C#, C++, Java, Swift, Visual basic, Rust
Concurrent Languages	These languages have the ability of allowing different parts or units of a program, algorithm, or problem to be executed out-of-order or in partial order, without affecting the outcome (Rutgers School of Sciences, 2012).	Java, Rust, Ada

Curly-Bracket Languages*	Languages which use curly brackets to construct blocks of code (Browne, 2004).	C#, C++, PHP, Java, JavaScript, Swift
Dataflow Languages	Used for reacting to discrete events and process streams of data, usually with a visual representation of the flow of data (Sousa, 2021).	Analytica, Hartmann pipelines, G (from LabVIEW)
Data-Oriented Languages*	Used mainly for manipulating relations between data table entities	SQL, Wolfram Language, Visual FoxPro
Decision Tables Languages	Used mostly to clarify the logic before writing a program in any language, expressed in a decision table (Metzner & Barnes, 2014).	Filetab
Declarative Languages*	Express logic of computation without describing the flow in detail, contrasting with Imperative Programming, where control is specified by serial orders (Britannica Dictionaries, 2022).	SQL, Analytica, Mercury, Wolfram Language
Embeddable Languages*	This category is divided in Server-Side, Client Side and In-Object Code Categories. Server Side languages, such as PHP, VBScript, allows for fragments of cod to be generated dynamically and disappear from it's output. Client-Side languages, such as JavaScript or VBScript or are limited by the browser's capability, but also allow for dynamic content to be generated, in case on Websites and Web Applications. In-Object Code languages, such as Python, Ruby or Lua, are languages which allow themselves to be embedded in compiled executable code (Github, 2021)	Described in-text
Educational Languages	These languages are developed for teaching and learning purposes (Github, 2021).	Pascal, Alice, Blocky, Snap!, Wolfram Language
Esoteric Languages	These languages are designed to test the boundaries of computer programing language design, mostly as a proof of concept and humorous purposes (Esolangs, 2022).	Beatnik, Befunge, Brainfuck
Extension Languages*	These languages are usually embedded into another program to prove said software with its features (GNU, 2022).	SQL, Python, Lua
Fourth-Generation Category Languages*	Designed around database systems, these languages are high-level and mostly used in commercial environments (Saylor Academy, 2022).	SQL, Visual FoxPro, SAS
Functional Languages*	Separated into "Pure" and "Impure" categories, these languages define programs and subroutines as mathematical functions. Impure languages contain imperative features. Pure languages include PureScript, Muranda, Mercury. Impure languages include C#, C++ (after version 11), Ruby, PHP, JavaScript (Britannica Dictionaries, 2022).	Described In-text
Hardware Description Languages	Specialized languages designed to describe the structure, design and operation of electronic circuits (Harris).	ABEL, Bluespec, Confluence, AHDL
Imperative Languages*	Imperative programming is a programming paradigm of software that uses statements that change a program's state. Most languages in this category belong to other categories as well (IONOS, 2020).	C#, C++, Lua, Pythoni, PHP, JavaScript
Interactive Languages*	These languages act as a shell, where expressions or statements can be entered one at a time and the result of their evaluation is displayed	BASIC, PHP, Lua, JavaScript, Python

	immediately (also known as REPL (Read, Eval, Print, Loop).) (Github, 2021)	
Interpreted Languages*	These languages are executed directly from the source-code by using an interpreter (Smith, 2022).	JavaScript, Lua, PHP, Python, Ruby
Iterative Languages*	These languages are built around or offer generators, which is a routine that can be used to control the iteration behavior of a loop (Smith, 2022).	PHP, Python, C#, Lua
Memory Management Classification Languages*	This category divides languages by how they manage the systems memory. The categories are: Garbage Collected Languages, such as C#, Java, JavaScript, PHP, Python or Ruby, possess automatic memory management, automatically allocating and deleting data in memory. Manual Memory Management, the developer can directly manipulate the systems memory. Example languages are C, C++ and Pascal. Optional Manual Memory Management, depending on the language, differences in memory management assistance may vary. Deterministic Memory Management, where the system memory can also be manipulated Examples are C, C++, Pascal, Rust Automatic Reference Counting (ARC), where memory usage is automatically tracked and managed by the application itself (MMR, 2022).	Described in-text
List-Based (LISP) Languages	Data-Structured Language, where a list or sequence is an abstract data type that represents a finite number of ordered values, where the same value may occur more than once (Britannica Dictionaries, 2022).	Arc, Joy, Scheme, Emacs Lisp
Little Languages	Languages designed for a very problem-specific domain. SQL fits this category by only needing a few keywords to manage a query in a database (Britannica Dictionaries, 2022).	SQL, Comet
Logic-Based Languages	These languages work by specifying a set of attributes a solution must have, instead of the steps necessary to obtain a solution (Britannica Dictionaries, 2022).	ALF, CLACL, Curry, ROOP
Machine Languages*	Designed to be executed directly by a computers CPU, these instruction sets are typically formulated as bit patterns represented in octal or hexadecimal manners. The most popular languages in the desktop market are Intel's x86 and AMD's x86-64 (CHope, 2019).	X86, x86-64, ARM, MIPS
Metaprogramming Languages	Languages designed to build programs which manipulate other programs (CHope, 2019).	C++, Lua, Python, Ruby
Multiparadigm Languages	Category attributed to languages which fit multiple categories.	
Numerical Analysis Languages	Technical computing focused languages (Britannica Dictionaries, 2022).	AIMMS, Analytica, Mathematica, PROSE
Non-English-Based Languages	Programming languages which use syntax with non-English words.	Portugol, Chinese Basic, Rapira
Object Oriented Languages*	These languages revolve around the concept of "objects", which can contain data and code: data in the form of fields, and code, in the form of procedures, constituted by classes and instances (Tran, 2021).	C#, C++, Java, PHP, Python, Python
Object Oriented Prototype Based Languages	In this category, the distinction between classes and instances has been removed (Tran, 2021).	JavaScript, ActionScript

Off-side Rule Languages*	These languages denote blocks of code by using indentation (Lee, 2013).	Python, Miranda, Scala
Procedural Languages	These languages based on the concept of the unit and data viewing range of an executable code statement. A procedural program is composed by one or more units or modules, either user coded or provided in a code library, with each module being composed by one or more procedures, such as a function, routine, subroutine, or method (CHope, 2019).	C, C++, COBOL, Java, Wolfram Language, Visual FoxPro, Python
Query Languages	Languages whose purpose is to query a database.	SQL
Reflective Languages	These languages provide programs with the capability of examining and modifying their high-level structure at runtime or compile-time.	Java, PHP, Python, Ruby
Scripting Languages*	A Scripting Language can have two similar, but different meanings. In the automation side, a scripting language is a language capable of automating tasks by passing commands and calling external programs. The other meaning are more complex languages with the capability of native scripting behavior, but are in fact imperative, but often called scripting languages (GNU, 2022).	JavaScript, PHP, Python, Ruby, Sh, GNU Bash
Stack-Based Languages	Using a Push.Pull stack mechanism, these languages are used for data-structures (Luerweg, 2015).	Forth, Joy, PostScript
Synchronous Languages	Optimized for programming reactive systems, often used in embedded systems (Benveniste, 1998).	Argus, Averest, Lustre, Signal
Shading Languages	Dedicated to 3D computer graphics, these languages are adapted to programming shader effects, be it for Real-Time Rendering or Offline Rendering (Alain, 2022).	AGAL (Adobe), HLSL (Microsoft's DirectX Shader Assembly Line)
System Languages	These languages are used for low-level tasks, such as memory management and task management of system software, such as the OS itself, drivers, compilers, etc (Techopedia, 2022).	C++, Odin, Zig
Transformation Languages	Languages used to translate source code produced in a specific language into another language (Smith, 2022).	ATL, AWK, MOFM2T
Visual Languages	These languages are interacted with in a visual-assisted in a multi-dimensional way (Verma, 2020).	Analytica, Blocky, Lava, Max
XML-Languages	Languages designed or based on the XML format (W3C, 2022).	Ant, XAML, XPath, XQuery

Table 2 - Most Common Categories of Programming Languages

CODED LANGUAGES

Although Low-Code and No-Code/Codeless tools are rising in both popularity and functionality capabilities, using a written language is still the most versatile way of creating software. As previously introduced, coded languages spread through various abstractions layers and are categorized in generations, but ultimately all share the common factor of having its code written. Written languages spread from the first to the fifth-generation layers, however, the most popular web development-centric languages sit in the fourth generation (Carlson, 2022).

Using table 2 introduced in the previous page, where a compilation of the various types and categories of programming languages is listed, we can understand that a specific programming language can have a multi-purpose focus and even differentiate itself from other programming languages.

WHAT IS AN IDE (INTEGRATED DEVELOPMENT ENVIOREMENT)

The process of writing in any programming language's syntax is as simple as opening any given text editor.

However, and especially depending on the PL chosen, to turn this code into a working application more steps might be necessary, and this is where an IDE can greatly help. An IDE as the name implies tries to consolidate a given Programming Languages workflow into a single software designed to help the developer begin a brand-new project (Codecademy, 2022).

An IDE can have different functionalities, but the most prominent and commonly found across the various programming languages are:

- Code Edition
    - Allows the developer to write the application's code.
- Syntax Highlight
    - Using visual cues, helps the developer distinguish between the various aspects of the written code, such as identifying variables or functions with different colors, or separating the code with indentations (white spaces).
- Auto-Complete Features
    - Allows a developer to introduce ready-to-use written code to be adapted to his needs, such as the basic structure a given PL uses for a "For" loop or for an "If" statement.

- Building Executables
    - Depending on if the language is compiled or interpreted, allows exporting a working executable file to be executed in the operating system.
- Debugging
    - Allows a developer to in a more intuitive and easier way to debug their application (Codecademy, 2022).

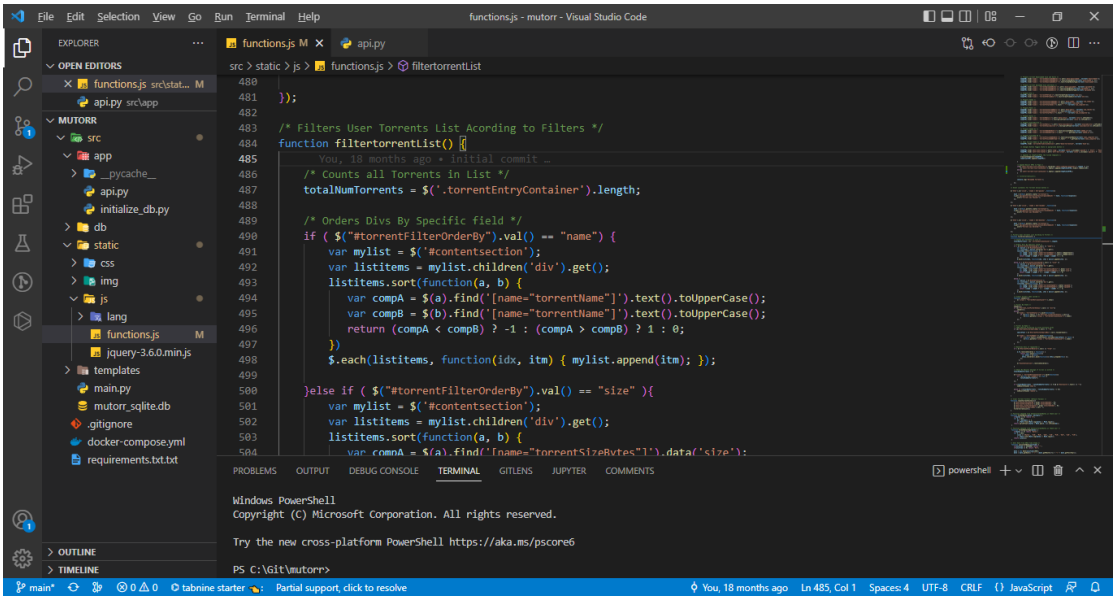


Figure 9 - Visual Studio Code using the necessary extensions to develop in both JavaScript and Python languages in a Web-Based project

IDEs themselves can have different purposes, and this can best be visualized by comparing the IDE “Android Studio” with the IDE “Visual Studio Code”, shown in figure 9.

Android Studio is an IDE designed from the ground up to create native applications for the Android Operating System, where Visual Studio Code is a more general-purpose IDE capable of creating applications not only for Android as well, but also many other platforms and technologies, including Web technologies (Microsoft Corporation, 2022) (Google Corporation, 2022).

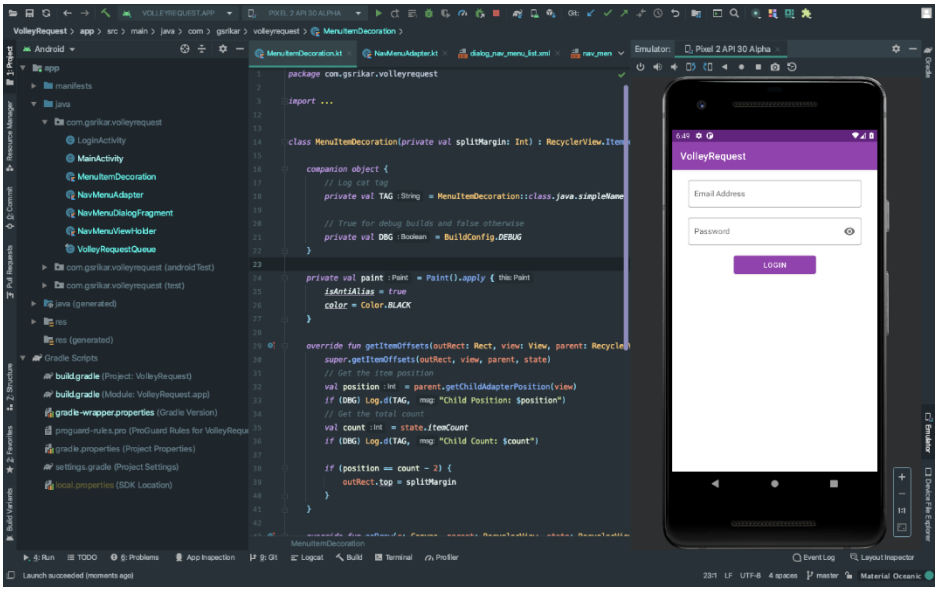


Figure 10 - Android Studio IDE

The main difference between the two IDEs is their modularity: where Android Studio comes preset from factory to work only for android applications, Visual Studio Code, thanks to its enormous extensions marketplace, can be adapted to work with many different programming language types and their common tools of assistance, providing the developer with the expected IDE functionalities introduced above (Codecademy, 2022) (Microsoft Corporation, 2022) (Google Corporation, 2022).

COMPILED VS INTERPRETED LANGUAGES

Computers can only understand their most basic code: the machine language code. For any given language to be executed by a computer, the code resulting from any given programming language must first be converted into machine code. This conversion process can be pre-executed by a compiler or be executed on-the-fly, or runtime, by an interpreter (CHope, 2019) (Smith, 2022).

A compiled language, such as C#, requires the entire of its code to be compiled into an executable to be executed in the native environment its destined to run in: this means that not only the developer's written code is translated to machine code, but all platform dependencies are also compiled and added to the resulting application to allow it to be executed in the destination operating system (Smith, 2022).

Interpreted languages, such as PHP, are similar in behavior, but differ in the aspect that the PHP code is converted into machine code and interpreted by the PHP engine at run time: this

means that if the file containing the written code is executed 2 times, but modified between runs, the resulting output can be different as the code was modified (Smith, 2022).

The biggest advantage of an interpreted language is that, to test the written code, the developer only needs to run their application, where in a compiled language, the compilation process must first be executed before being able to run the application (Smith, 2022).

Compiled Programming Language	Action	Interpreted Programming Language
Source Code is translated into machine code before execution	**Execution Process**	Source code files are translated into machine code as they are needed by the interpreter
Faster execution speeds due to being pre-compiled	**Speed**	Slower execution speeds caused by the code's on-runtime interpretation
Generated binary is platform-dependent	**Binary Output**	Generated binary code is platform-independent and is adjusted as needed by the Interpreter's native platform port.
Compilers usually detect errors and prevent binary creation.	**Error Handling**	Code is debugged by the interpreter on the fly, allowing for on-runtime interpreter errors to be caused.
Computing-Intensive software which requires heavy resource usage and optimal performance.	**Use Cases**	Less computing intensive applications or applications which need a broader cross-platform / device support.

Table 3 - Comparison between compiled and interpreted programming languages

These different approaches have their advantages and disadvantages, with the biggest difference between the two being visible in table 4 where the biggest differences between the two types of programming languages is solidified.

UNDERSTANDING A PROGRAMING LANGUAGE SYNTAX

Programming languages, aside from their distinctive syntax of writing, usually provide pre-made functionality which allows the developer to save time and avoid writing complex code to execute certain calculations. Each language comes with a preset of "helpers" designed to help the programmer execute certain actions faster: the more advanced and abstracted a language is the more tools and "helpers" it provides to be used in many different code-oriented scenarios (Moreau, 2019).

These helpers are, in their essence, functions pre-made by the language designer themselves with the objective of avoiding writing repetitive and time-consuming code.

One very good example of a helper function is the one used to find out if a certain value is in present in a variable of the type “Array”.

```
$array = ['computer', 'programming', 'is', 'fun'];
```

Figure 11 - Example of an array containing 4 elements written in PHP

In figure 10 a variable of the type “array” is declared, with this code being written in the PHP language. The variable in question is constituted by 4 elements containing a “String” word. In PHP, in order to know if a certain element is present in any given array, the programmer can use two different approaches: either iterate over the 4 elements in a “foreach” loop and checking in each iteration if the necessary value matches their criteria or he can use the PHP helper function “in_array()” (PHP, 2022).

```
#
# Iterating using a foreach
#

    $criteria = "fun";
    $result = false;

    foreach($array as $element){

    if($criteria == $element){
        $result = true;
    }
```

```
#
# Using the in_array() function
#

    $result = in_array($criteria, $array);
```

Figure 12 - Comparison of executing the same action in PHP (finding an element in an array variable) using native PHP instructions or using a language built-in function

Both previous mentioned examples are present in figure 11: on the left is represented the manual way of iterating trough the array variable and to get the desired result in the “$result” variable (a Boolean type, true or false) by iterating over every element in the array. In this same figure 11, on the right side, the same process is represented as a single-line instruction which does the exact same thing by using the “in_array()” helper function of the PHP language (PHP, 2022).

Using the correct arguments, the programmer only needs to specify the “Needle” (first argument, containing the element that’s to be searched for) and a “Haystack” (the array containing all the elements to be processed) to the function with the result being a simple “True” or “False” which specifies if the given element was found in the array (PHP, 2022).

FRAMEWORKS - EXPANDING THE PROGRAMMING LANGUAGE

Frameworks are not present in the hierarchy pyramid, but they are a very commonly used asset in project development, and if they had to be represented, they would be placed as an intermediate step in the pyramid above their base programming-language representation step .

These complement a specific language by introducing new helper methods and simplifying specific tasks, being a great way to speed up the entire process of building a software. A good Framework provides the programmer with a very solid base to build upon, allowing the programmer to focus on any given task by providing more tools to reach a desired result (RENTEC Digital, 2021).

A Computer Programmer that focuses on Web Development has many Frameworks to choose from depending on the language and technology their project uses. The process of selecting any given Framework to be used in a project will depend on the complexity and project necessities.

Using a Framework is not limited to speeding up the development process: using Web Development as an example, a commonly overlooked aspect of a project that is often overlooked is security. In the early internet days, when back-end languages such as PHP where introduced, security issues where quite common, most of these originated from lack of understanding of security principles. A very common vector of attack was SQL code injection, where websites which did not sanitize their SQL statements could allow a potential attacker to not only have access to information in the databases they shouldn't have, but also change and potentially compromise the entire project (RENTEC Digital, 2021).

Frameworks can also be used to improve on good practices and help the programmer meet the necessary security standards the modern web requires, often by implementing these practices automatically for the developer (Shukla, 2021).

To understand how a framework can overall help the process of creating a project, let us analyze one of the most currently used Frameworks across the world to build Web Based Applications: Laravel (Technical Study, 2022) (Statista, 2022).

With PHP being one of the most popular Web-Development languages in the world, Laravel, which is written in PHP and makes use of this language as its base, provides the programmer with a plethora of features and good practices, and is accompanied by a vast and very detailed documentation which consolidates every single aspect of the framework in a simple manual the programmer can use.

Some of Laravel's features include:

- Solid project file and directory structure.
- Full separation of "Frond-End" and "Back-End" code trough the templating engine "Blade", allowing for cleaner and better organized project files, as well as simpler "dynamic constructed" pages code.
- Complete ORM (Object Relational Mapping) helper called "Eloquent ORM" which allows for faster CRUD (Create-Read-Update-Delete) operations upon the database.
- Solid documentation: every aspect of the functionality provided by Laravel, including a simple, but critical, "how to get started" guide, can be found in Laravel's official home.
- Many ready-to-use components, such as the Authentication modules that can be downloaded and integrated into an app, providing a safe way to implement security-oriented elements, such as Authentication, following the best practices available (Shukla, 2021) (Laravel, 2022).

These are not the only advantages Laravel provides but is a great starting point to understanding why the usage of frameworks has grown over the years: not only it helps the programmer adhere to best-practices easier, but it also provides many projects with critical base functionality (Technical Study, 2022).

The complexity of a framework depends on the functionality it aims to provide: Laravel is one of the most complex back-end frameworks available for web development, and it can be further complemented by using other frameworks, such as Vue.js, a framework dedicated to help with the process of building user interfaces (Vue.JS, 2022).

WHAT IS THE DEBUG PROCESS

Debugging an application is the process of removing flaws, errors or anomalies in the code which cause unintended or unpredictable behavior of the resulting application.

To achieve this, and depending on the programming language chosen, several different approaches can be used to detect and correct the bug.

Compiled languages usually try and give hints of what code is causing the issue by throwing errors during compilation time. Interpreted languages work in a similar manner but at run-time.

To assist the debug process, most IDEs have debugging tools integrated. These tools provide functionalities such as Step-By-Step code analysis which allows the developer to monitor each

instruction the code is executing by observing how / what parts of the code are executed and observing the changes caused in the variables (Cocca, 2022).

ABSTRACTION LAYERS – PERFORMANCE IMPACT

The higher level the language is, typically, the higher the resource impact on any given computer hardware is to execute a specific task. To help understand the difference between some of the languages, let us observe in figure 12 a small table comparing the use of the "C" language, used as the base, to a set of other popular programming languages, and their respective impact in power usage, code execution time and system memory used.

Total					
	Energy		Time		Mb
(c) C	1.00	(c) C	1.00	(c) Pascal	1.00
(c) Rust	1.03	(c) Rust	1.04	(c) Go	1.05
(c) C++	1.34	(c) C++	1.56	(c) C	1.17
(c) Ada	1.70	(c) Ada	1.85	(c) Fortran	1.24
(v) Java	1.98	(v) Java	1.89	(c) C++	1.34
(c) Pascal	2.14	(c) Chapel	2.14	(c) Ada	1.47
(c) Chapel	2.18	(c) Go	2.83	(c) Rust	1.54
(v) Lisp	2.27	(c) Pascal	3.02	(v) Lisp	1.92
(c) Ocaml	2.40	(c) Ocaml	3.09	(c) Haskell	2.45
(c) Fortran	2.52	(v) C#	3.14	(i) PHP	2.57
(c) Swift	2.79	(v) Lisp	3.40	(c) Swift	2.71
(c) Haskell	3.10	(c) Haskell	3.55	(i) Python	2.80
(v) C#	3.14	(c) Swift	4.20	(c) Ocaml	2.82
(c) Go	3.23	(c) Fortran	4.20	(v) C#	2.85
(i) Dart	3.83	(v) F#	6.30	(i) Hack	3.34
(v) F#	4.13	(i) JavaScript	6.52	(v) Racket	3.52
(i) JavaScript	4.45	(i) Dart	6.67	(i) Ruby	3.97
(v) Racket	7.91	(v) Racket	11.27	(c) Chapel	4.00
(i) TypeScript	21.50	(i) Hack	26.99	(v) F#	4.25
(i) Hack	24.02	(i) PHP	27.64	(i) JavaScript	4.59
(i) PHP	29.30	(v) Erlang	36.71	(i) TypeScript	4.69
(v) Erlang	42.23	(i) Jruby	43.44	(v) Java	6.01
(i) Lua	45.98	(i) TypeScript	46.20	(i) Perl	6.62
(i) Jruby	46.54	(i) Ruby	59.34	(i) Lua	6.72
(i) Ruby	69.91	(i) Perl	65.79	(v) Erlang	7.20
(i) Python	75.88	(i) Python	71.90	(i) Dart	8.64
(i) Perl	79.58	(i) Lua	82.91	(i) Jruby	19.84

Figure 13 - Comparison of energy consumption, execution time and memory used by different programming languages (Cassel, 2018)

In figure 12, the "C" language is used as a baseline for the conducted tests for Energy and Time of execution, with Pascal for memory usage, as in, it's the lowest energy, time consuming and memory usage-heavy, language of the list. This is mostly because, from all the represented languages, this language sits in the hierarchy table as being a Middle Level language, the closest programming language to the previous step in the pyramid, the Assembly language. For contrast, languages such as PHP and Python, often referred to as "Scripting Languages" by the way they do not need to be pre-compiled to be executed, sit in the higher level of the hierarchy pyramid, referred to as "Scripting languages" (Cassel, 2018).

Analyzing figure 12, comparing the "C" language or the newer language "Rust" with higher-abstraction level languages such as "PHP" or "Python", the differences observed in the case study can be quite large, with, for the same task, "PHP" being over 27 times slower to process

a specific task and “Python”, another modern and very popular language, being as far as 71 times slower compared to the middle level language “C” (Cassel, 2018).

These two scripting-like languages are very popular, especially in the Web Development side of the Computer Programming world: this is mostly due to these languages being very easy to learn, especially for Computer Programmer beginners (Chawla, 2016).

A great way to further understand this difference is by comparing the written code complexity different languages, and how certain elements are simply “removed” from the code-writing element.

In figure 13 is represented a small comparison of a simple program that sums two variables (“a” and “b”) and presents the result trough the console to the user, with the Left side being written in the low-level language “Assembly” and the right side being written in the middle-level “C” language. This comparison shows just how different the syntax and control over the hardware these two programming languages are.

```
    .file   "example.c"
    .text
    .section    .rodata
.LC0:
    .string "%d"
    .text
    .globl  main
    .type   main, @function
main:
.LFB0:
    .cfi_startproc
    endbr64
    pushq   %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq    %rsp, %rbp
    .cfi_def_cfa_register 6
    subq    $16, %rsp
    movl    $2, -8(%rbp)
    movl    $3, -4(%rbp)
    movl    -8(%rbp), %edx
    movl    -4(%rbp), %eax
    addl    %edx, %eax
    movl    %eax, %esi
    leaq    .LC0(%rip), %rdi
    movl    $0, %eax
    call    printf@PLT
    movl    $0, %eax
    leave
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
.LFE0:
    .size   main, .-main
    .ident  "GCC: (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0"
    .section    .note.GNU-stack,"",@progbits
    .section    .note.gnu.property,"a"
    .align 8
    .long    1f - 0f
    .long    4f - 1f
    .long    5
0:
    .string  "GNU"
1:
    .align 8
    .long    0xc0000002
    .long    3f - 2f
2:
    .long    0x3
3:
    .align 8
4:
```

```
int main() {
    int a = 2;
    int b = 3;
    printf("%d", a+b);
}
```

Figure 14- Example of the sum of two variables in Assembly Code (left) and C Code (right).

The code from the “C” language is much more readable and comprehensive: two integer variables “a” and “b” are defined with “2” and “3” as their value, then, through a print statement, the result of the sum “a”+”b” is presented to the user.

In “Assembly” this same process is much more complex, as this language expects the programmer to specify every detail of the operation, including moving values into memory, adding them, moving the result, and finally showing the result in the console. Do note that some of the code present in the “Assembly” language side in figure 13 was created by the conversion process itself (in this case by the GCC compiler ran on an Ubuntu 20.04 Operating System).

This direct comparison of a Low-Level language to a Middle-Level language clearly illustrates the code-complexity differences. However, as we further climb the hierarchy pyramid, the differences are less drastic, but just as important.

To illustrate this, we can observe figure 13, where we can find a side-by-side comparison of the same simple program, although this time written in the “C” language the higher-level scripting language “Python”.

```
int main() {
    int a = 2;
    int b = 3;
    printf("%d", a+b);
}
```

```
a = 2
b = 3
print(a+b)
```

Figure 15 - Summing of two values in C and Python

Observing the example provided in figure 14 the differences are less drastic, but some very important differences can be highlighted. The base operation of these languages is different, with Python applying a higher level of abstraction, with the most notable differences being:

- The C language requires the user to specify each variable type (in this case, integer values) while Python does not.
- The C language needs to be ran in the main function while Python code can be executed “as is”.
- The C language code needs to be first compiled into machine code, where python is executed and interpreted “on-the-fly”.
- There are no specific code-ending marks on Python (such as the semi colon symbol on the end of each line in C), however, Python is an ident-based language, this meaning that the spacing from the start of the line in the text editor is what determines if a certain part of the code is independent or, for example, part of an “If” statement (Scaler Academy, 2022).

Another very important detail to notice is how each layer of the hierarchy pyramid allows the programmer to “forget” about certain “machine-related” specific elements, as stated in the comparison before, for example, the necessity of explicitly declaring variable types is mandatory in the “C” language and not needed in the “Python” language: the computer itself assumes this responsibility.

Other important aspects, such as allocating and freeing system memory, are also further obfuscated the higher-level the language is, allowing the programmer to focus on the final objective: writing just enough code for the machine to be able to execute the needed actions.

It is this abstraction that causes the biggest impacts in performance and energy usage found in the previously analyzed figure 14, and it is also why for a programmer having experience only in higher-level languages can hinder his possibility of being able to seamlessly transfer to work in projects coded in lower-level languages, as this requires a steeper learning curve to further comprehend some of the abstractions higher-level languages transparently provide (Scaler Academy, 2022).

LOW-CODE PLATFORMS

Low-Code Platforms are software-based ways of making another software. These tools rely mainly on a visual interface constructed in a model-driven, drag-and-drop mechanism of operation. Due to its more simplistic approach, these platforms allow novice developers to construct software without the necessity of knowing how to write code in any given coded programming language (Shawn, 2021) .

These platforms try to abstract and automate most steps of the application development cycle, which, in a typical CRUD application involve all the core-logic, database interactions, database construction and other steps.

Low-Code Platforms greatest and biggest disadvantage is that they are usually built with a target-application output type purpose: some of these platforms are designed to build games, data analysis, data CRUD, and more.

The advantage part of the specter is that by being use-case specifically designed, it allows for very inexperience users to do most of the work without requiring any programming knowledge: they just visually set up their application and program the necessary behaviors according to their needs.

This is also the disadvantage for these platforms: by being very specific in their target software development type, they can often not meet expectations or the full necessities the developer needs. This disadvantage is usually bettered by one key-aspect all low-code platforms have: they also allow the developer to write parts of their own code, allowing more knowledgeable developers to write code to execute the necessary out of scope actions he might need (Shawn, 2021) .

	Traditional Coded Languages	Low-Code Platforms
Use Cases	Usually, Multi-Purpose	Use Case-Specific Scenarios
Technical Skill	Language-Dependent knowledge necessary	No previous programming knowledge required
How the software is built	Written Code	Visual-Assisted Tools
Basic-Software Creation Speed	Slower, more hands on	Faster, mostly abstracted from programming knowledge
Versatility	Language-dependent, but suited for all programming-related challenges	Limited to the scope of pre-set actions in the low-code platform
Device / OS Compatibility	Dependent on the coded language and IDE chosen	Dependent on the Low-Code Platform's capabilities

Table 4 - Comparison between Coded Languages and Low-Code Platforms (Carlson, 2022)

In a simplified way, we can compare Coded Languages Software Development with Low-Code Platforms in table 5, where a small overview between these two types of software development methods is compared.

STATE OF THE ART – POPULAR LOW-CODE PLATFORMS

Aside from Web-based technologies, there are a plethora of different Low-Code platforms destined to produce a large variety of different project types. Without entering in finer details, let us gloss over some popular low-code platforms at the time of writing.

Do notice that most of the platforms explored have business centric purposes, providing businesses with the capability to create their own workflows in either a broader functionality support, such as the Mendix Platform or the Microsoft Power Apps tools, or a task-focused such as, with the outputting resulting application being either capable of running natively on a range of devices or directly in a web browser. Open-Source and free tools are also explored in the following overview to show existing work outside business-related necessities.

APPIAN

Appian's Low-Code platform is geared towards businesses and can output applications geared towards both Desktop and mobile form factors. The platform provides integration

capabilities with options such as DevOps and Jenkins being available, with its interface observable in figure 15.

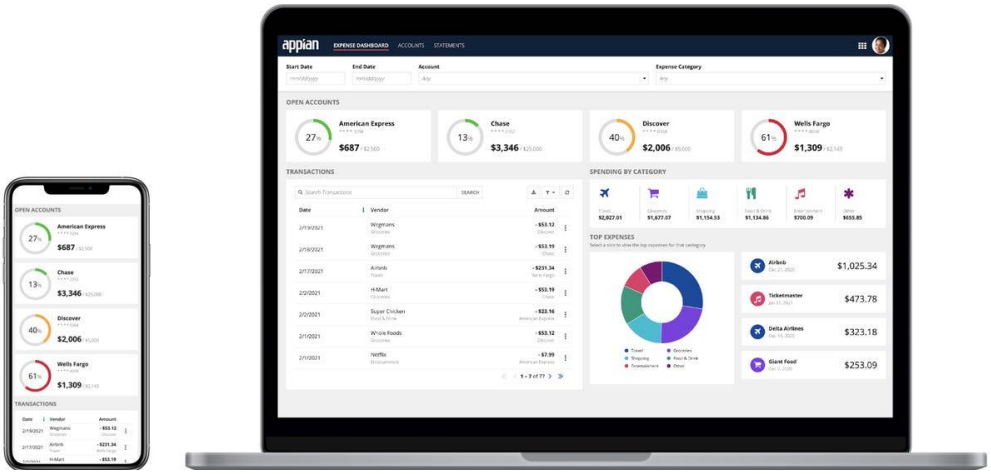


Figure 16 - output application example of the appian platform (Appian, 2022)

The platform's main objective is to provide businesses with the capability of easily create BPM (Business Process Management) applications, with a high focus on dashboard and data analysis with the objective of helping businesses organize and better optimize their business processes.

Appian also offers free-to-learn models to attract potential new customers, and the paid tiers are divided in 3 modalities, with pricing not available to the public, provided on a per-quote base (Appian, 2022).

MENDIX

Mendix focuses on a broader purpose support, providing tools which allow testing, building, and deploying applications of various specters of usability to a wide range of platforms and device form factors.

Figure 17 - Mendix Workspace, designing a dashboard page for a web environment (Mendix, 2022)

Being a versatile low-code platform capable of creating a wider range of applications through a WIMPs, using drag and drop based interface represented in figure 16, it has the capability of outputting a variety of applications, ranging from simple dashboards to web stores.

The platform offers free-to-learn to attract potential new customers, and the paid tiers range from Basic, Standard and Premium Platform Support beginning at 50€ / month for the basic plan, Standard, large scale applications starting from 800€ / month and premium support for mission critical operations on a per-quote basis (Mendix, 2022).

OUTSYSTEMS

OutSystem's Low-Code platform gives emphasis to low-code, high-performance based applications, using in-house built AI to improve processes and give a more specific, task purpose tools to businesses worldwide. Created as a general-purpose platform, OutSystem's Low-Code platform is based on processes and guides the applications development trough easy to understand use-case graphical workflows, as seen in figure 17.

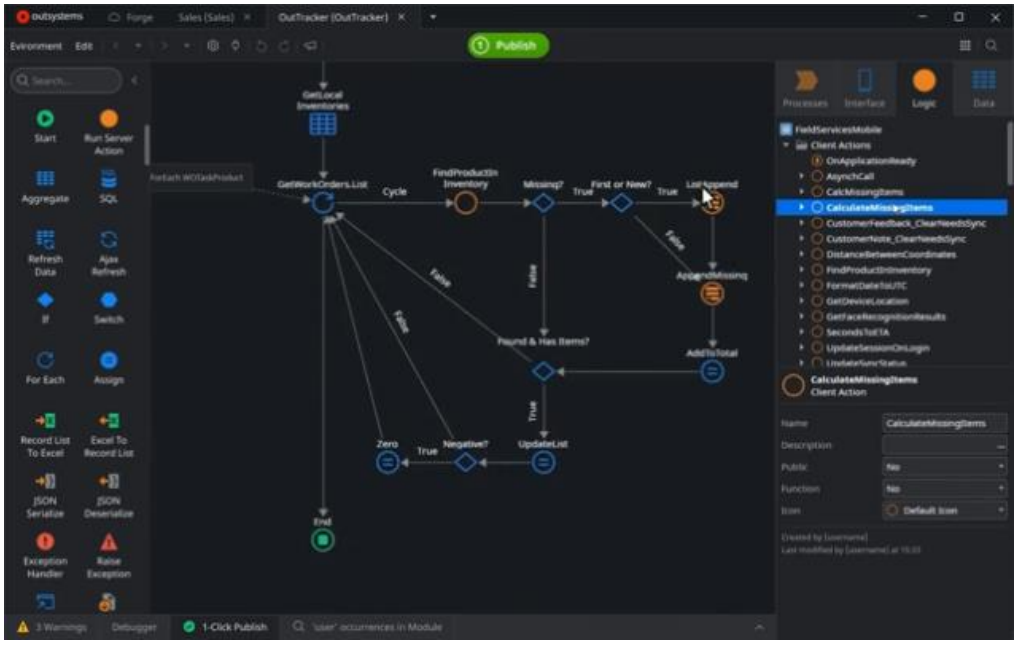


Figure 18 - OutSystem's Workflow based process (OutSystems, 2022)

The platform can output to a large variety of platforms and device form factors, and even output content to run in a web-based environment. The use cases range from simple dashboard information gathering to complete business specific management software.

OutSystems pricing differs, like other business-oriented platforms, with the difference being that the free tier can be used to create applications for up to 100 end-users, with on-line documentation available. Paid tiers start at a standard 1512.50$ US dollars per month for more advanced application support and a per-use case, quote-based Enterprise variant also available (OutSystems, 2022).

MICROSOFT POWER APPS

Microsoft Power Apps is an integrant part of the Microsoft Office 365 suite and allows businesses to create a wide range of applications, powered by all other Microsoft's tools functionalities. Its interface is WIMPs based and can be observed in figure 18.

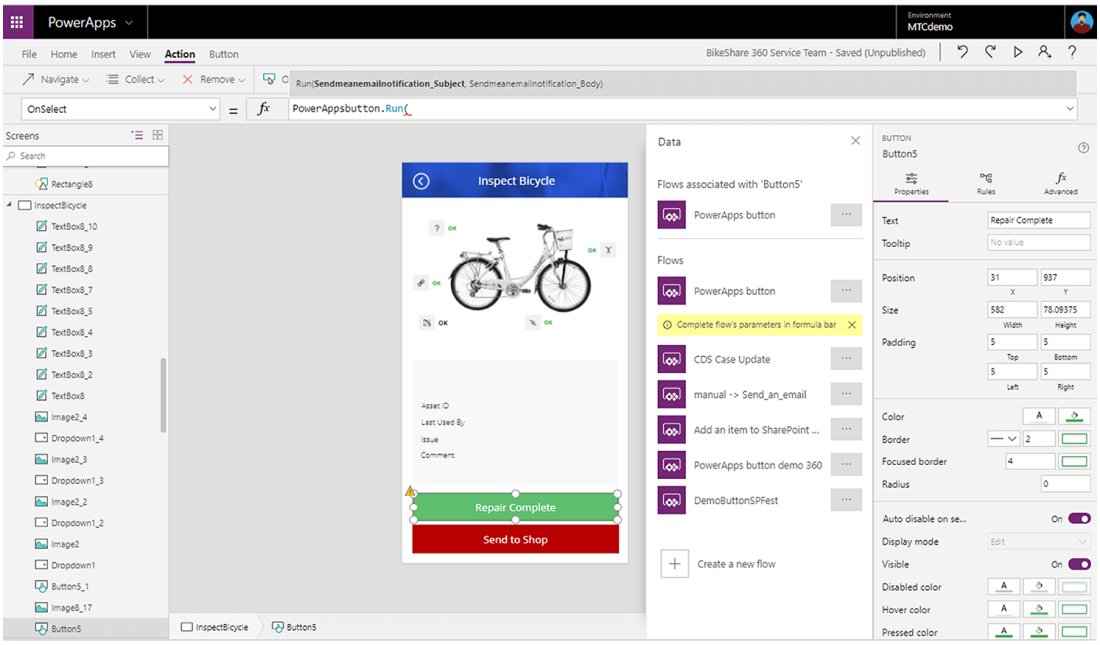


Figure 19- Microsoft Power Apps interface, designing a bicycle inspection page in a mobile view (Microsoft Corporation, 2022).

Focused on business centric needs, Microsoft Power Apps allows the user to interconnect data using over 200 connectors, allowing the creation of native applications or web-based applications tailored to the business requirements.

Integrated in the Microsoft Office 365 subscription, there is no free tier, but a 30-day trial can be used to introduce a specific user to the platform. Pricing is per-user based, and ranges from 5$ per user per month, with limited access to Microsoft's AI-Builder service, to a more premium, 20$ per month subscription unlocking the entirety of the platform's capabilities (Microsoft Corporation, 2022).

WORDPRESS

WordPress is a website creation focused low-code platform, with the main objective being providing the user with the capability of creating a simple website to share information, although having grown greatly in extra capabilities thanks to its marketplace extension store.

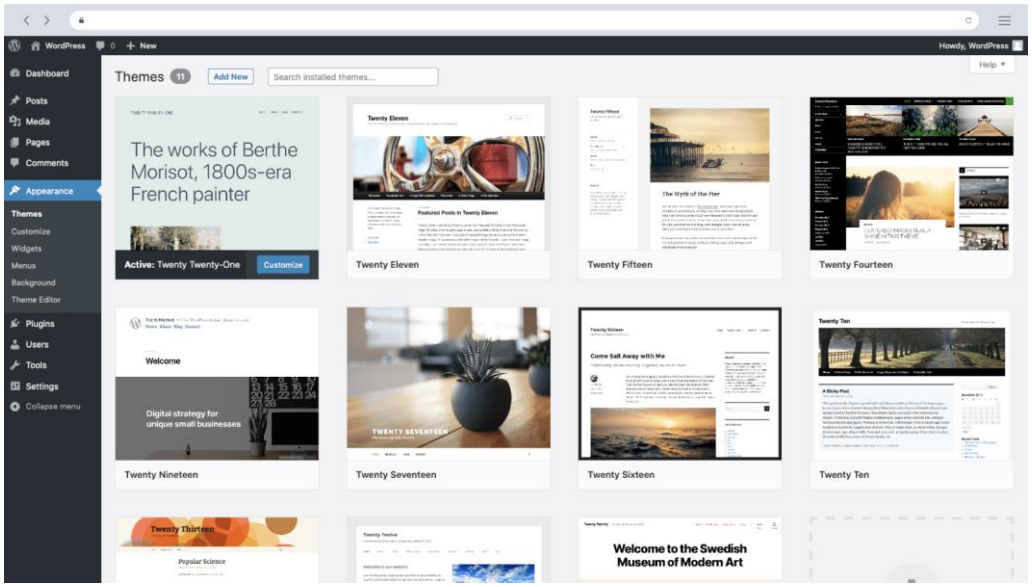


Figure 20 - WordPress Interface, in the theme selection page (WordPress, 2022)

The focus of the vanilla WordPress platform, observed in figure 19, is the creation of simpler websites, such as personal Blog, however, thanks to its many extensions available in the integrated marketplace, WordPress can do much more, such as creating web stores directly integrated with a business ERP. These extensions, themes and other add-ons are available not only in free tiers but also paid tiers, with the licensing for business use varying on a per-extension basis.

The WordPress platform is free and Open-Source, but the many extensions, themes, and add-ons available to it are available in different models, with the licensing for business use varying on a per-extension basis., ranging from free and open-source to paid tiers, with some extensions even having commercial and personal use-case limitations (WordPress, 2022).

OVERVIEW

To better help visualize some of the differences between the introduced platforms, in the following table 5 we can observe a summarized version of the contents previously introduced, with some extra data provided by third party SoftwareTesting.

	Appian	Mendix	OutSystems	Microsoft PowerApps	WordPress
Software Type	Proprietary	Proprietary	Proprietary	Proprietary	Open-Source
Business Sizes	All Sizes	All Sizes	All Sizes	All Sizes	All Sizes
Typical Use Cases	BPM	Dashboards, Internal Business Processes	Broad Support from Dashboards to fully dedicated Internal Business Processes	Internal business Process Informatization	WebSite Creation
Bundleware	None	None	None	Office 365	None
Extension Support	Yes	Yes	Yes	Yes	Yes
Multi-Platform Support	Yes	Yes	Yes	Yes	No, Web-Based Only
Allows Code Injection	Yes	Yes	Yes	Yes	Yes
Marketplace	Yes	Yes	Yes	Yes	Yes, with both free and payed options
Free Trial	Not Available	Available	Available	Available	N/A
Price	Paid, Quote-Based	50€ Base, 800€ Standard, Premium On-Quote	Starting at 1512 US Dollars and on per-quote basis	Starting at 5 US Dollars for base, 20$ for standard, per user.	Free

Table 5 - Overview of the previously introduced platforms (SoftwareTesting, 2022).

NO-CODE/CODELESS PLATFORMS

No-Code, also known as Codeless platforms, operate in a very similar way to Low-Code platforms, but main distinguishing point is that no-code platforms do not allow any additional code to be added beyond the platform's tools native capabilities (Vodafone International, 2022).

Inside these codeless platforms specter reside the most out-of-the-box programming platforms and implementation ideas, with many deriving from the WIMPs traditional method of development (Vodafone International, 2022).

These platforms, due to their no-code approach, also tend to be very task / output application type specific, often limiting the user to the platform's native capabilities and vision (Vodafone International, 2022).

Other concept approaches, such as text-to-application or sketch-to application, where an AI component generates an application based on text or image input also exist, but at the time of writing still no commercial application uses these techniques (Webflow, 2020).

Although not application-centric, but art related, text to image AIs is gaining traction, where the end-user inputs a description of the final image desired, and the AI will try to create an image based on the inputted text. This has lifted from a concept to a working state in the recent months, and even has caused already some controversy as an artist entered and won an art contest using an AI-generated image (Webflow, 2020) (Roose, 2022).

Knowing these differences between a Low-Code and a Codeless platform, let us observe some examples of fully codeless platforms.

STATE OF THE ART

In the same way most low-code platforms are oriented towards business, so are most no-code/codeless platforms existing, but this is not a rule. Of the following three examples of codeless platforms, two of the platforms are geared towards BPM, with the last one's target application type being game creation (Webflow, 2020).

Before beginning it is good to remember that, where Low-Code platforms are usually more versatile, a codeless platform are usually target software output specific in their functionality provided (Shawn, 2021).

ORACLE BPM

Oracle BPM is a complete set of tools designed for creating, executing, and optimizing business processes, giving companies a better way of collaboration between business and IT.

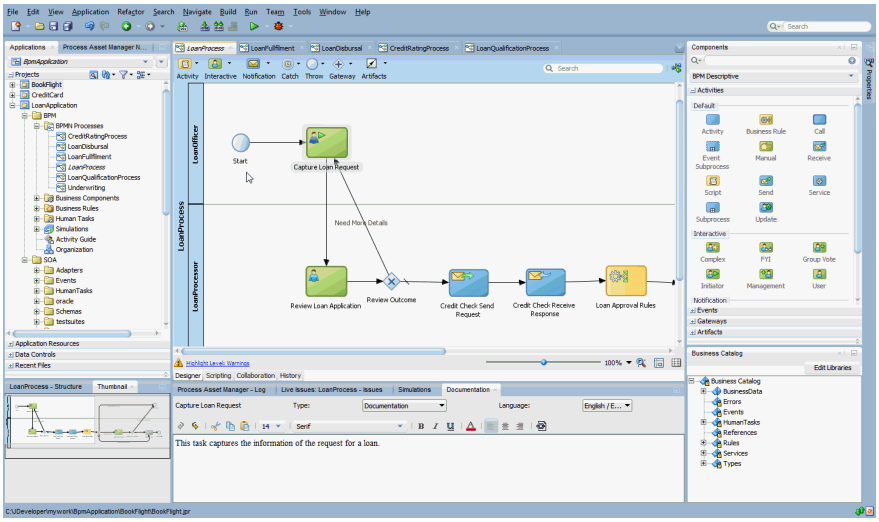


Figure 21 - Oracle BPM studio interface (Oracle Corporation, 2022)

This platform, offers a complete set of tools designed for creating, executing, and optimizing business processes, giving companies a better way of collaboration between business and IT.

The focus of the Oracle BPM platform is to enable businesses to lay down and optimize their business processes in a visual and intuitive WIMPs based interface, observed in figure 20.

Oracle BPM pricing is given on a per-quote and use-case basis, with no base price being available to the public (Oracle Corporation, 2022).

SENSUS BPM ONLINE

Sensus BPM Online is a cloud-based, web-powered, and intuitive BPM platform designed for modeling and publication, and its interface can be observed in figure 21. Using the Sensus method, a uniform methodology and clear process language, it allows business to make sure their team members are organized.

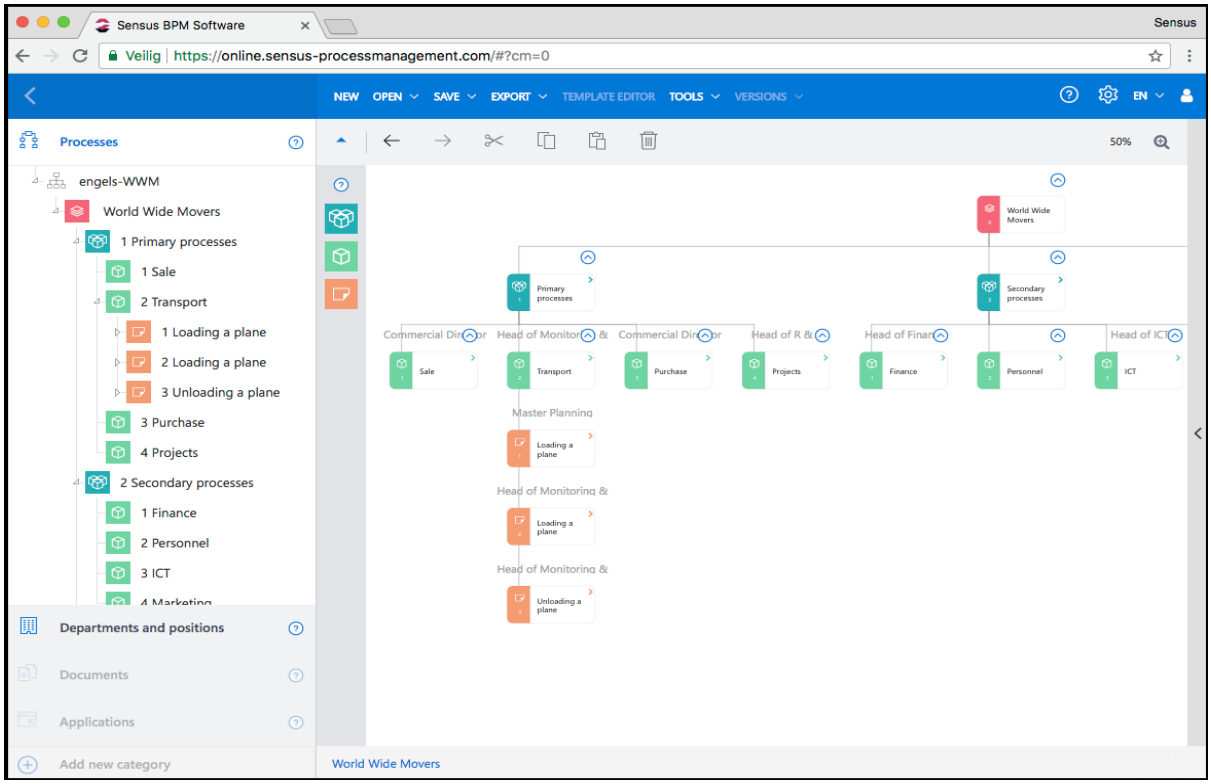


Figure 22 - Sensus BPM Software running on a web browser (Sensus Corporation, 2022)

Sensus BPM online's focus is on providing business and teams with tools geared towards allowing them to optimize their business internal processes. The tool divides itself in three sections: BPM Online, designed for business process orientation, BPM Designer, the tool that provides the user with an intuitive way of modeling their processes into visual artifacts and BPM publisher, the part of the tool which helps the user export the plans into intuitive layouts.

Online pricing is given on a per-quote and use-case basis, with no base price being available to the public (Sensus Corporation, 2022).

GDEVELOP

GDevelop is a cloud-powered codeless platform which allows for the creation of 2D, 2.5D and 3D games without writing a single line of code.

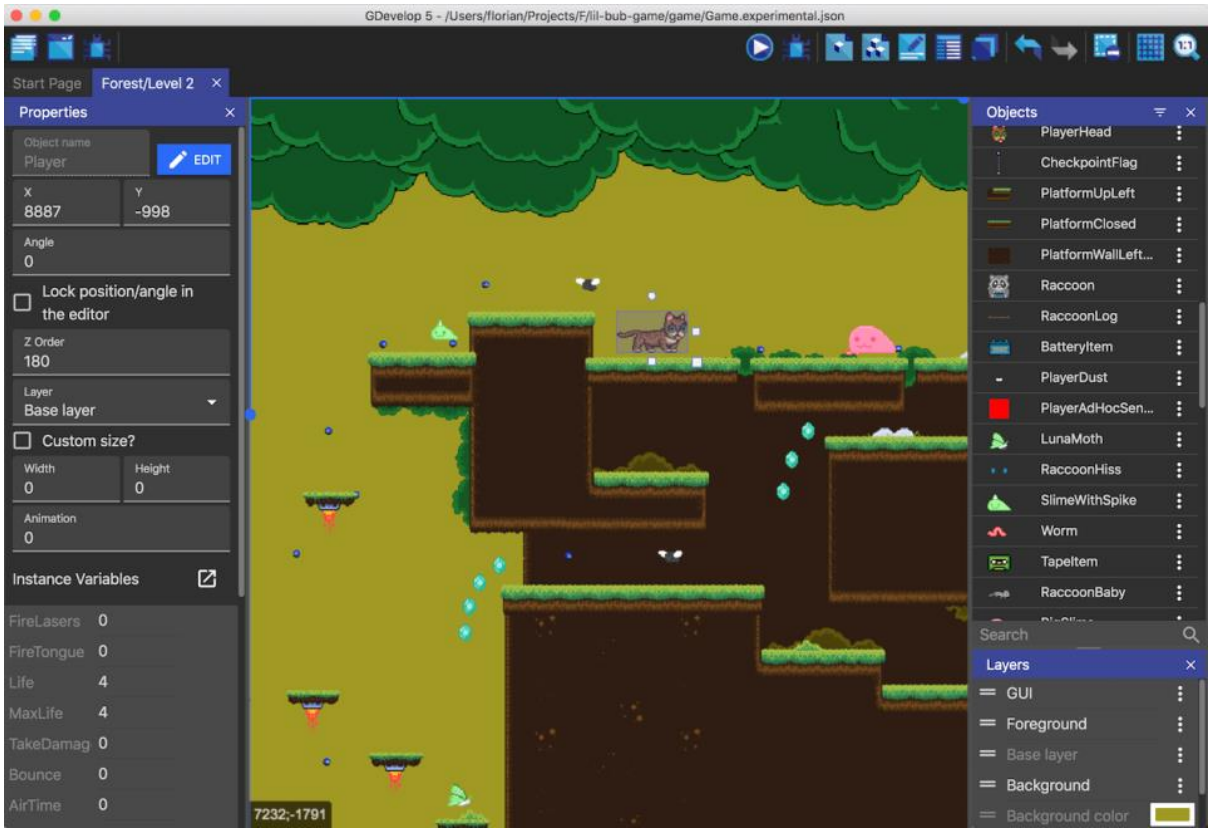


Figure 23 - GDevelop interface, designing a simple 2D platform game (GDevelop, 2022)

The application's focus is providing users with an easy-to-use engine and platform for game creation, allowing the developer to develop the game once and target it to multiple platforms and form factors at once. An example of a project being done in this platform can be observed in figure 22.

The core functionality of the software is free and open source, with some limitations. Pricing for the platforms has three tiers: free, silver and gold, with the difference being the number of projects and size of assets that can be used along with total target of platforms (GDevelop, 2022).

OVERVIEW

To better help visualize some of the differences between the introduced platforms, in the following table 6 we can observe a summarized version of the contents previously introduced, with some extra data provided by third party SoftwareTesting.

	Oracle BPM	Sensus BPM	GDEVELOP
Software Type	Proprietary	Proprietary	Proprietary
Business Sizes	All Sizes	All Sizes	All Sizes / Personal Use Available
Typical Use Cases	BPM	BPM	Game Development
Bundleware	None	None	None
Extension Support	Yes	Yes	Yes
Multi-Platform Support	Yes	Yes	Yes
Allows Code Injection	No	No	No
Marketplace	Yes	Yes	Yes
Free Trial	Available	Available	Available
Price	Paid, Quote-Based	Paid, Quote-Based	Free, ranging from 4.99€ per month per user to 9.99€ for advanced support.

Table 6 - Overview of the previously introduced platforms (SoftwareTesting, 2022)

INTRODUCTION TO WEB DEVELOPMENT

Web development is the process involved in the creation and maintenance of a website or web-based application and consists mainly in coding the server-side of the logic (back-end) and the client size of the logic (back-end) and front-end (interface) (Letendart, 2018).

A website or web-application, with some exceptions, needs a server machine running a Web Server Software and a client machine running a Browser Software to process the client-side interface rendering and logic. For this type of development, it is common to find a developer working in a Full-Stack position, where the developer is responsible for both maintaining the server-side code and the front-end code (University of Toronto, 2022).

ORIGIN AND EVOLUTION OF THE WEB

Since its origin in 1989, developed by Tim Berners-Lee in the CERN Laboratory, the Web has undergone at least 2 major evolutions, and at the time of writing, many consider the Web to be the process of transitioning to the third major evolution, the Web 3.0, the so called “Web 2.5” (CERN Laboratories, 2022) (Madurai, 2018).

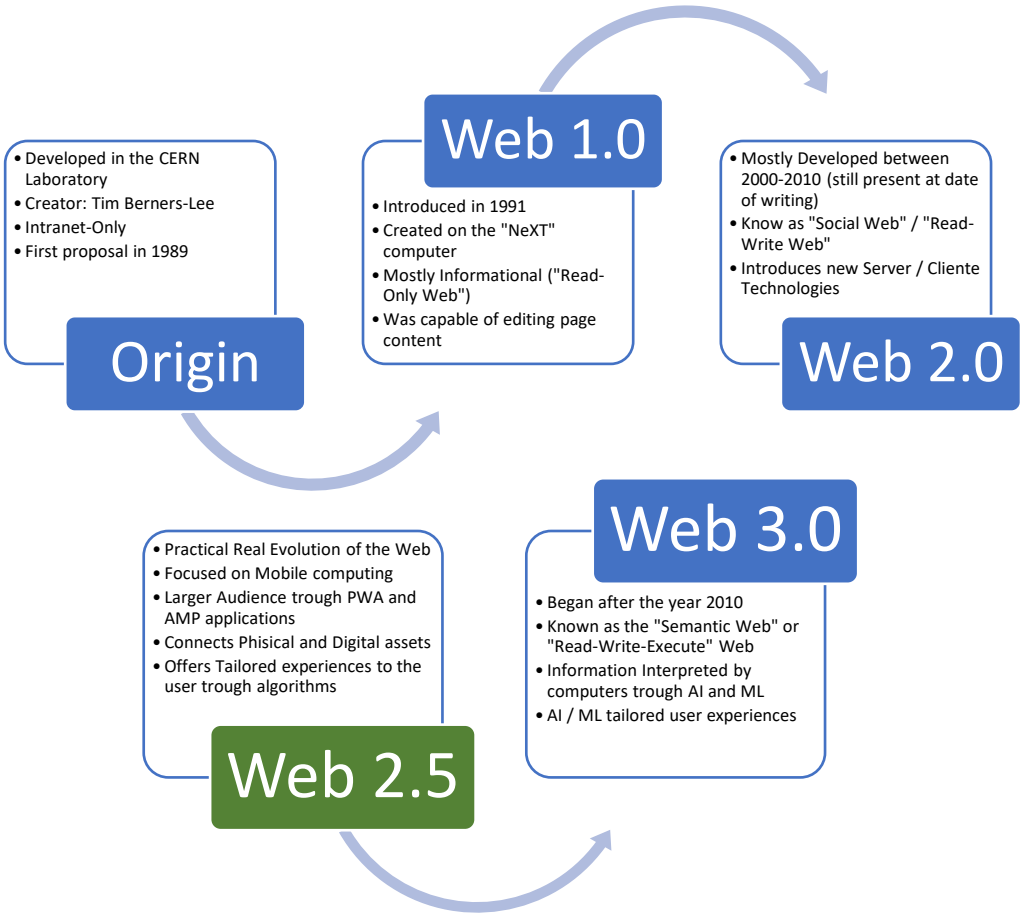


Figure 24 - Evolution of the Web, summarized

Following Figure 23, where a brief summary of the evolution of the Web can be visualized, we start at the origin: the Web began its development cycle thanks to the necessity of merging evolving technologies and data networks into an easy-to-use tool to make this information readily available, and at a global scale, and especially applicable to CERNs business model, since CERN is not an Isolated Laboratory entity, but instead is a community of scientists spread worldwide over 100 countries: providing scientists with the right tools and information was a crucial task to allow better studies and projects collaboration (CERN Laboratories, 2022).

Web 1.0 was first introduced in 1991 after being under development and tested during the year 1990, where it was running at CERNs for demonstration purposes. This first version, developed on CERN's NeXT computer, was already capable of rendering and editing server page's content, but its main purpose was informative. At this point in time only a few users had access to this network. Parallel to the graphic interface version, a "line-mode" version was also developed by Nicola Pellow, a student during her work placement at CERN, to allow other systems to access the network (CERN Laboratories, 2022).

By the year's end, the first internet-accessible webserver had come online at the Stanford Linear Accelerator Center in California, and by the year 1993 a new web-browser was released, the Mosaic web-browser was introduced, developed by Nacional Center of Supercomputing Applications, University of Illinois, allowing X-Based Window System environments to access the internet. The web quickly started growing, having over 500 websites by the end of 1993, and over 10 000 servers by 1994 years-end (CERN Laboratories, 2022) (Madurai, 2018).

By the year 2000, the paradigm "Web 2.0" (otherwise known as being the "Social Web") was introduced, with web became more personal. Pages started growing in complexity, growing from static text and image-based pages to interactive pages, allowing users to share animated images and videos in application-like platforms, such as Facebook. This was possible thanks to the introduction of newer web browsers and new client-side languages, such as HTML, CSS, JavaScript, and many others (Madurai, 2018).

It was expected that by the year 2010 the web would reach its third evolution, the Web 3.0, becoming a "Read-Write-Execute" platform, powered by AI and ML, allowing computers to interpret humans, and providing them with tailored experiences (Madurai, 2018).

By the end of the year 2022, the web still has not reached this step, and to help distinguish the web's evolution, an intermediary concept known as "Web 2.5" was introduced in an attempt to better illustrate the years 2010 and above web evolution. In this concept, the web grew to

become more mobile-focused and application-focused, allowing users to execute many tasks usually only accomplished by standalone, OS-specific applications: this allowed many platforms to be fully built using web technologies, allowing these platforms to easily being executed between vastly different devices (Madurai, 2018).

Although still at early stages, the current iteration of the web is already capable of offering a level of tailored experiences, with the most common practice being websites suggesting information based on a user's history of interest (Madurai, 2018).

CLIENT / SERVER ARQUITECTURE CONCEPT

A website or web application is operated by interconnecting two different computer machines: the server and the client.

The server machine is responsible for hosting the applications entire code base (including both the server side and client-side code) and related files and is responsible for processing all the back-end logic of the website, such as performing data calculation validations, security access levels validation and writing content to databases or storing the documents and images a specific website or application needs to operate. This machine usually runs a webserver software, capable of serving trough the HTTP or HTTPS protocols the front-end code and files to be rendered by the client machine (RFC 9110, 2022).

Depending on the technology approach, the webpage can be fully rendered server-side, where the elements on the page are changed based on the user connected to the website by the server's back-end logic itself or changed fully client-side, by the server providing the necessary codebase and information to the web browser and letting the client machine logic assemble the interface. This latter approach is becoming more widespread due to the performance benefits of offloading this logic to the client, thus making the server-side code more information/logic/validation driven, abstracting the design assembly logic to the client-side codebase (RFC 9110, 2022).

The client machine is the user's device and to access the web it needs a web browser software such as Mozilla Firefox or Google Chrome. This software is responsible for communicating with a webserver and rendering the client-side interface and logic: this is effectively what allows a user to see and interact with a website or web application (Unadkt, 2019).

The web operates on the principle of "no-trust" connections, where a developer should always assume the client-side machine can be manipulated and falsify the information sent to the server. With this principle in mind, all code logic that needs to perform validations or any other type of calculations should be strictly executed by the server machine, where a reliable and trustworthy result can be obtained. Do note that these data calculations can also be fully performed by the client-side programming language, however, since these calculations are being performed in user's machine, this means the user himself can manipulate the result output data, and without performing the same exact calculations again, the server would have no way of knowing the information was adulterated or is incorrect (RFC 9110, 2022) (Unadkt, 2019).

Taking a web banking application as an example, where a user can perform payments which directly changes their account balance, it is possible to see the need to properly dividing where logic calculations are performed: if this application updated the current user account balance based on the client-side calculations, a malicious user could easily change their current balance to any value they wanted before sending it to the server.

HTTP/S COMMUNICATION METHODS

The HTTP *(Hypertext Transfer Protocol)* protocol provides a set of rules established to allow proper communication between the client browser with a webserver. These methods can be derived from to create custom methods, but most website and web applications rely on the standard protocol methods to operate (RFC 9110, 2022).

HTTP methods are divided into Safe and Idempotent Methods:

- Safe Methods:
    - The website or application, before using any of these methods, should alert the user of the consequences of their actions. By convention, it has been established that the GET and HEAD methods should not have the significance of taking an action other than retrieval (RFC 9110, 2022).
- Idempotent Methods
    - Methods can also have the property of idempotence. This means that (aside from error or expiration issues) the side-effects of multiple identical requests must be the same as for a single request (for example, when updating an users e-mail address, if the request is sent twice, the result should still be the same). Examples of methods with this behavior are the GET, HEAD, PUT and DELETE methods. Other methods such as the OPTIONS and TRACE

should also not have side effects, and so are inherently idempotent. However, it is possible that a sequence of several requests is non-idempotent, even if all the methods executed in that sequence are idempotent. (A sequence is idempotent if a single execution of the entire sequence always yields a result that is not changed by a re-execution of all, or part, of that sequence). For example, a sequence is non-idempotent if its result depends on a value that is later modified in the same sequence. A sequence that never has side effects is idempotent (RFC 9110, 2022).

There are 8 major HTTP/S standard methods, each designed with a different purpose:

- OPTIONS
    - Used to describe the communication options with the destination server, allowing the client to determine the options and/or requirements associated with a resource, or the capabilities of a server, without implying a resource action or initiating a resource retrieval (RFC 9110, 2022).
- GET
    - This method is primarily used to retrieve information from the server, and this method is expected to only retrieve data, and can be performed in a full or partial manner, along with conditional statements, commonly known as filters (RFC 9110, 2022).
- HEAD
    - Similar behavior to the GET method, however, does not contain the body of the message. This is useful mostly to test Hyperlinks validity, accessibility and to check for recent modifications (RFC 9110, 2022).
- POST
    - This method's primary usage is to allow the client side to send data to the server, usually causing data-change impact on the server side and the responses from the server to this method cannot be cached and usually are not a resource identified by an URI (Uniform Resource Identifier) (RFC 9110, 2022).
- PUT
    - Having a similar behavior to the POST method, this method is used for an entity already exists on the server side and is used to perform changes to this

entity. These changes should be performed by overwriting all the entities elements (RFC 9110, 2022).

- PATCH
    - Alternative method to PUT, this method patches only some of the information on a given entity instead of fully overwriting the entities information with new data (RFC 9110, 2022).
- DELETE
    - This method is used to delete a given entity from the server, however multiple validations should be carried on the server-side to prevent data anomalies before executing the action (such as checking for database relations) (RFC 9110, 2022).
- TRACE
    - This loop-back method is used to obtain the full path the server uses to reach the given URI target. This method should not return an entity, but instead the full list of origin servers, proxies and gateway the request traveled through, and is used mostly for debugging purposes (RFC 9110, 2022).
- CONNECT
    - This method establishes a connection between the client and the server machines, keeping the connection open until it is no longer needed (RFC 9110, 2022).

SECURING THE CONNECTIONS (HTTPS)

The first HTTP connections, and even e-mail communications, where mostly established unencrypted, but security of data and data connections was also important, even before the year 2000. The first browser to use HTTPS *(Hypertext Transfer Protocol (Secure))* was the Netscape Navigator, developed by Netscape Communications, with this browser having first implemented it in the year 1994 (Heaton, 2014).

The HTTPS protocol's objective is to secure the data transferred between the server and the client machine, avoiding attacks such as information hijacking or content manipulation while the data flows from the server to the client machine (these types of attack are more commonly known as MITM *(Man-In-The-Middle)* attacks) (NIST, 2022).

The HTTPS protocol works by using the TLS *(Transport Layer Security)* protocol, which is an evolution of the previously known SSL *(Secure Sockets Layer)* protocol. This protocol secures the transmission by encrypting the information using an Asymmetric Public Key Infrastructure (Heaton, 2014).

This method of encryption relies on two the information being encrypted / decrypted by using two keys:

- Private Key
    - Key controlled by the owner of the webserver. This key is only known to the server itself and is used to decrypt information encrypted by the public key.
- Public Key
    - This key is provided by the server to every client machine that wants to establish a secure connection with it and is used by the client to encrypt the requests and data sent by the client to the server (Heaton, 2014).

The client machine also needs to generate a random private and public key to allow the server to establish a secure connection with it.

A key aspect of this two-key process is that any information encrypted by using the public keys, can only be decrypted by using the private key, which is why it's crucial this key remains private throughout all communications (Heaton, 2014).

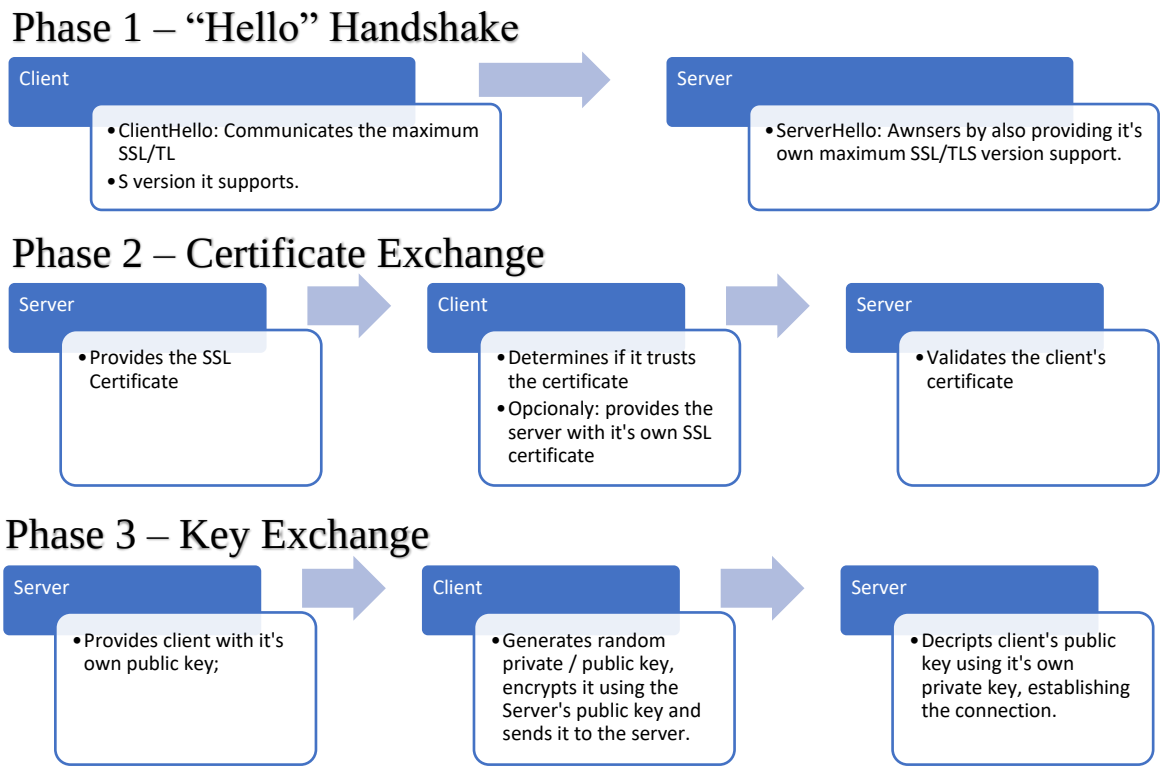


Figure 25 - Initial communication process between server and client machines (Heaton, 2014) [2]

Following figure 24, we can follow the three-phase process of how a secure connection is established between both machines.

The process begins with the "Hello" handshake, where both parties inform one another of their encryption version capabilities. After both machines agree on the version to be used, the server provides the SSL certificate which proves its identity for the client to validate. The client validates the server certificate by either implicitly validating it or by using trusted entities data to validate the certificate (Heaton, 2014).

The certificate consists of various pieces of data, such as the name of the owner, domain address and the certificate's public key (Heaton, 2014).

In some cases, the reverse process also happens, where the server also requests the client machine to prove its identity. This extra check is usually performed in very sensitive operations (Heaton, 2014).

After deciding on the version of SSL/TLS to be used, the server provides the client machine with its own public key. The client machine then generates a random private and public key

[2] Image created based on the work of Robert Heaton.

and uses the servers public key to encrypt its own public key and then sends it to the server (Heaton, 2014).

Using its own private key, the server decrypts the client machine's public key, and from this point on all communication with the client machine is performed by encrypting the data using the client machine's public key (which, in turn, decrypts the information received by the server using its own generated private key) (Heaton, 2014).

STATE OF THE ART (WEB DEVELOPTMENT)

Before creating a c platform capable of creating web applications without using any code, it is important to first understand how a Web Application works and what tools are already available. Understanding what the basic principle of a web application is and its client / server duality it is now possible to focus on each key element separately. In the following topics we will discuss the state of the art of the programming languages, frameworks, low-code and codeless tools and development tools, doing so by using as a base statistic provided by the most recent studies and case-studies performed at the time of writing.

BROWSER ENGINES

To browse the internet and render web applications, a web browser software is needed. This software is designed to communicate with a web server, send a receive data from it and interpret the code received to render the web page or application the user is trying to access (Unadkt, 2019).

The modern web browser is much more than an interpreter, and often provides the user with many quality-of-life improvements to help enhance their web browsing experience, such as saving favorite websites, printing webpages, with some browsers even providing support for custom add-ons which provides the user with even more advanced and specific tools to further assist and enhance their navigation experience (Unadkt, 2019).

At the hearth of these software lies the browser engine. This engine is the part of the software responsible for interpreting the HTML (Hypertext Markup Language) code, executing the JavaScript logic, and styling a webpage using CSS (Cascade Styling Sheets) to correctly render the visual elements of a webpage (Unadkt, 2019).

These engines, used to power web browser software, are often shared between different web browsing software, for example, Google's Google Chrome and Microsoft's Microsoft Edge are

both powered by Google's Blink Engine, where Mozilla's Firefox uses Mozilla's in-house developed engine Gecko (Unadkt, 2019).

Although different browsers can have different engines, all of them share the common factor that all of them are built with the Web Standards in mind and try to interpret these standards as closely and precisely as possible, with the objective that no matter the web application or website, a web browser can display and properly render the content seamlessly and correctly to the user. At the time of writing there are 6 engines currently being developed and updated with the most current web standards, with the market being dominated by a single web engine, Blink from Google. (Unadkt, 2019) (Statcounter, 2022).

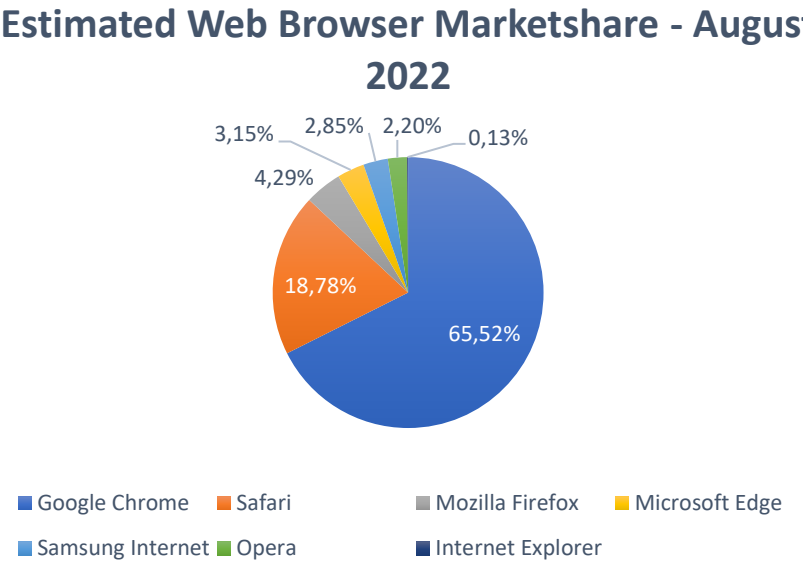


Figure 26 – Estimated Web Browser Market share – August 202 (Statcounter, 2022) 2[3]

Starting with the Web browser software market share, represented in figure 25, we can observe the current web browser market share in percentages.

As stated before, most of the market is dominated by Google's Chrome Web browser, with a staggering 65.2% market share, with the second most used, Apple's Safari, having only a 18.78% presence.

This web browser monopoly is further aggravated by the fact that some of the web browsers listed share the same Browser Engine: Blink, from Google.

[3] Graph information does not include all web browsers detailed in the case study; percentages may not add up to 100%. Statistics provided by Statcounter.

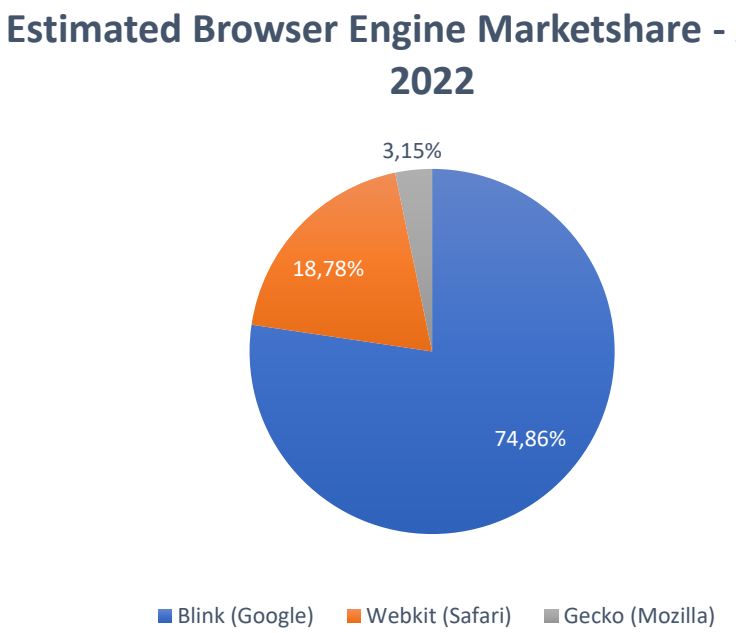


Figure 27 – Estimated Browser Engine Market share – August 2022 (Statcounter, 2022)[4]

Following figure 26, we can observe the same information present on figure 25 but directed towards the engines themselves, where we can see the Blink engine having an incredibly high slice of market share.

Having an engine browser monopoly is not ideal, as it can hamper technologies advancements, as it could be observed during the MSHTML, Internet Explorer's browser engine, era in the year 2000 through 2010, where Microsoft's Internet Explorer dominated the market and significantly hampered new standards adoptions, often causing developers issues by having to support multiple engine versions, as Internet Explorer was not updated at the same pace as today's modern browsers are.

[4] Graph information does not include all browser engines detailed in the case study; percentages may not add up to 100%. Statistics provided By Statcounter.

STATE OF THE ART – PROGRAMMING LANGUAGES

Programming a website or web application requires developing code for two different parts of the application: the server side and the client side, and although a single language, JavaScript, is capable of being used by both the server and client side as the base programming language, other more suited languages, are typically used for the server-side operations.

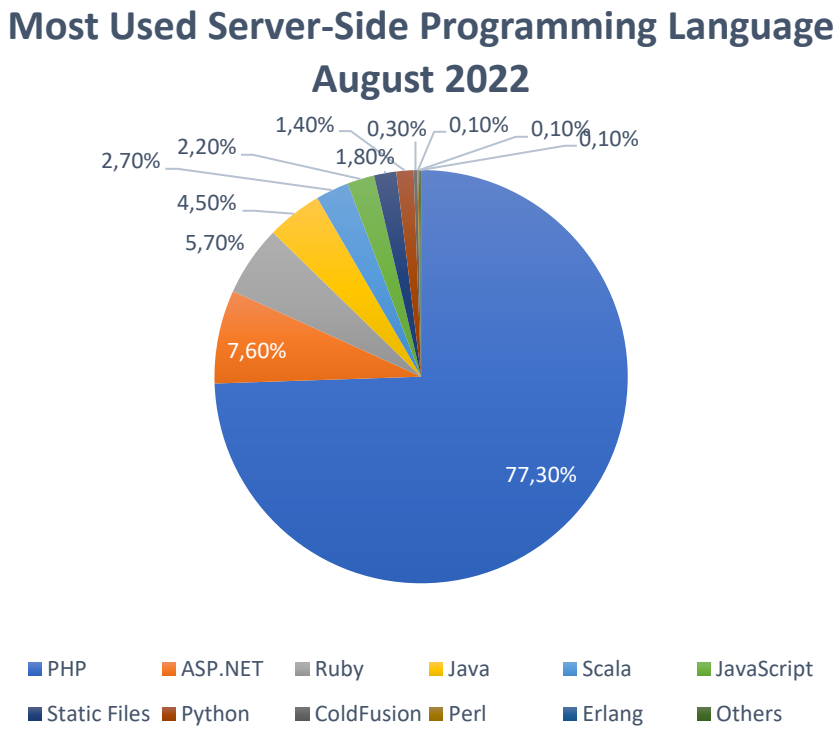


Figure 28 – Percentages of websites using various server-side programming languages, August 2022 (W3Techs, 2022).[5]

Starting with a case study related to the server-side of web programming, following the graph in figure 27, with the source study provided by W3Techs Studies, we can observe that the most popular server-side programming language is PHP, followed by Microsoft's ASP.NET.

[5] Percentages may add up to more than 100%: one website may use more than one server-side programming language. Statistics provided by W3Techs Studies.

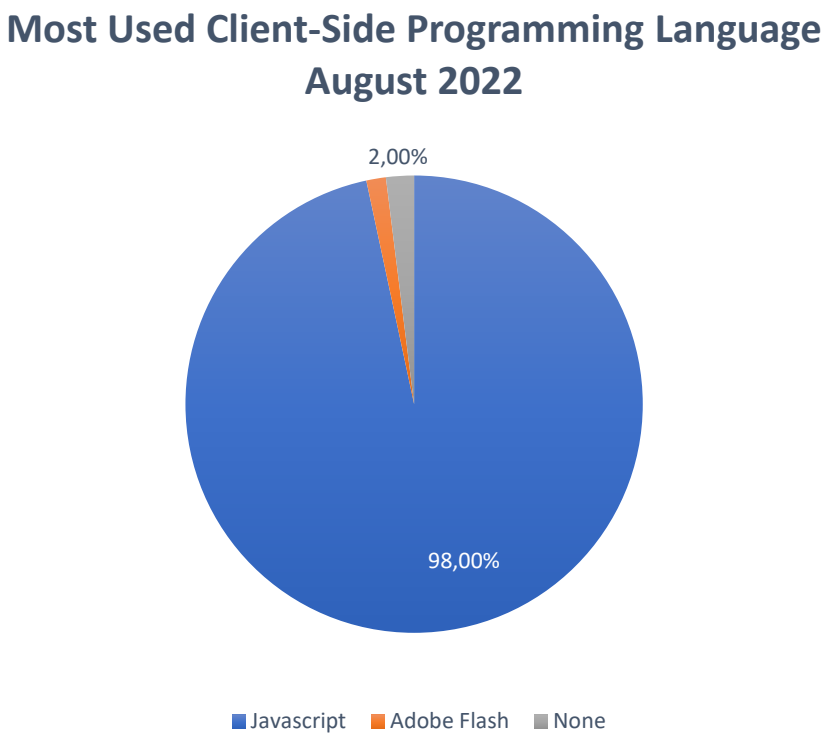


Figure 29 – Percentages of websites using various client-side programming languages, August 2022 (W3Techs, 2022).[6]

The client-side programming language market share is very different from the server-side, as observed in figure 28: currently being the only logic processing language supported by all major web browsers, JavaScript dominates the case study statistics with only a small fraction of the now-discontinued Adobe Flash remaining present in some websites.

Although less prevalent, a very small fraction of websites still consists in static-pages website which do not use any programming language in the client-side, consisting only in files written with the markup HTML and styling CSS languages.

[6] Percentages may add up to more than 100%: one website may use more than one client-side programming language. Statistics provided by W3Techs Studies.

STATE OF THE ART – FRAMEWORKS

Programming a website in any one of the native web development programming, markup and styling languages is entirely possible, but it is very time consuming for the developer. Many server-side and client-side frameworks were developed to speed up the website and web applications development, allowing developers to focus on the core functionality of their software instead of programming every single detail and functionality their website or application might need (Schäferhoff, 2022).

Web Frameworks do not all try to do the same exact thing, with the distinction being in the fact that they might be directed towards different software / websites needs: some focus towards server-side logic and operations, others focus on the client-side logic, styling and navigation, and they can derive from being developed towards a mobile environment deployment and some even try to be a one-size-fits-all solution to enable the developed website / application to more easily become cross-platform and cross-device compatible (Intelegain Technologies, 2019).

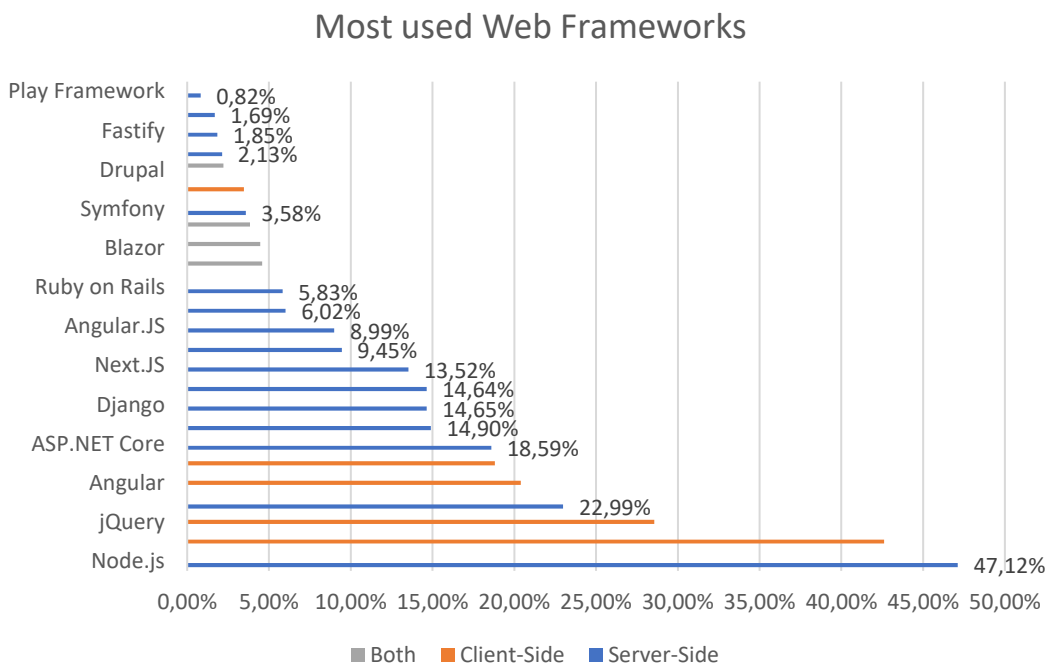


Figure 30 – Most used web frameworks among developers worldwide (Statista, 2022).[7]

Using a case study performed by Statista as a base, the graph in figure 29 displays the most popular web frameworks worldwide.

[7] Percentages may add up to more than 100%: one developer may use more than one technology for web development. Case study performed by Statista using an online form between May 11th and June 1st 2022 with a total of 58743 respondents. Provided by Statista

Although the case study combines both server-side and client-side technologies, it is possible to get a mostly accurate idea of what server-side and client-side frameworks developers are trending towards.

Analyzing the case study further, a trend to switch towards a JavaScript-based development process can be observed, with many developers stating they use the Node.JS framework.

This case study contrasts with the previous statistics in figure 27, where PHP was the leading server-side technology, therefore this case study needs to be evaluated carefully: just because a developer uses one technology, does not mean most of his production-ready projects are based on this technology.

PHP powered frameworks, such as Laravel and Symphony are present in this case study as well as ASP.NET technologies (Statista, 2022).

STATE OF THE ART – DEVELOPMENT TOOLS

Development tools are software designed to assist a developer during the coding of their software. These tools do not need to be exclusively directed towards writing code, as a website and application uses other components, such as images or videos.

There are different aspects that play a part when deciding which development tools to use, but the biggest and most important is the programming language and other technologies the developer chose to write their software in, as it is not feasible, for example, to use a platform specific tool, such as Android Studio, which is directed towards the Android OS platform, to write applications for iOS, another competing OS platform (Wallack, 2019).

In Web Development's case, if the objective is to develop a static website page, a simple notepad text-writing application will suffice a s it allows the developer to crate text-based files containing all the HTML and CSS code needed to display a page but will not provide the developer with any kind of assistance during the process (Schäferhoff, 2022).

An IDE (Integrated Development Environment) software is more than a tool to write code: a good IDE will help the developer not only write code, but will also check the code for errors, provide visual aids to differentiate various aspects of the code, such as color-coding functions, variables, constants and many other aspects of the written code to facilitate the reading process, but more advanced IDEs can even provide context-aware suggestions of how a specific framework or programming language method works, with other IDEs even integrating other

necessary components for a specific language, such as integrated compilation capabilities and advanced debugging features (Codecademy, 2022).

The IDE itself can also not be exclusively directed to work with a single programming language or framework, with Visual Studio Code being the most notorious example: this IDE can work with a variety of different languages and help develop software for multiple platforms (Codecademy, 2022).

Apart from the IDEs themselves, and focusing on Web Development, other tools can be used to assist the development of the website or application: database management software, such as Heidi SQL can be used to easily manipulate database data to see how different or unexpected data affects directly the software behavior, as well as tools for the design itself, such as CSS snippets generators, color coding tables, or even JSON and XML formatters used not only to validate the outputted result, but to visually assist by formatting the results in a more readable way (Schäferhoff, 2022) (HeidiSQL, 2022).

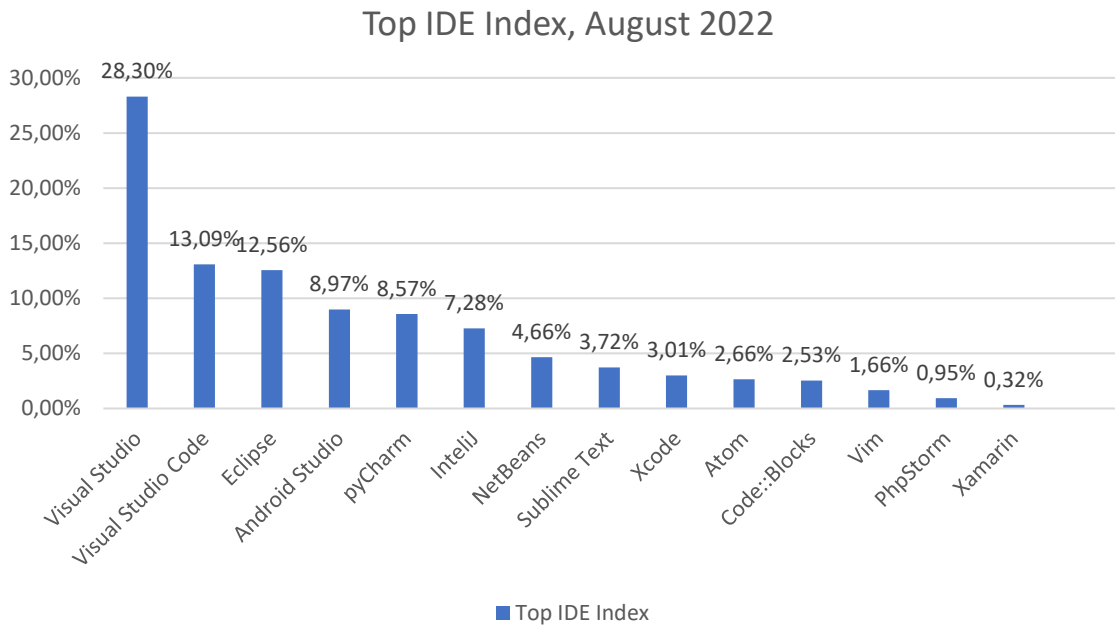


Figure 31 – Top IDE Index, August 2022 (GitHub, 2022).[8]

Analyzing figure 30 on the IDE statistics market share, the most popular IDE for development is Visual Studio, followed by Visual Studio Code, both tools created and maintained by Microsoft.

[8] Percentages may not add up to 100%: graph results are limited to the top 15 out of 31 ranked IDEs. Statistics provided by Github

Although they share similarities in their name, these IDEs are not the same, as Visual Studio is directed towards Windows OS development and Visual Studio Code is a cross-platform IDE, capable of programming in multiple languages with multiple device and OS targets.

Both IDEs are capable of web development, with Visual Studio being the tool of choice for ASP.NET programming and Visual Studio Code, although also capable of ASP.NET coding, being mostly used towards all other programming languages.

Visual Studio Code is rapidly growing in popularity, having grown over 10.8% in the last 5 years, surpassing Eclipse, another very popular C/C++ IDE, in market share recently.

This IDE is exponentially growing in popularity thanks to its modularity approach: Visual Studio Code as a barebones IDE is only capable of processing text, but thanks to its marketplace, it is possible to add support for almost every programming language: If a developer is preparing to build a website based on Laravel and Vue.JS, he only needs to go to the marketplace and install the “Laravel Extension Pack” and “Vue.JS Extension Pack” packages: these packages bring not only support to understanding the framework-specific files but also include bundled in needed extensions, such as the PHP and JavaScript extensions, which provide Visual Studio Code with the capability of understanding and providing assistance in these programming languages codding process (GitHub, 2022) (Microsoft, 2022).

APPLYING THE TECHNOLOGIES – EXAMPLE OF A WEBSITE'S CREATION PROCESS

After knowing many of the programming languages, frameworks, IDEs, and other tools that exist to create a web-based website or application, the developer must choose the path he wants to take to create the software.

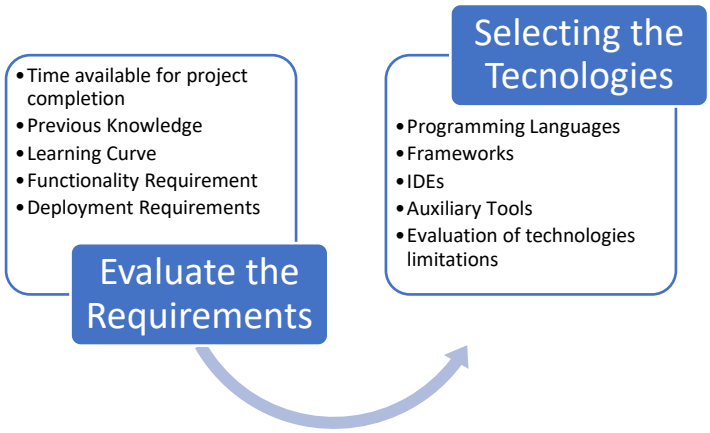


Figure 32 – Topics to consider before beginning development, summarized (Danciu, 2018).

The base starting point are the project requirements themselves, as observed simplified in figure 31: understanding the time constraints, the technologies needed to achieve the software's target functionality and even the developer's knowledge are key to the success of the software development (Danciu, 2018).

For the sake of future comparison with the process of using a codeless tool to archive the same result, as it will be developed in the next section, and to further help visualize the decision process, let us assume the objective is to create a simple CRUD (Create, Read, Update, Delete) platform with the objective of managing clients, cars, and car tour appointments.

Software Requirements
•Manage Clients •Manage Cars •Manage Appointments •Hosted in a Linux OS •Must be cross-platform compatible •Must be cross-device compatible •2 week development timeframe

Developer Skills
•Languages: •C#, PHP, Javascript, Python, SQL •Frameworks: •Laravel, Flask, Bootstrap, Vue.JS •IDE: •Visual Studio, Visual Studio Code •Software: •Laragon, Apache, Nginx, PHP Interpreter, Heidi SQL, MySQL Workbench, MariaDB, MySQL •OS: •Windows Server, Ubuntu Server

Figure 33 – Fictitious Project requirements and Developer skills

In figure 32 we observe the developer's client requirements and the developer's skills. The client needs a simple platform to manage both his client's information, cars information and client appointments to show off the cars.

The three main aspects of this request are the short time span of development time, and the contradictious request of cross-platform compatible. If the developer where to build every aspect of his application in vanilla PHP, HTML and CSS he would take a considerable bigger amount of time to achieve not only basic functionality but also the cross-device perquisite.

Knowing all the requirements and the short time frame, the developer must opt for the easiest and fastest way to create the software, and this is done by combining multiple technologies.

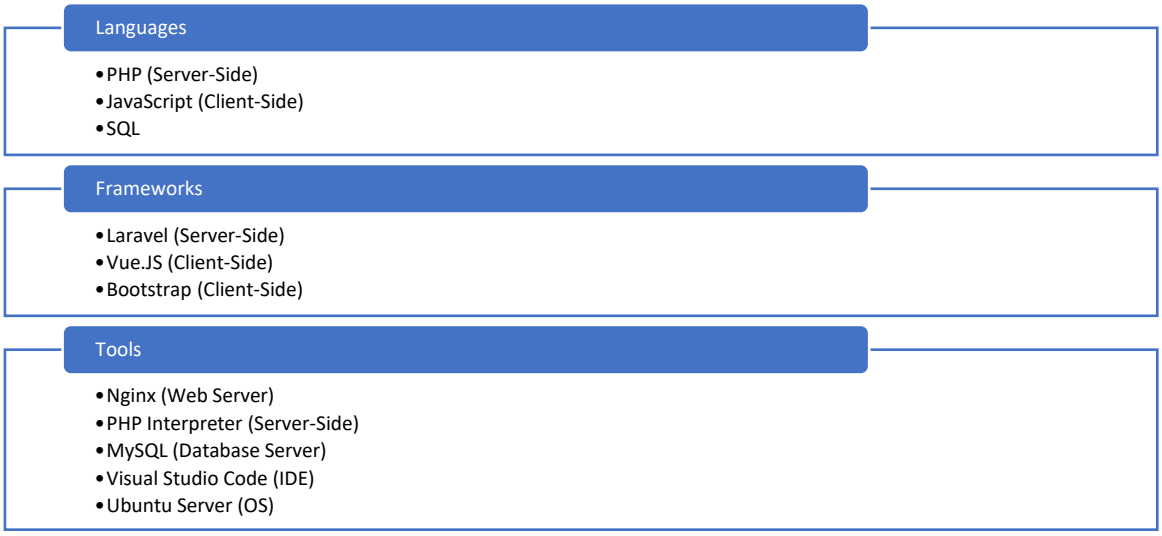


Figure 34 – Fictitious development decision by the developer

Assuming the developer has better experience with the Laravel Framework in contrast to the Flask framework, and that he has a solid knowledge of both Bootstrap and Vue.JS frameworks, the technologies the developer would use to assist the software creation are as described in figure 33.

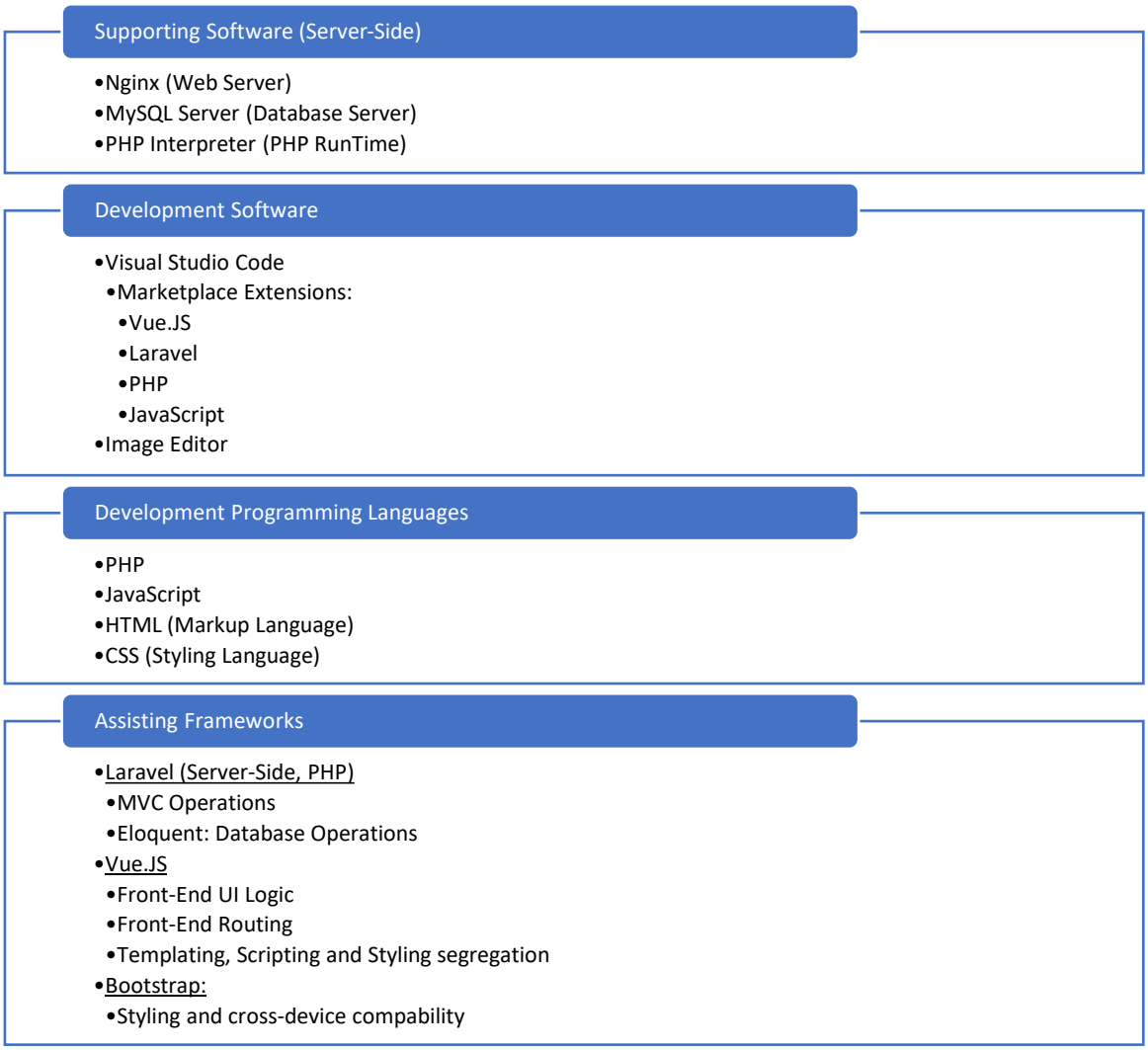


Figure 35 – Simplified overview of the fictitious project necessary software, tools, languages, and frameworks

In figure 34 we can see all these elements with a brief explanation of what each of the technologies is responsible for in this small example fictitious project.

To understand how each of the selected technologies would assist the developer, let us start on the server-side of the development. Laravel is an MVC (Model, View, Controller), and in this case will provide the developer with the separation of these aspects in the server-side. Laravel also includes Eloquent, an ORM (Object Relational Mapper) which serves the purpose of interacting with database records (Laravel, 2022).

The main operations of the server-side part of the software includes the CRUD operations in a MySQL Database and managing the validation of the inputted user data in the client side.

Laravel also includes Blade, a templating engine designed to help construct HTML pages dynamically on the server-side, however, the developer chose to use another framework for this purpose, Vue.JS, which is processed completely on the client-side, allowing the server to

perform less calculations in comparison to if the application was constructed using Laravel's blade (Laravel, 2022).

On the client-side, and regarding the user interface construction itself, the developer chose to use Vue.JS and Bootstrap as the helping frameworks. Vue.JS allows for easy logic operations separation of not only the HTML and CSS code, but also the JavaScript code.

Operations logic such as sending and retrieving a client's data information from the server can be easily separated in the **.vue* file's structure, which consists in three main sections:

- Templating Section:
  - Section holding the Vue's component HTML code.
- Scripting Section:
  - Section containing all the JavaScript code with needed methods, such as the update or delete method.
- Styling Section:
  - Section dedicated to all the CSS styling code custom made for the specific Vue component (Vue.JS, 2022).

Vue itself is composed of components, which can contain other components inside. This relation between components is referred to as the "Parent / Child" component relation, with the parent being the Vue component which contains the other (child) component inside (Vue.JS, 2022).

Having a way to properly separate all the server-side and client-side application code answers to all the project's needs, but one: being cross-device compatible.

A cross-device compatible applications means it should be capable not only of running in a different OS, but also a different form factor, such as running on a desktop computer or running on a smartphone (Asper Brothers, 2020).

A desktop and a smartphone operate very differently: the desktop relies on mouse and keyboard inputs to navigate and has a much larger screen real estate compared to the smartphone's gesture based and smaller screen, and this is where the last framework comes into action: bootstrap.

Bootstrap is by itself a framework designed to help quickly stylize and make a webpage responsive to the various device sizes and capabilities. Vue itself has a derivation of this

framework, called Bootstrap Vue, which operates in a very similar manner to the main framework (Vue.JS, 2022) (Bootstrap, 2020).

Bootstrap consists of pre-made CSS code classes and JavaScript code responsible to stylize a webpage and operates by being directly integrated into native HTML page elements trough a means of class and attributes, allowing a determined element, such as a table element to correctly respond and adapt when displayed on a desktop or a smartphone (Bootstrap, 2020).

Having decided on the technologies, the only remaining aspect to be determined is the IDE to use and the supporting server software and client software needed to execute the web application, which in this case passes by using Visual Studio Code with the needed marketplace extensions for the Laravel and Vue.JS frameworks, as well as PHP and JavaScript languages, as well as the server-side software to host the application, consisting in the Web Server Nginx, PHP Interpreter and MySQL Database Server (Microsoft, 2022).

INTRODUCTION TO PROGRESSIVE WEB APPS

The process described previously is both applicable to the creation of a website or a web application: what determines the difference between the two are the functionality, contents, and target audience with a website is typically an informative or entertaining web page, while a web application usually serves a more directed task-related purpose.

Capabilities	Web Application	Native Application
Device Compatibility	Cross-Device Compatible	Device-OS Ecosystem Specific
Offline Capabilities	Possible trough the implementation of PWA concepts	Capable
Device Resource Usage	Higher due to the use of the Browser Engine as a rendered	Lower due to the use of native OS APIs

Table 7 – Comparison overview between a Web Application and a Native Application (Mozilla Developers, 2022).

PWA (Progressive Web Apps) are the next step in the evolution of web applications, where the difference between a native application and a web application starts to be less discernible. The objective of converting a web application into a PWA consists mainly in providing the user with a more native, seamless experience for the platform he's using the application on, and this includes access to typically native-reserved functionality, such as a smartphone's accelerometer or camera in a seamless way, like the native applications written to the smartphone's OS platform, with the biggest differences listed in table 6 (Mozilla Developers, 2022).

The first big difference between a Web Application and a PWA is it's offline capabilities: raditional a web application requires both a server and a client machine, where a PWA, depending on the features and functionality it intends to provide, can be ran fully offline in the client device, with all logic and data being processed and stored in the device locally, and examples of these type of applications can be a simple word-processing application, a game or a note taking application (Mozilla Developers, 2022).

The other main difference between a web application and a PWA is its capability of being installed just like a native application in the client device.

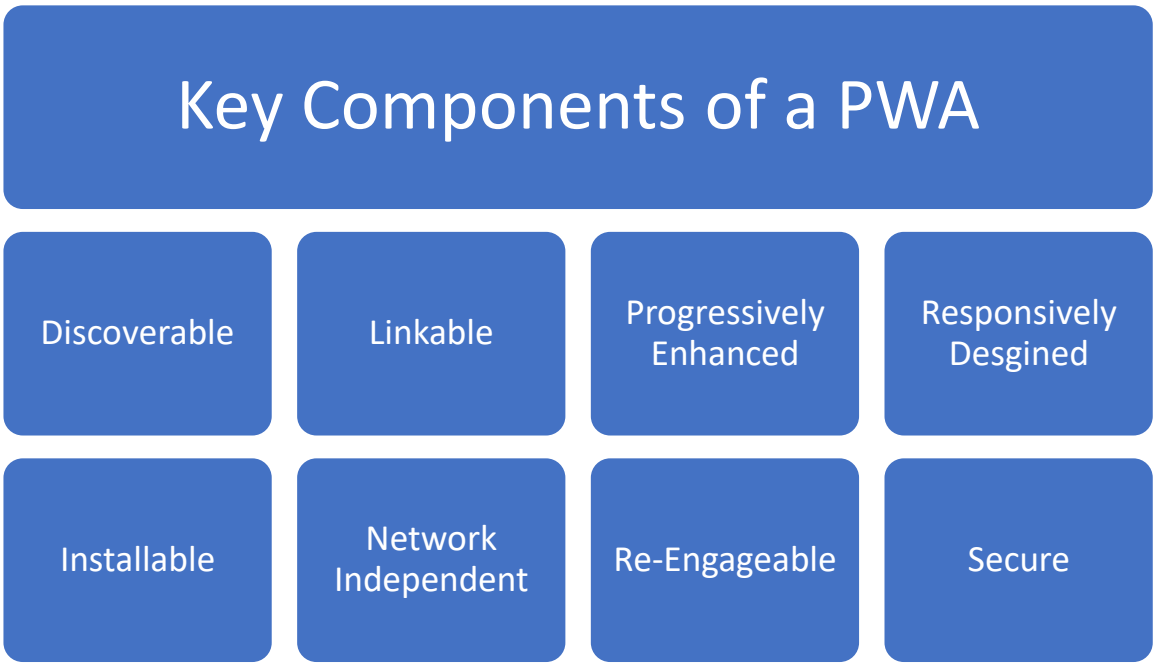


Figure 36 – The 8 key aspects of what distinguishes a Web Application from a PWA (Mozilla Developers, 2022)[9]

[9] According to Mozilla's Developers MDN Web Documentation

Following figure 35, based on official Mozilla Developer's documentation, are listed the 8 key differentiators between a normal Web Application and a PWA. To become a PWA, the application must be:

- Discoverable:
  - Its contents should be found trough search engines.
- Installable
  - Capability of being installed into the device's main menu.
- Linkable
  - Accessible using a sharable URL (Uniform Resource Locator).
- Network Independent
  - Should be capable of working, even if partly, offline or under poor network connectivity conditions.
- Progressively Enhanced
  - Should be able to be usable with a basic level of functionality in older versions of browser engines.
- Re-Engageable
  - Should be capable of sending notifications whenever new content or information is available.
- Responsively Designed
  - Should be usable in any device's form factor, ranging from Desktops, Laptops, Tablets, Smartphones, and even other browser-ready devices, such as smartwatches or smart appliances.
- Secure
  - The connections between the user, application and the main server should be designed with a security-first mindset to process any sensitive data (Mozilla Developers, 2022).

PWA COMPONENTS

To adapt a web application into a progressive web application can range from easy to difficult, depending on the application we are designing and the PWA functionality desired. There are 3 core elements that are necessary to build a PWA (Mozilla Developers, 2022):

Web App Manifest File

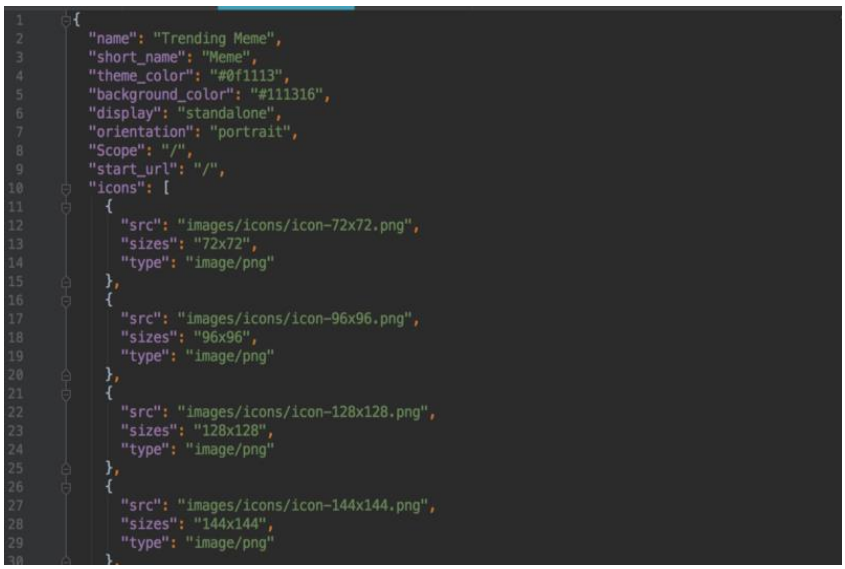


Figure 37 – Example of a Web Manifest File contents

This file, written in the JSON format as seen in figure 36, gives meta information about the web app itself, such as title, icons, color schemes, starting URL, and more. It is crucial for the user to be able to install the application and the operating system adapting itself to look more like a native application rather than a website (Mozilla Developers, 2022).

Service Workers

```
self.addEventListener('fetch', event => {
    //caching for offline viewing
    const {request} = event;
    const url = new URL(request.url);
    if(url.origin === location.origin) {
        event.respondWith(cacheData(request));
    } else {
        event.respondWith(networkFirst(request));
    }
});
async function cacheData(request) {
    const cachedResponse = await caches.match(request);
    return cachedResponse || fetch(request);
}
```

Figure 38 – Example of some service-worker functions.

Service Workers are event-driven workers which run in the background of an application and are designed to interact with the back end allowing the application to receive notifications

and new content in a timelier manner from the server. Service workers can be used for more than just notifications, being also capable of caching content offline to allow the user to view the most recent changes cached, even without an internet connection. These files are written in JavaScript as seen in figure 37 and listen primarily to events such as “install” or “fetch” as they perform tasks (Mozilla Developers, 2022).

Served through HTTPS

For a PWA to be considered a PWA, it must always be served over an HTTPS connection with a valid certificate (Mozilla Developers, 2022).

STATE OF THE ART

PWAs are not only accessible through mobile devices, but also trough desktop and other types of devices. To fully function and mimic native application’s capabilities, support on the OS level is necessary.

Although support on all major platforms is still not fully implemented, with some platforms missing some features others have (Caniuse, 2022). In the following pages we will analyze platform compatibility more thoroughly and see some examples of popular services which have already transitioned into using PWA.

PLATFORM SUPPORT

The main functionality of PWA is supported by in all modern Windows, Android, Linux and iOS/OSX based systems. The actual level of functionality depends on a per-platform and per-platform version basis, with most of the features being tied to the browser engine itself.

As a basis of comparison, let us observe the compatibility table, at the time of writing, for the “Add to Home screen” functionality support on a per-browser and per-platform scenario.

Browser / Platform	Current Version	Support Status
Chrome	109	Fully Supported
Edge	105	Fully Supported
Safari	16.1	Not Supported
Firefox	107	Not Supported
Opera	91	Not Supported
Internet Explorer	11	Not Supported
Chrome for Android	105	Fully Supported
Safari on iOS	16.1	Partially Supported

Table 8 - Support for the "Add to Home Screen" functionality of a PWA on a per-browser and platform basis (Caniuse, 2022).

As can be noted in table 7, support on the most popular web-browser, chrome, is fully implemented, where in lesser popular, more platform-specific browsers, such as Safari for the Apple ecosystem, support is either missing or only partly implemented for this functionality. The same holds true for a wide variety of other PWA functionalities, with support being gradually implemented in browsers and platform which lack support currently (Caniuse, 2022).

EXISTING PWA EXAMPLES

PWA are not exclusive to websites, but can originate all kinds of software, from purpose-specific software, such as stores, games, to general-purpose software, such as word processing software.

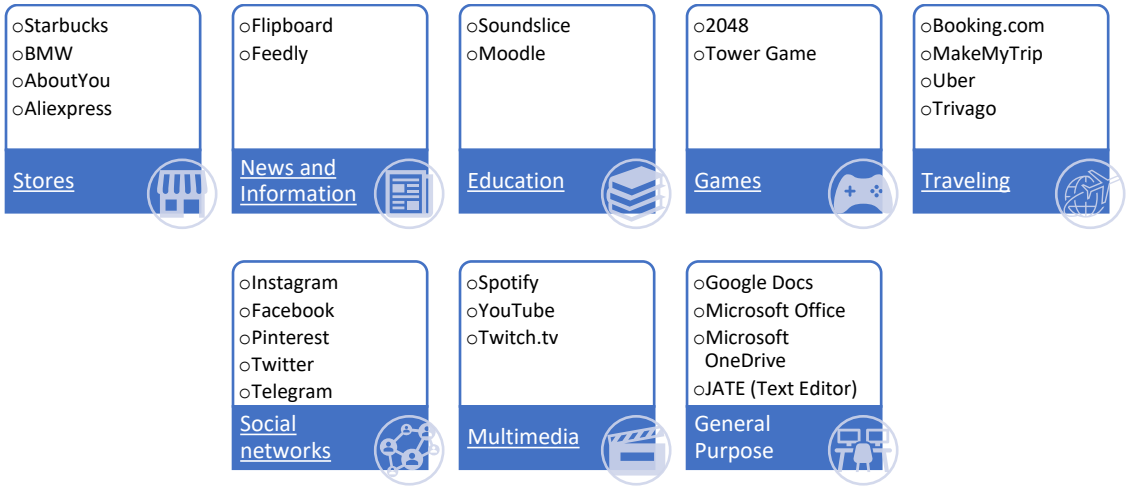


Figure 39 – Examples of some popular PWAs at the time of writing (Simicart Compilations, 2022).

Following figure 38, we can observe a few popular PWAs capable of running in multiple devices as if they were native applications of the device they are running, with full functionality available (Simicart Compilations, 2022).

In figure 39 we can observe one of these apps, Spotify, displayed and correctly adapting its interface to the three mainstream form factors: Desktop, Tablet and Smartphone.

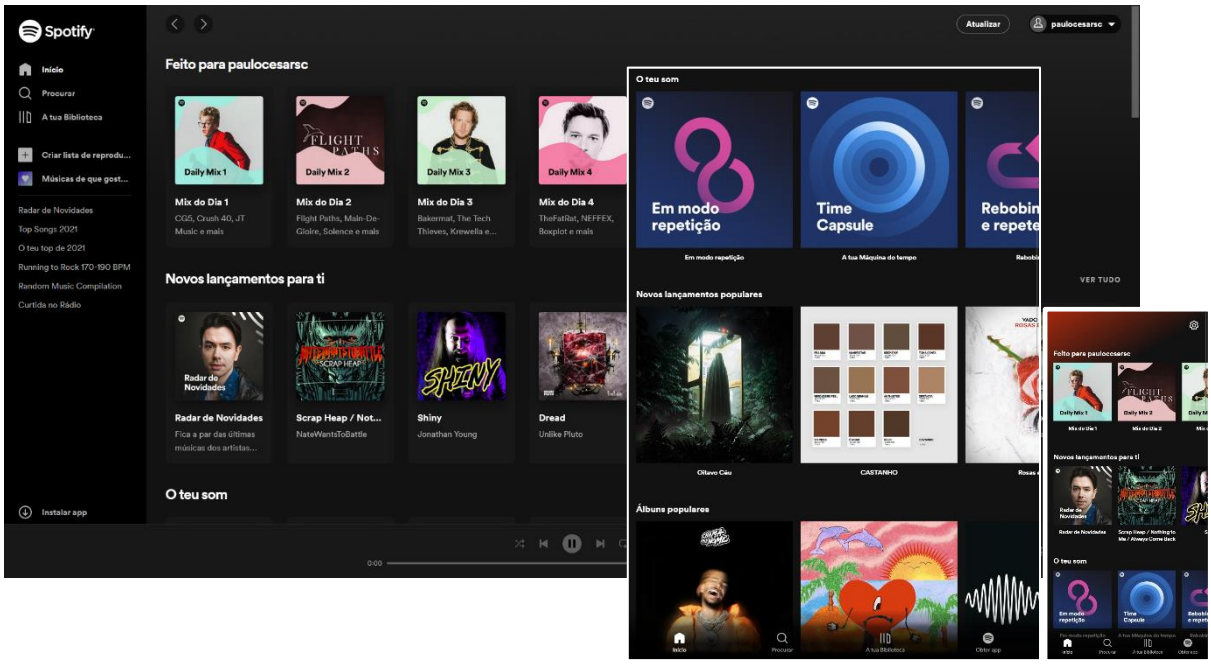


Figure 40 - Spotify PWA adapting itself to Desktop, Tablet and Smartphone form factors (Spotify, 2022).

DESIGNING A CODELESS WEB-BASED PROGRAMMING PLATFORM

Producing a codeless platform, in its essence, is producing a software capable of creating other software: the resulting software can only be as powerful as the base platform allows it to be, dictated by the platform's pre-coded capabilities.

Using Web-Technologies as a base for the platform, in the following section we will explore the creation process in detail for a software platform capable of creating the most basic CRUD software.

ESTABLISHING THE PLATFORM'S OBJECTIVE

A crucial aspect identified in our state-of-the-art investigation of various products was that understanding the intended output of the platform is crucial for not only having a useful tool, but also for avoiding misguiding potential clients.

Using all previously discussed knowledge for this case study, the objective is to create a prototype concept of a codeless platform designed to create most typical "CRUD" projects, with the first step being, determining what exactly are the functionality requirements this platform will need to address.

A CRUD platform has one crucial aspect: turn data into information. Data by itself has no meaning, but when properly processed, it can provide information (Diffen Dictionaries, 2022).

A very good example of an application of data transformation process can be found by looking at the data collected throughout the various countries during the recent pandemic, Covid-19 test results: the information gathered from data from the various districts of a specified country, on a per-day basis, can be evaluated into a graph over the course of a few weeks and provide information such as (Governo Português, 2022):

- Expected previsions of rise or shrink in number of cases
- Most affected areas by population density
- Percentage of Deaths per age-group
- Percentage of Death by district
- Percentage of recovered patients
- Mortality Rate of the virus
- Virus Mortality Rate across mutations
- Overall Country Infection rate compared to other countries (Governo Português, 2022).

This type of data collection and information transformation is exactly the kind of best-suited use case scenario a CRUD platform can have, and due to the systems worldwide not being prepared to handle the kind of massive data collection, where during the first months of the pandemic many institutions not only used Microsoft Excel to catalogue the patients, but also used very old versions of the software, which in turn resulted in the loss of data due to the software not being fitted for the volume of it being collected, which could have proven useful to make decisions on how to guide the containment measures to prevent the proliferation of the Covid-19 virus (BBC News, 2020).

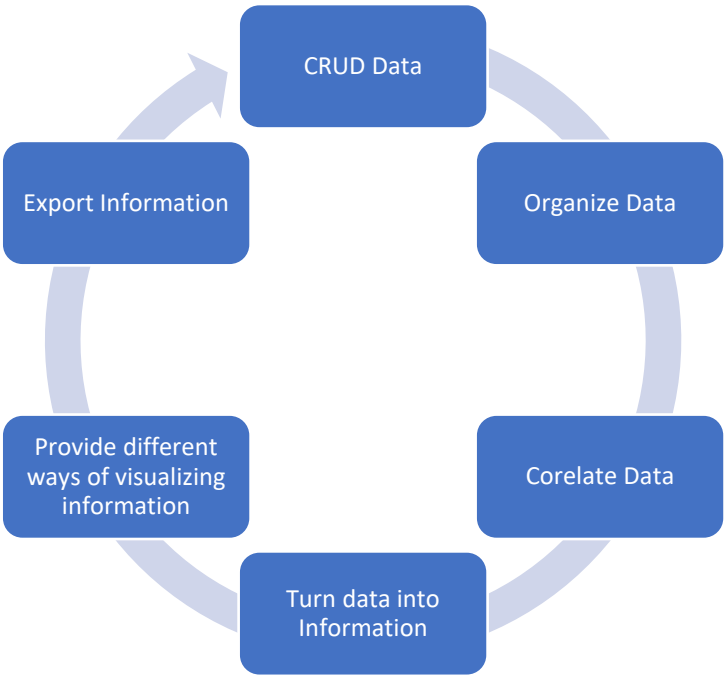


Figure 41 - Typical cycle of use cases for a CRUD platform (Johnston, 2021).

Having overviewed one potential use case, let us analyze the most superficial actions, as demonstrated in figure 40, the previous example given, which are the most common use cases for a CRUD platform to have, the kind of platform we are conceiving has 6 major operations a user expects the platform to be capable of (Johnston, 2021).

To understand the difficulties we may encounter, let us analyze each of these key topics to understand the basic functionality expected from the platform:

1. **CRUD Data**
    a. Expectation:
        i. The platform must be able to do every single CRUD action on the data fed throughout its pages.
    b. Key aspects to consider:
        i. There must be security levels to access and edit the information.
        ii. There must be resiliency when performing Update and Deletion actions to ensure no data is lost.
        iii. The process of performing these actions must be simple and intuitive to the end-user.
2. **Organize Data**
    a. Expectation:
        i. It must be possible to create multiple pages where different kinds of data are inputted by the user either for later consulting or correlation with other data.
    b. Key aspects to consider:
        i. The platform must allow the creation of multiple pages and sub pages
        ii. It must be possible to define which data each page is expected to be given, by providing different form input types and a way to identify each of them.
3. **Correlate Data:**
    a. Expectation:
        i. The platform must be able to correlate data from different pages.
    b. Key aspects to consider:
        i. It must be possible to use data from other pages during the CRUD process of a specific page, allowing data correlation:
            1. Ex.: When creating an Equipment, we must be able to link it with an Equipment Model, which in turn belongs to an Equipment Brand (three different data types / pages).
        ii. When performing changes to already-existing pages, the platform must alert the user whenever a change can result in data-loss.

iii. It should be expected that the user might want to rename any given information field without causing data loss.
iv. Conversion between form input types, in some situations, should be supported, and the tool should help the user through the conversion process:
1. Ex.: The user decided to create a text field named “Supplier” in the equipment’s page, but later decided he wanted a page called “Suppliers” and now wants to convert the existing “Suppliers” text field into a Select field.

4. **Turn Data into information:**
   a. Expectation:
      i. The result of correlating and analyzing data should provide the user with useful information.
   b. Key aspects to consider:
      i. It must be possible to correlate data not only within the same page, but also with other page’s data.
      ii. The information should be able to result from mathematical operations performed on the inputted data.
5. **Provide different ways of visualizing information**
   a. Expectation:
      i. There must be multiple ways of visualizing data.
   b. Key aspects to consider:
      i. Data and information should be able to be viewed in the traditional table-centric disposition.
      ii. Information should be able to be converted into simple “Total” numbers and / or displayed in different kinds of graphs.
      iii. The data must be easily viewed in multiple device formats.
6. **Export Information**
   a. Expectation:
      i. Data should be able to be outputted from the platform in different formats.
   b. Key aspects to consider:
      i. The most standard file types should be outputted, such as:
         1. CSV.

2. Excel.
3. PDF.
4. Image.

ii. There must be a possibility of creating output-layouts for PDF reports.

iii. It should be possible to export graphics in the generated reports (Johnston, 2021).

TARGET AUDIENCE AND PRACTICAL APPLICATIONS

Knowing the platform's basic capabilities expectations and having already observed a potential use case for the platform, the real target audience, and practical applications of the platform itself can be explored.

Starting with the target audience, a codeless, easy to use, web-based platform's main target audience, concentrates mostly in business or end-users who need a way to create a simple application to feed data to and obtain information (Pratt, 2022).

A Web CRUD codeless platform, especially a simple to use and objective-focused one, could help businesses, end-users and even starting developers in different ways, as it can be observed in figure 41.

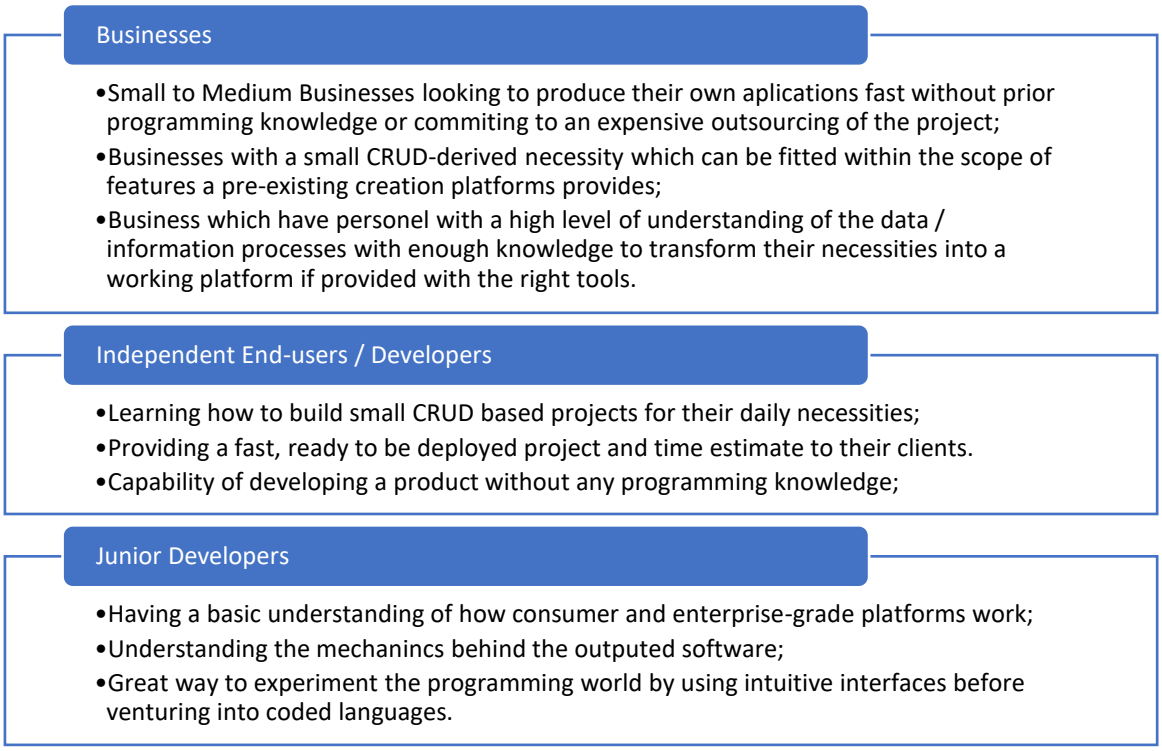


Figure 42 - Target Audience's interest in creating a Web Codeless Platform (Pratt, 2022)

On the practical applications point of view, a CRUD platform can fit any business: after all, billing software is on its essence, a CRUD platform specialized towards processing data and outputting bills, providing the business with a legal level of documentation obligations (Pratt, 2022).

Creating a specialized software such as a billing software, provided the codeless platform is powerful enough, would be possible, but not practical for the businesses to proceed due to the extensive list of legal obligations and validations such software must be capable of guaranteeing, which for the typical use cases shown, would be impractical for businesses to support (Pratt, 2022). There are however many other uses for creating an in-house CRUD software with information-generation capabilities.

In Figure 42, we can observe a few real-world use cases this type of platform could be useful for, spanning across different variety of business types, where many of the examples already having dedicated software capable of addressing those needs, but with the added possibility of being versatile to adding more data sources and correlations beyond the hard-coded pre-existing software's needs: in order words, a more "in-house, business ideology focused" solution created by the business itself.

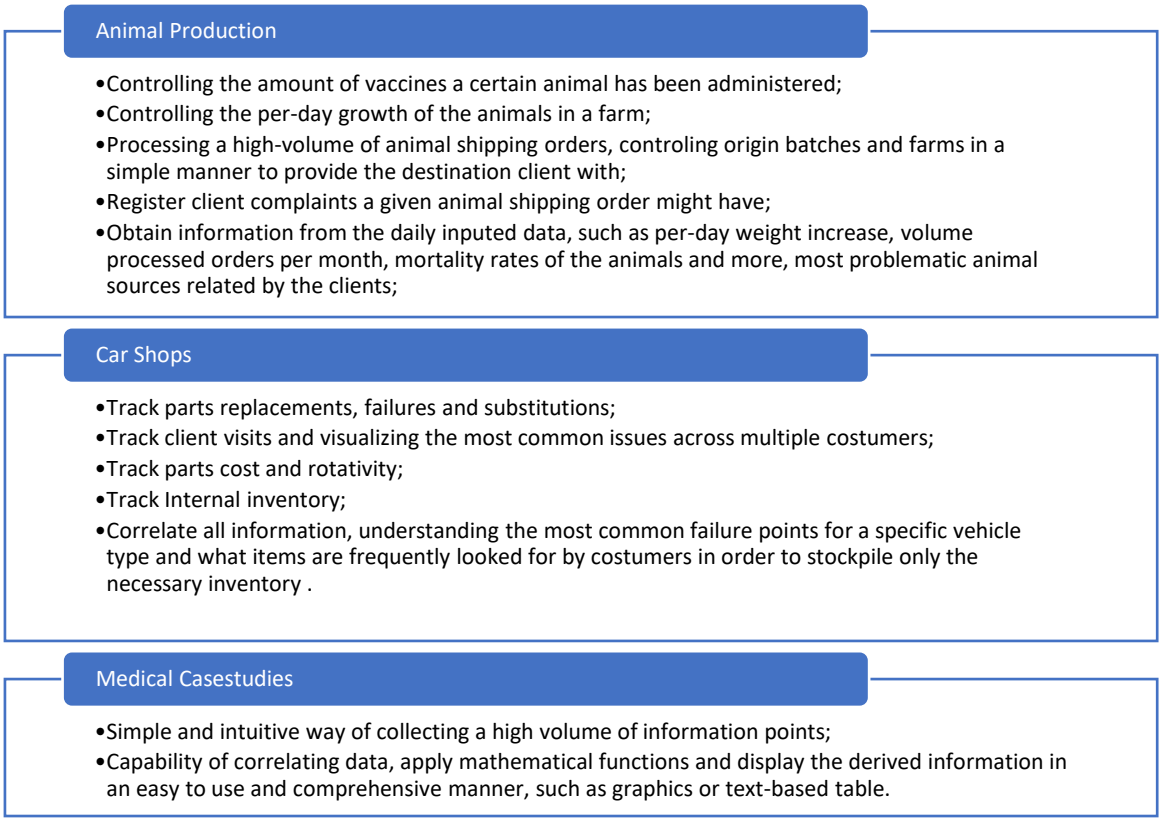


Figure 43 - Practical applications a in-house CRUD project could address (Agri Expo, 2022) (Shopbox Software, 2022) (Governo Português, 2022).

Observing figure's 42 use cases, the limitations for the applications lie on the built-in creation platform's capabilities, which is why it is important for the platform to support different ways of inputting data and customize the data and information gathering pages: the more options the platform has, the bigger the potential client base can be.

SELECTING THE TOOLS

Any programming projects begins with the selection of the tools to be used. As seen in the "Coded languages" subtopic of the "Introduction to Programming Languages", different languages and frameworks can have different levels of abstraction layer performance impacts. Selecting a very complex to process programming language and framework might facilitate the work on the platform's creation, by using already well-established rules set by these languages and frameworks but can hinder the outputting platform's performance if not suited for the workloads expected to be handled.

Other key aspect in developing a project like this is the developer's own skillset and the projects development, time wise, scope: no matter how much more optimized a given language or framework could potentially provide the project with, if the developer is not familiar with the tools, he might not take full advantage of them, or worse, end up using them in a less-optimized way causing severe performance impacts.

Understanding these limitations, it was chosen to base this concept prototype in the following tools, programming languages and frameworks, as specified in figure 43:

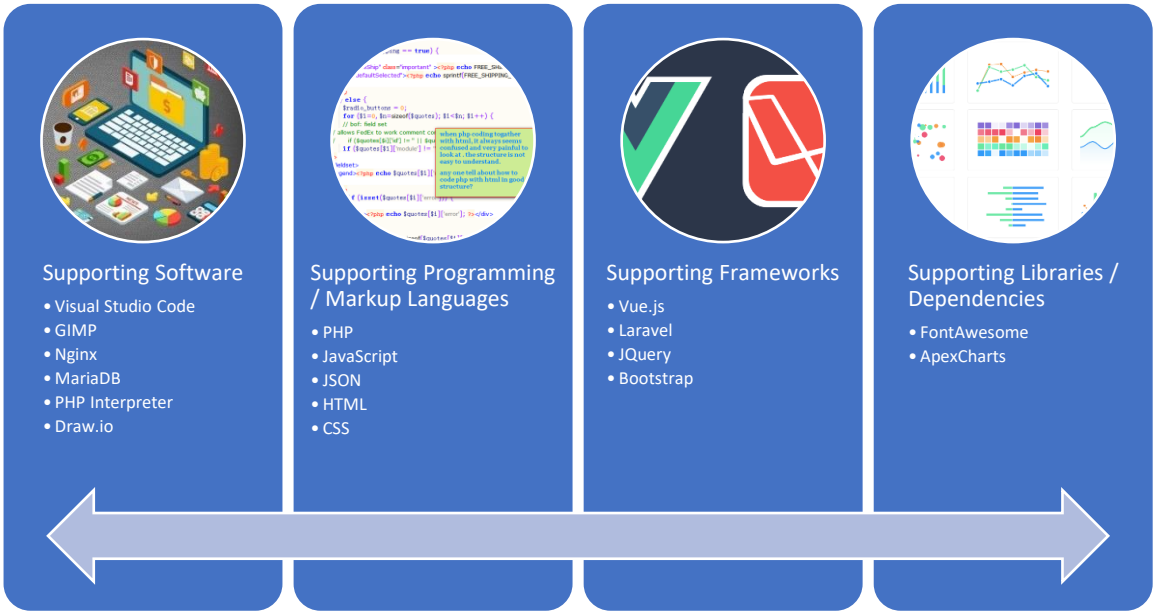


Figure 44 - Chosen Tools, Programming Languages, Frameworks and Supporting Libraries to develop the concept

The selected programming language (PHP) and its complementary supporting framework (Laravel) aren't particularly fast, mostly because PHP itself isn't the fastest language (Juricic & Radosevic, 2018). Due to its amount of abstraction, it is for all purposes a slower language, but it was selected both because it was a focus subject during the degree classes.

Most of the other technologies where also touched upon during the course, except for Vue.JS, which is a platform in which I have some experience.

Not all of the mentioned tools in figure 43 were used in the proof-of-concept prototype: to make development easier, only a few essential tools and phases of the CRUD platform were implemented in this first draft of the prototype.

PLATFORM CONCEPT DEVELOPMENT – DETAILED PHASES

Creating an actual artifact is different from a concept design, where the actual development of the artifact does take place. This does not mean, however, the steps in the software development cycle can be skipped: while a concept can disregard the coding itself, the proper structure and documentation must be done in a way such as the implementation of the real artifact becomes simpler.

Before exploring in detail, the methodologies and case study, let us observe in figure 44 the different phases and their appropriate description. Do note that the phases “Database, Back-End and Front-End” case-studies are consolidated into a single “Automatic Source Code Generation” example, as they all basically follow the same construction principle.

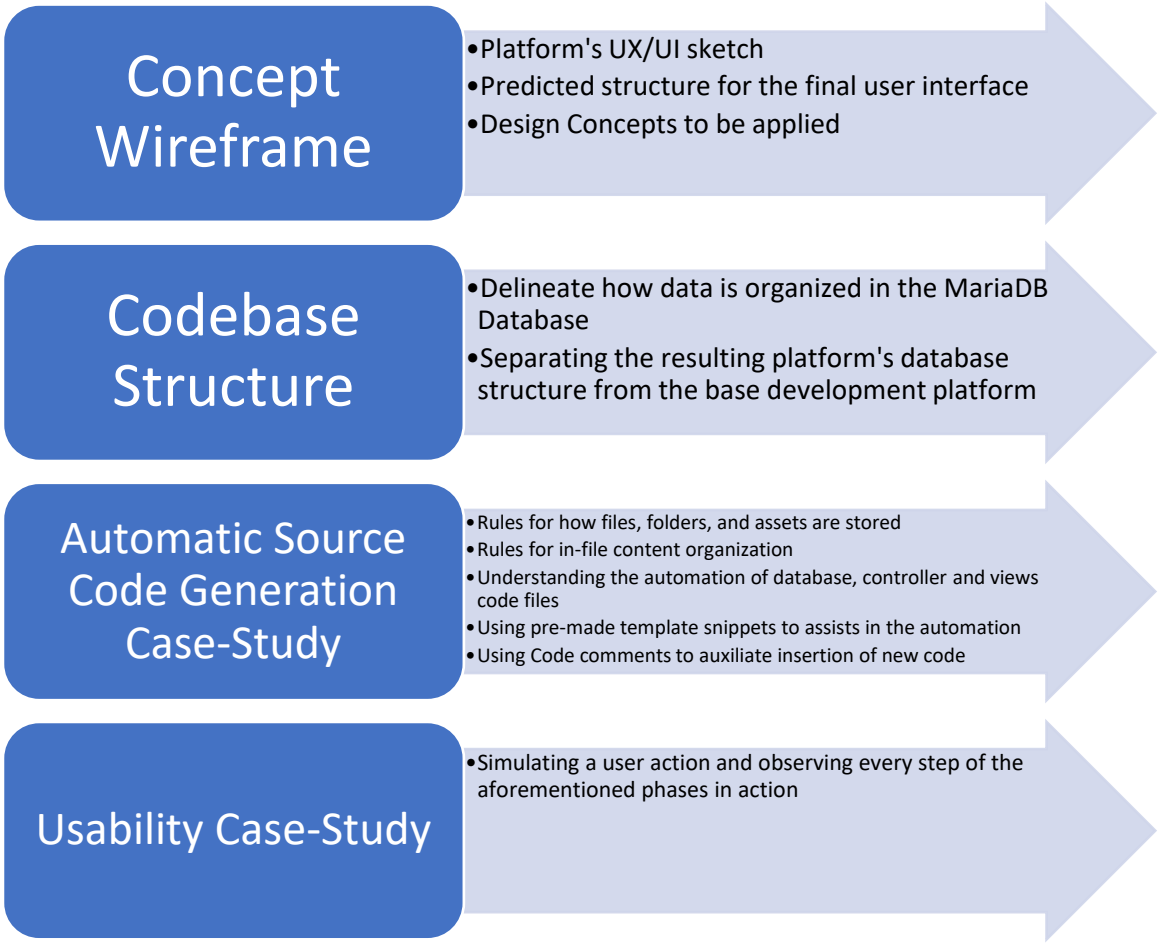


Figure 45 - The 6 phases comprising the "Codeless CRUD platform" case study

CONCEPT WIREFRAME - DEFINING FACTORS FOR THE UI/UX

One of the most important factors a low code / codeless WIMPs based platform have, are its user interface (UI) and user experience (UX), as both are a defining factor for if this type of platforms being the primary and only way of "creating code" for the resulting platform.

To further give emphasis on this concept, in the following texts we will analyze step-by-step the creation of the working area concept, with accompanying justified artificial limitation decisions imposed for the concept platform.

WORK-AREA CONCEPT

The working area of the platform begins with one major limitation: the form factor. Due to the complexity of the UI, and due to the likely use case scenarios being the user using a desktop form factor due to its bigger versatility and precision, the entire working area will be limited to a desktop, horizontal screen real estate form factor.

This limitation is artificial, as it would be entirely possible to create the interface to be executed on other form factors, such as mobile / touch inputs, however, due to the smaller screen real estate and because, with this being a WYSIWYG platform, it would be difficult for the user to visualize the changes on smaller form factors. This is because a desktop screen size can preview how the outputting application will be on smaller form factors, but not the other way around.

WORK-AREA MAIN SECTIONS

Beginning with the interface, and as a rule for the following sections, the wireframe concept is based on a desktop computer running on 100% screen scale with a resolution of 1920x1080 pixels and contained in a web-browser tab being displayed either in full-screen or in a contained tab-view.

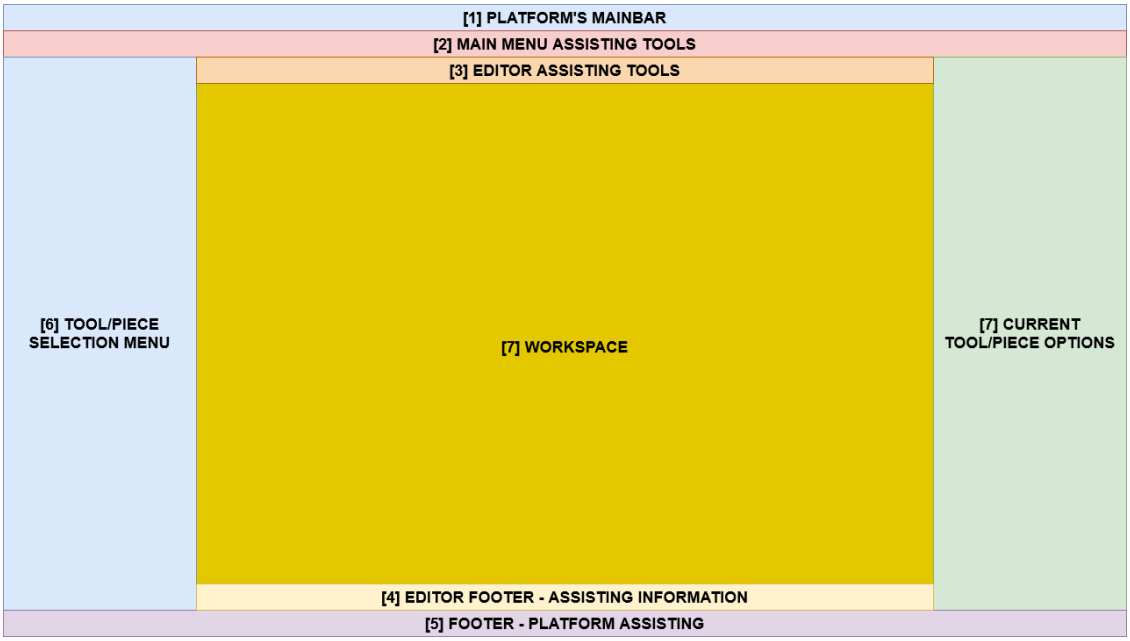


Figure 46 – Basic CRUD Platform Workspace Areas Layout

In figure 45 we can observe the basic structure the CRUD platform has when editing any given page of the platform. The main editor of the platform is page-focused and are as follows:

1. Platform's Main bar:
    a. This will contain the main platform's menus and will provide the user with platform specific assistance, such as save menus, configurations, and more.
2. Main Menu Assisting Tools
    a. These tools will morph depending on the in-view page contents and will allow the user to add different components and add-ons to the current page.
3. Editor Assisting Tools
    a. These tools are focused on the Workspace area and provide the user with easy-to-access functionality to help with the workspace visualization.
4. Editor Footer – Assisting information
    a. This area provides the user with additional information, such as the simulated device model, work area size, and other editor-related information.
5. Footer Assisting Information
    a. This area provides the user with additional information, such as the path for the current page, total objects in use, model relations shortcuts and more.
6. Tool/Piece Selection Menu

a. This section will morph depending on the Main Menu Assisting Tools selection and allows the user to select and drag-and-drop elements into the current page.

7. Current Tool/Piece Options
    a. Allows finer control over the components selected.
8. Workspace
    a. If the user is manipulating a page, this workspace will display the current worked on page is displayed in a WYSIWYG manner, meaning what the user sees is the final version.
    b. In case the user is setting up core-functionality, such as creating new pages, it will display the current future page options.

Do note these sections will be introduced in greater detail gradually, and we will see some components morph as the need by the user arises.

PLATFORM'S MAIN BAR

The Platform's main bar is a narrow horizontal section which will contain the platform's functionality configuration menus. These menus can be used to modify the current platform, switch between projects, export projects, and more.

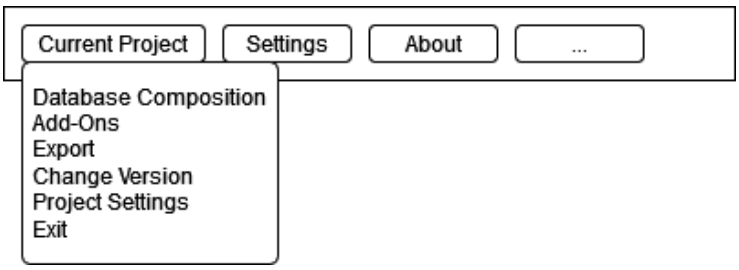


Figure 47 - Example of the possible Main Menu options

An overly simplistic wireframe of this menu, with one of its possible options expanded can be observed in figure 46.

MAIN MENU ASSISTING TOOLS

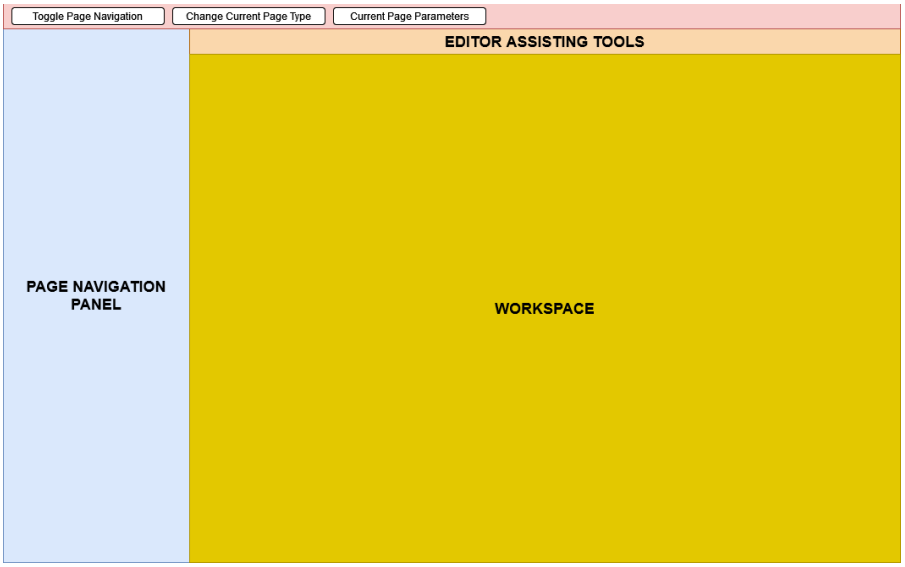


Figure 48 - Example of possible main menu options

This area of the platform, cropped and zoomed-in on figure 47, is dedicated to containing the current work area changing elements. In this area it will be possible to, for example, switch between template page types or adjust current page dedicated settings. By clicking the “Toggle Page Navigation”, this button will overwrite the left panel with a page navigation panel, allowing the user to quickly switch between the current platform’s already existing pages.

EDITOR ASSISTING TOOLS

The editor assisting tools area is designed to help with the current work area items in manipulation, and thus it will morph and change contents depending on what the user is currently performing on the platform.

Figure 49 - Example of possible tools existing in the editor assisting tools menu

In figure 48 is an example of a tool which would be placed on this panel: the “preview” toggle, which allows the user to see the current page in various form factors. Additional settings, such as rotation and form-factor specific behaviors, such as touch-input or mouse-input toggles are other examples of other types of tools provided in this section of the platform.

EDITOR FOOTER - ASSISTING INFORMATION

Figure 50 - Example of additional functionality provided in the footer

In this area, assisting information and tools can be displayed. Following the previous example of the form factor switch buttons, along with figure 49, additional options, such as the current simulated device settings or current display size being exhibited can be provided in this panel.

TOOL/PIECE SELECTION MENU

This menu will provide the user with any components the current page template allows to be added. For example, if the page template is a form creation page, it will allow the user to add different types of inputs, buttons, images, text, and other necessary information the user requires.

CURRENT TOOL/PIECE OPTIONS

By selecting a piece placed in the work area, the right area will be populated with its possible options. For example, if a user adds an input field, he can change its description, data type, and more. The platform will create all necessary database fields based on the information provided, or the user can map a specific field to an already existing database field.

WORK AREA

The work area is where the current page preview is displayed, or any configurations of the platform currently being displayed.

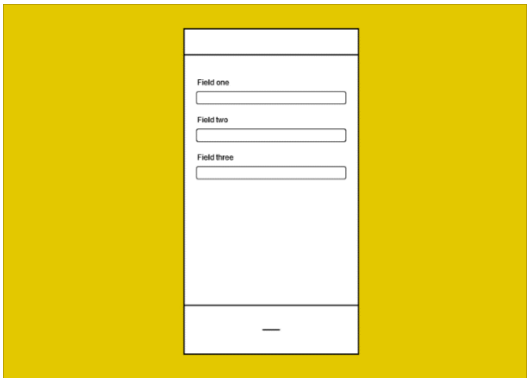


Figure 51 - Example of a form-factor simulation for a simple form page

Following figure 50 we see an example of a page constructed by three input fields being previewed in a mobile form factor. The user can directly interact with this page, moving and resizing elements and adding new pieces to the page using the tool/piece selection menu.

USABILITY CASE-STUDY - SIMULATING AN USER ACTION

Understanding how the concept platform performs all the tasks a typical web developer would perform to create the platform, and to consolidate knowledge, let us follow step-by-step all actions related to the creation of a “client” form page. After this brief walkthrough trough the produced artifact, a more detailed overview of how the source code of the application was built will be given.

In a quick overview of the process behind the artifact, every element the user creates is stored in a database, to allow the software to re-create all automatically generated code for the output project such as the database, back-end and front-end operations. This process in the following example is implicit in every step. We can also assume, as it will later be introduced that the platform will use template files as a basis when creating new files.

To better understand this, let us follow the workflow in the current platform’s prototype to generate a simple project used to store “Clients Information”.

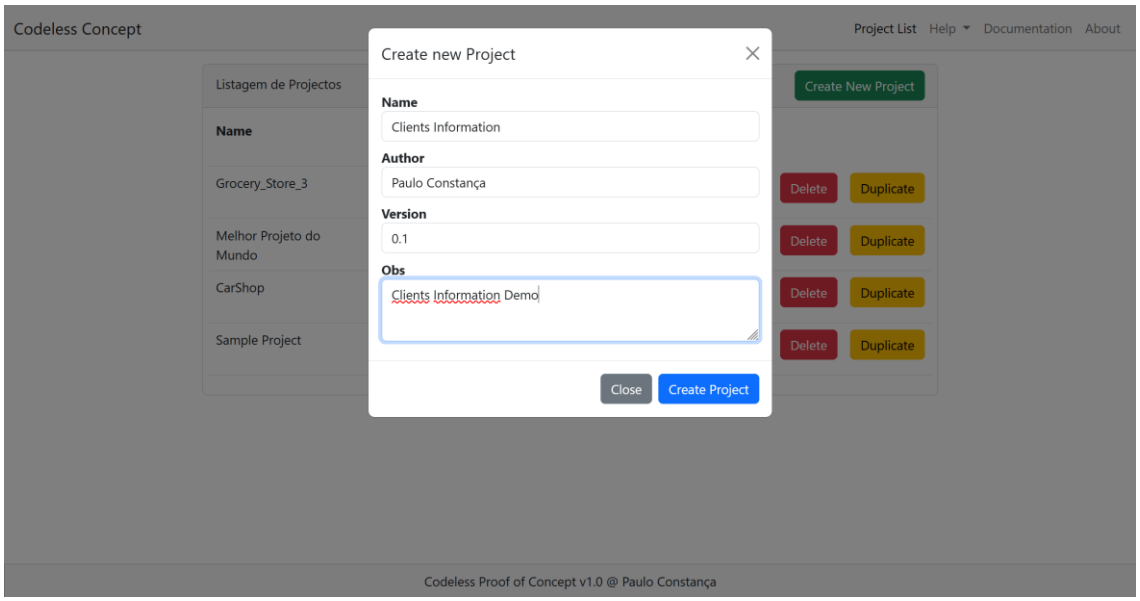


Figure 52 - Creation of a new project

First, the user creates a new project called “Clients Information”, specifies the author and the version, along with a small observation.

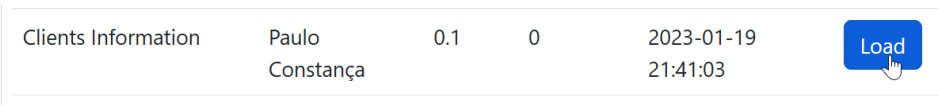


Figure 53 - User loading a project

After creating the project, the user presses the “Load” button on the project list table, as seen in figure 52.

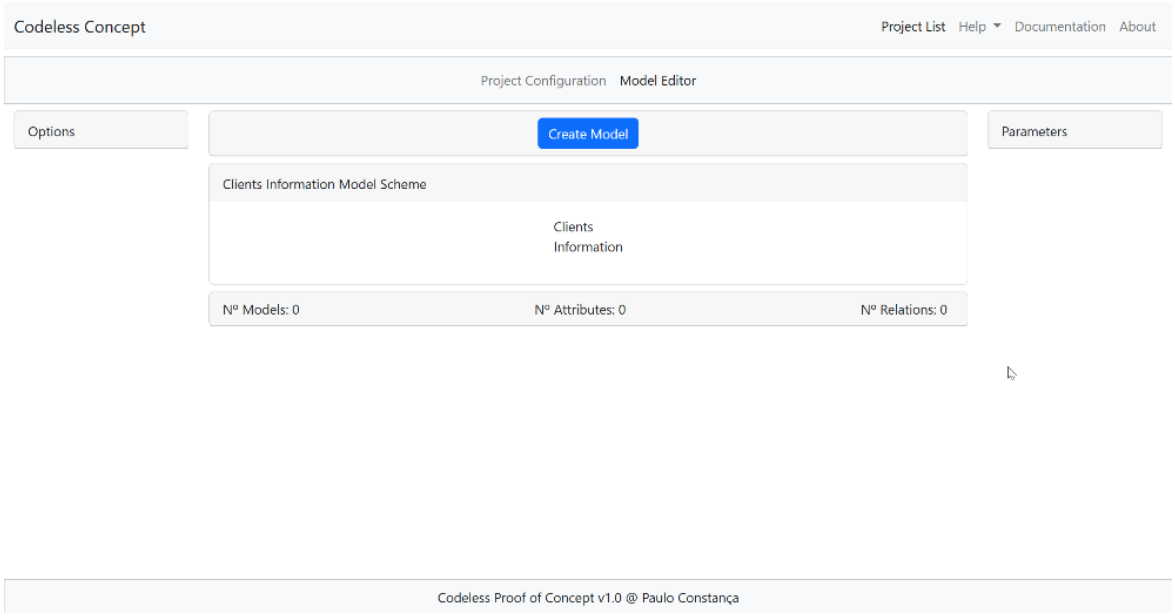


Figure 54 - Empty dashboard without any models or attributes.

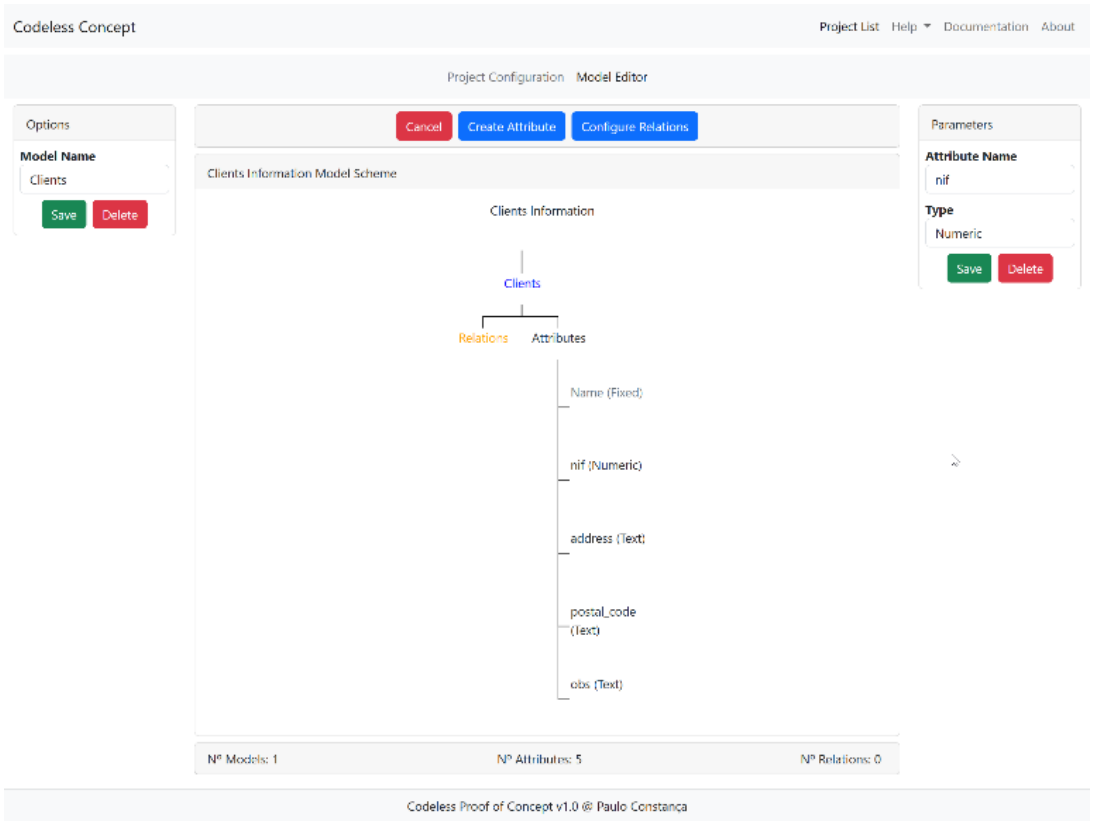


Figure 55 - Dashboard with some actions already performed by the user.

After loading the project, the user is presented with an empty board in figure 53 where he can begin creating the models and fields. The user can now begin by creating Models (or "Pages") and specify which fields each of the models should have (attributes).

After this process is done, the user can click on "Project Configuration" and deploy the project as seen in figure 54.

Once the user is satisfied with the models and attributes created, he can now generate the final project based on his actions. To do so, the user must head into the “Project Configuration” and press the “Deploy” button, where the platform will generate the final output project, compile it into a compressed *.zip file and download it in the user’s browser.

The final output project is something like the following figures 55, 56 and 57.

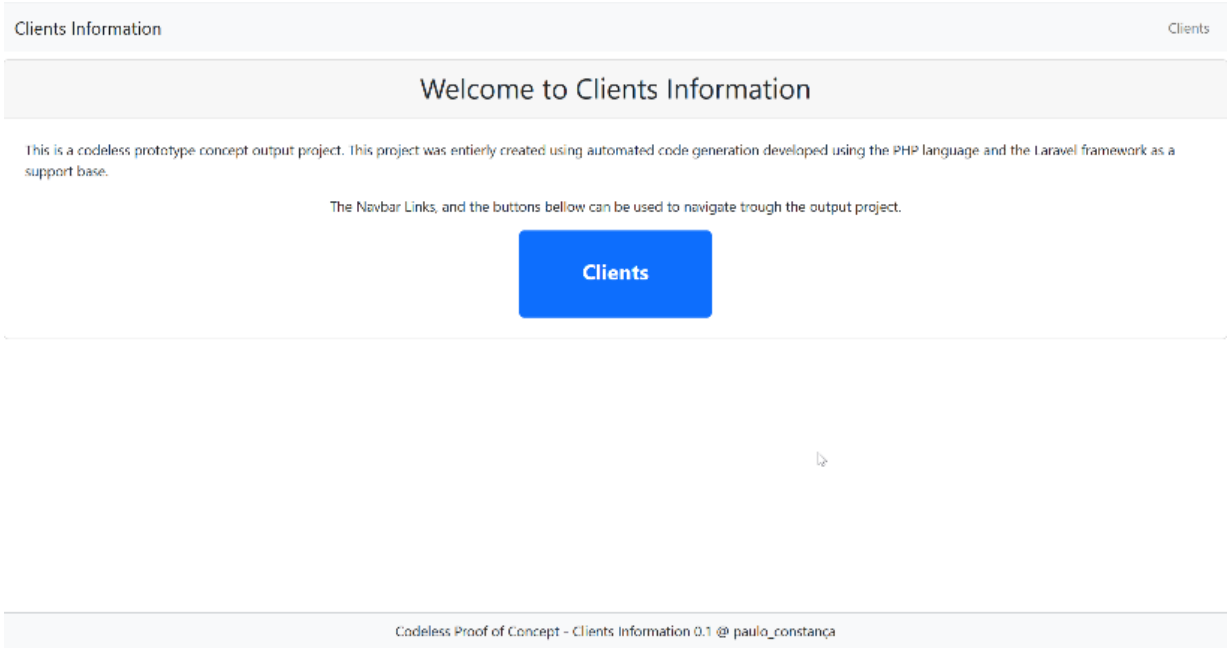


Figure 56 - Homepage of the generated concept project

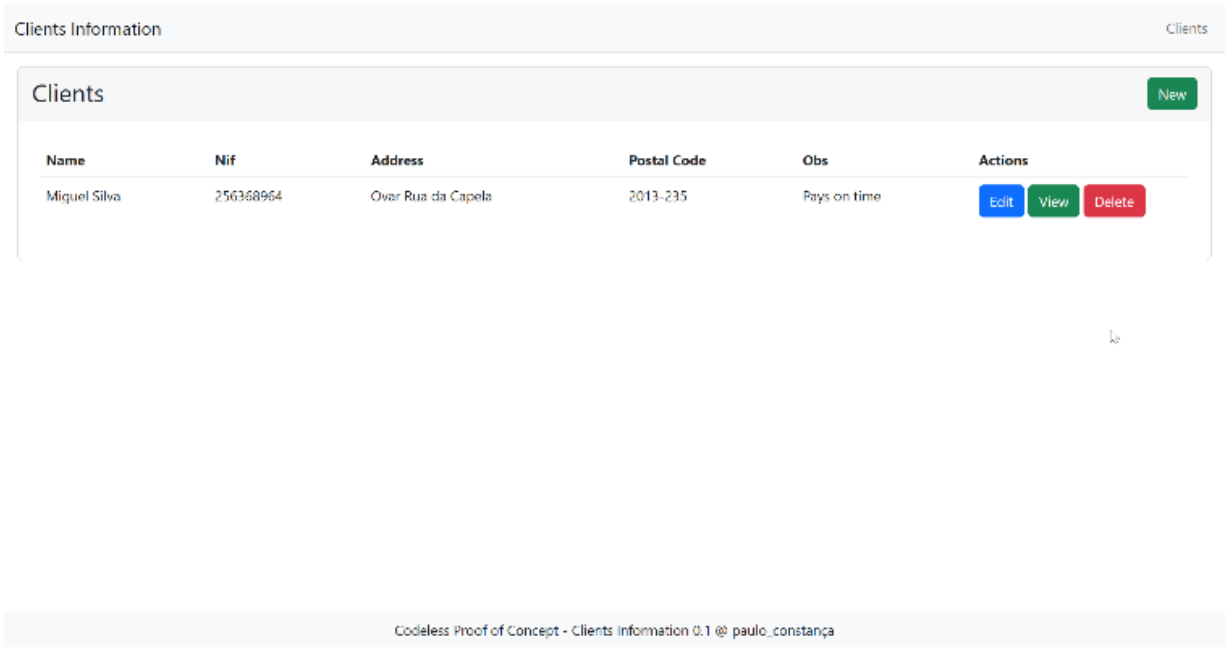


Figure 57 - Project Client's page, showing the information’s the user introduced during the project creation process.

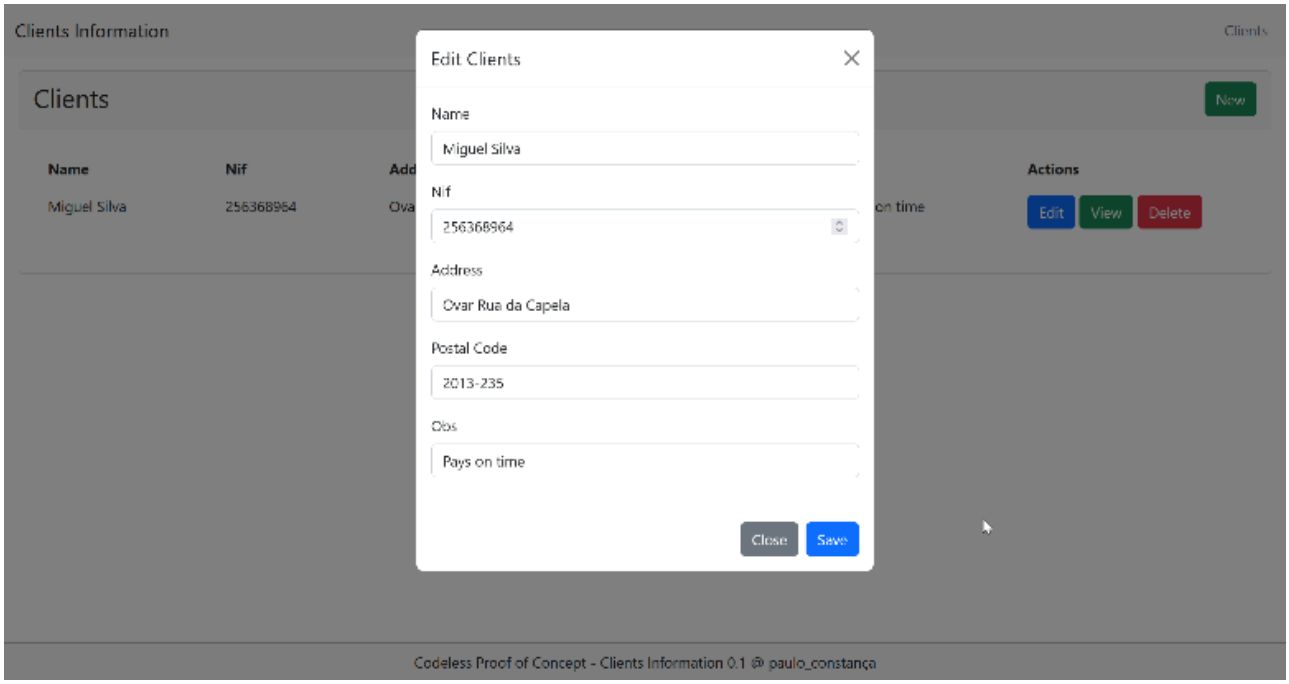


Figure 58 - Data edition modal showing the "Miguel Silva" client details.

With this process done, the user can begin to use the new platform to create as many client entries as needed and navigate the project like a normal web application works.

This concludes the walkthrough of the concept platform created. To better understand how a simple interface containing models and attributes generates a website, let us look at the codebase created for this artifact.

CODEBASE ORGANIZATION

The main functionality of the codeless platform is to generate source-code based on user instructions, but to achieve this the platform itself needs to have its own code and code-templates, or snippets which are functionalities programmed to build the tools intended by the user. To achieve this capability, it was chosen to do this is by dividing the code snippets into smaller “template” files and delimiting the code within them with specific comments the platform will search for to locate and use the appropriate snippet.

To use these files, the platform makes use of the Laravel framework built-in command methodology by creating custom events which generate the output files based on source templates.

PLATFORM EXPORT MODES AND CODE SECURITY

Before advancing, it is important to understand that this platform is predicted to have two different exportation modes for the outputting project: a standalone mode and an independent mode, however, for the current case study, only the independent mode will be implemented.

For the standalone mode, the platform’s code itself will be uploaded to the destination server and will function similarly to how the WordPress low-code platform works, with the administration pages being accessible through a dedicated URL path.

In the independent mode, a clean version of the outputted project will be generated containing only the output code necessary for the created platform to function.

This prevents the platform itself from being stolen and manipulated by a potential client, and the source-code can potentially be obfuscated (either through a coded-in mechanism or through the use of third-party obfuscation software) to difficult any changes by the developer’s client (Lutkevich, 2021).

Obfuscation is necessary only for when the service is being provided on a contract basis, and it is critical to guarantee that only developer has access to the codeless platform editing capabilities, as all web-projects require the source code to be uploaded directly to the destination server which, in this scenario, would not be under control of the developing entity.

FOLDER STRUCTURE

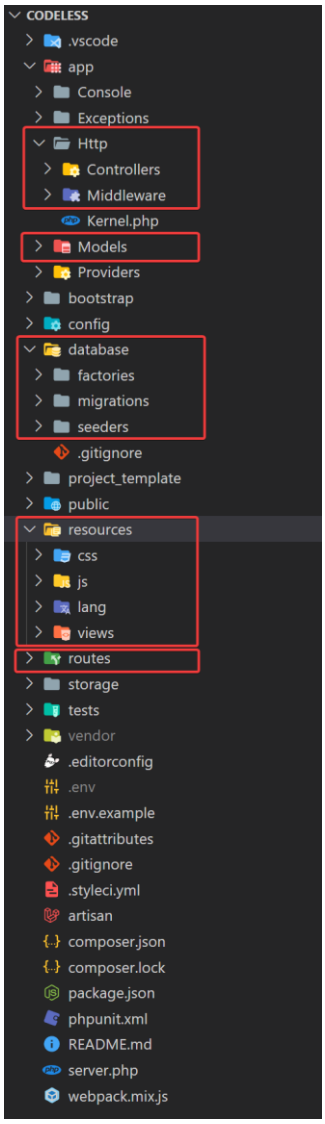


Figure 59 - Laravel 9 Folder Structure

Since the platform was built using the PHP's based Laravel framework, it makes sense to use Laravel's folder and file organization scheme as a basis, where all of the codeless platform's own code will also reside.

The main folders to contain the codeless platform code are exhibited in figure 58 and are as follows:

- App\Http\Controllers
- App\Http\Middleware
- App\Models
- Database\Factories
- Database\Migrations
- Database\Seeders
- Resources\CSS
- Resources\JS
- Resources\Views
- Routes

These 10 folders are the basis for the produced codeless platform prototype itself: all of the platform's code and pages will reside in these folders, as well as dependencies and auxiliary frameworks and modules.

Since Laravel uses the MVC model, the base structure includes the Controllers, which contains code that enables the user to make communications between the pages and the database, being validated by the Middleware.

Models are representations of concepts outlined by fillable fields and respective relations with other models.

Factories are self-imposed rules made by the user that allows the creation of model instances in a quick and automatic way.

Migrations are the template models used to generate the database table structure by specifying the table name, field names and field types. Since Laravel is a code first framework,

the process of creating the database is done by writing code into the framework itself not needing the manual configuration of a database.

Seeders are useful to generate dummy data of Models using manual inputted data or Factory originated data. These are mostly used for testing purposes and not included in the output product.

CSS and JS folders are used to contain the platform's front-end code, with the CSS folder being used to store styles and the JS folder to store front-end logic written in JavaScript.

The Views folder is used to store the Laravel Blade's front-end code: these files contain the HTML structures which originate the front-end aspect of the project. Due to its capabilities, Laravel Blade files can also contain logic, processed by the server on page opening.

Finally, the Routes folder contains the project navigation logic, which allows the end user to navigate between the different project pages and execute the various CRUD operations over the project.

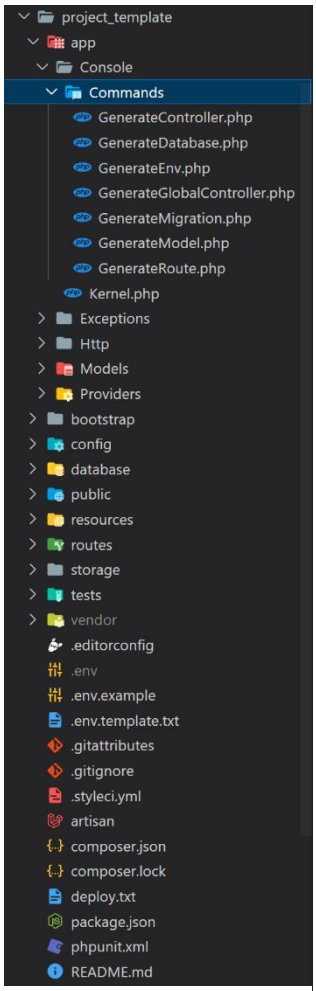


Figure 60 - Project Template folder Structure

Shown in figure 59 is the source code of the output platform. The folder structure is like the codeless platform itself, with the main difference being the App\Console\Commands folder, this time used to store events that can be ran with Laravel commands.

These events contain the code that turns the template project folder data into the output project, as the user specified, using the template files as a basis.

The template files are stored into each of the Laravel's previously mentioned folders: Controllers templates are in the App\Http\Controllers folder and so on.

The files contained in these folders will have the basic structure the PHP, HTML and JavaScript files. All external frameworks will either be contained in the "public" or this folder. (Laravel, 2022).

For example, if it is necessary to create a database text field, the platform will use a "field sample" along with a "database migration file sample" and adapt it as needed to create this new field.

AUTOMATIC SOURCE CODE GENERATION CASE-STUDY

Knowing the folder organization, it is time to identify how the platform will use the code snippets. First, when parsing the files, the platform needs to know where to input certain types of data. To do this, these files contain a unique character sequence the platform can use as a basis.

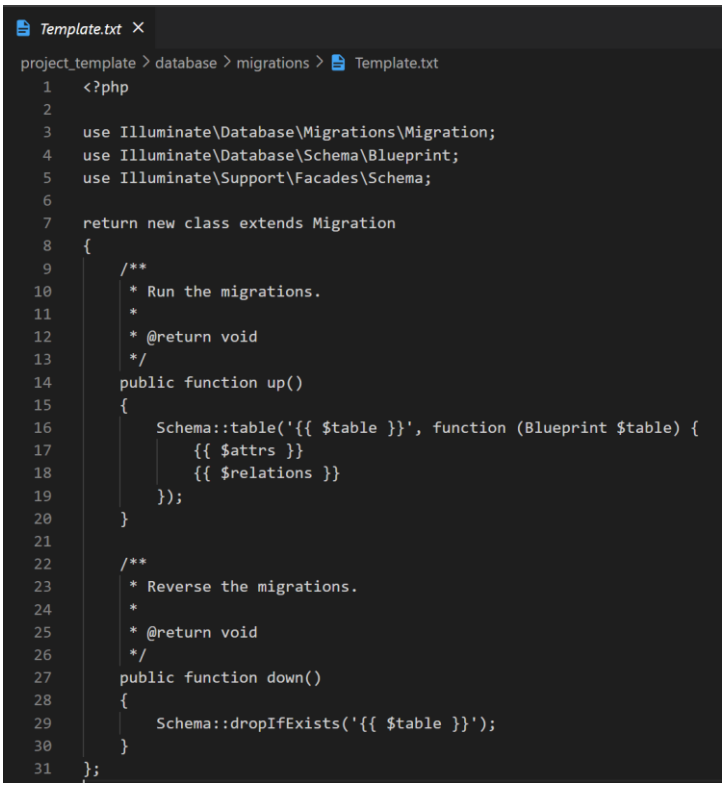


Figure 61 - Template file structure for a database table creation migration

Following figure 60, we have an example of the template file for a database table creation migration.

In this example we can note three different unique code strings the platform's code will use to populate this template with:

- **{{ $table }}**
  - This anchor will be replaced with the to-be-created table's name. This name should be unique in the destination database.
- **{{ $attrs }}**
  - This anchor is placed in this specific file section, so the platform knows where to insert the fields to be created for this table. The code placed here will be provided by another template file containing each respective column type template snippets.

- **{{ $relations }}**
    - This anchor is replaced with code pertaining to each model relation with other models.

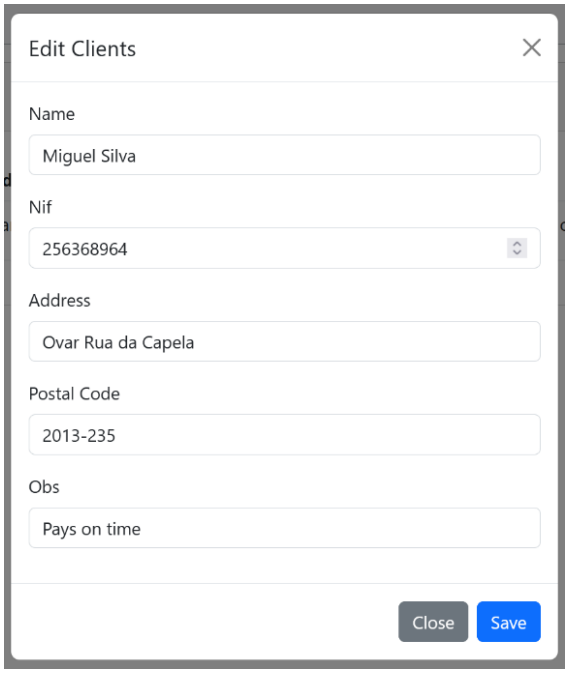


Figure 62 - Edition Modal of the platform, for attribute comparision.

To complement the introduced example, let us assume the user decides to create a "client" table and add the following fields, also exhibited on Figure 61:

- Name [Text Type]
- NIF [Text Type]
- Address [Text Type]
- Postal Code [Formatted Number Type]
- Observations [Text Area]

As explained, the platform will generate the necessary migrations to create the database once the platform is exported, but it also needs to do this on-the-fly to allow the user to preview the platform in the work area.

The template file it will use was previously shown in figure 60, but to create the fields, the platform will need to populate Laravel's "up" function with the appropriate code.

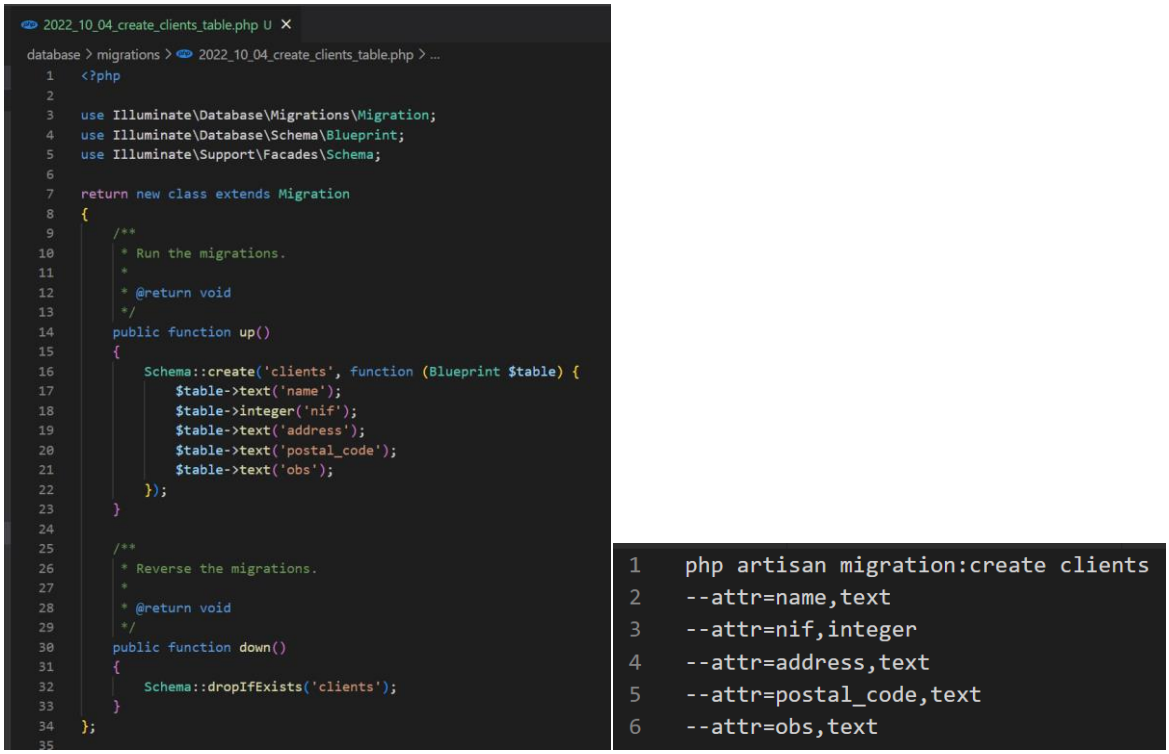


Figure 63 - Example of an output file of the codeless platform, ready to be interpreted by PHP and PHP artisan command line interface (Left) and example of a migration event (right)

Using the given example and using the migration event seen in figure 62, the platform would then generate the following final output file, as seen in figure 62 to the right. To do so, the platform renames the file accordingly and generates the database table name and column names containing the necessary code to create the clients table with the user's desired fields, placing it in its own database migrations folder to be executed by the current in-execution platforms "php artisan" command line mechanism.

This same principle will hold true for the generation of both the back end (PHP) code and the front-end (Laravel Blade / JavaScript) code, with the platform having as many template files and code snippets as necessary to account for all potential user-actions.

CONCLUSION

Software development, no matter if it is destined to the creation of dedicated software or an application-creation platform such as the case study, is a long and complex process. Understanding how the typical modern computer works and having a basic understanding of what is and how a programming language works is crucial to begin developing software.

To create a platform capable of allowing an inexperienced user, all explored technologies and concepts need to come together as harmoniously and as intuitive as possible: after all, a codeless platform is for all purposes a software capable of creating other software.

On the Web Development side, web technologies have grown in functionality and capabilities over the years, allowing for incredibly powerful web applications to be exist and being used directly in any device without requiring any installation, as seen during the document, many types of software with many different workflow targets functioning under a web powered server and browser engine.

The inclusion of PWA and broader adoption, and with web technologies becoming ever more popular, a shift has begun to be noticed in how developers chose to focus their time: by creating a dedicated PWA, one developer can not only have his work reduced, but also target several platforms and form factors at once.

Having a universal interface to accommodate all form factors and forms in interaction is difficult, and it is up to the developer to integrate support for inputs such as mouse and keyboard, or touch inputs in his application to assure full compatibility across devices and platforms, and as seen, there are many frameworks and other tools a developer can choose from to speed up the process.

Low Code tools are beginning to become more prevalent, even more so than codeless tools due to their versatility and possibilities of adding external functionality which does not exist in the low code platform, with the biggest platforms having a notorious focus on business productivity and applications.

Creating a Codeless platform is like creating a dedicated software: if the objective is to facilitate the creation of a BPM system, creating a codeless platform geared towards this problem is the best course of action, just like creating a CRUD creation platform was for the case study performed.

Using all knowledge introduced during the research phase, the small CRUD platform concept was conceived with the objective of showing how all parts of the coding process, namely how the programming languages, tools, frameworks, and other technologies all played a part in the design and creation of this platform.

The added value of describing all technologies and creating this concept is that, first, it provides a more entry-level semi-in-depth view on the creation process of these types of platforms have as well as showing a proof of concept where, by applying web-centric technologies a proof of concept tool was designed towards businesses, where this new idea of a concept CRUD platform designed to manage data and turn data into information was created, becoming a great starting point in creating a business-ready artifact to be used in real-world business scenarios.

FUTURE WORK

As a proof of concept, a small working prototype was made, proving that a codeless project is possible and viable however many of its anticipated functionality was not implemented.

The idea can be further developed into adding more functionality and elevating the platform from a simple data-information managing system to a more complex and sophisticated business software, allowing the implementation of other business critical functionality.

As future work and as a validation of the present concept, the next step should be developing a more sophisticated artifact capable of being used to design a more complex CRUD platform and eventually being used in a real environment.

BIOGRAPHY

Agri Expo. (30 de 09 de 2022). *AgriExpo.* Obtido de AgriExpo: https://www.agriexpo.online/agricultural-manufacturer/animal-husbandry-software-963.html

AkinsIT. (10 de 09 de 2022). *AkinsIT.* Obtido de Understanding Storage Capacity: https://www.akinsit.com/understanding-storage-capacity

Alain, G. (12 de 02 de 2022). *A Review of Shader Languages.* Obtido de A Review of Shader Languages: https://alain.xyz/blog/a-review-of-shader-languages

Altvater, A. (8 de 04 de 2020). *What Is SDLC? Understand the Software Development Life Cycle*. Obtido de Stackify: https://stackify.com/what-is-sdlc/

Appian. (23 de 09 de 2022). *Appian Platform.* Obtido de Appian: https://appian.com/

ARM Corporation. (13 de 09 de 2022). *What Is an Instruction Set Architecture?* Obtido de ARM: https://www.arm.com/glossary/isa

Asper Brothers. (13 de 12 de 2020). *Cross-Platform App Development – Explore Frameworks, Technology and Business Benefits* . Obtido de Asper Brothers: https://asperbrothers.com/blog/cross-platform-app-development/

Awati, R. (04 de 06 de 2019). *Binary.* Obtido de Binary: https://www.techtarget.com/whatis/definition/binary

BBC News. (5 de 10 de 2020). *Excel: Why using Microsoft's tool caused Covid-19 results to be los.* Obtido de BBC News: https://www.bbc.com/news/technology-54423988

Benveniste, A. (1998). Synchronous Languages and Reactive System Design. *ScienceDirect*.

Big Water Consulting. (08 de 04 de 2019). *Software Development Life Cycle (SDLC).* Obtido de Big Water Consulting: https://bigwater.consulting/2019/04/08/software-development-life-cycle-sdlc/

Bootstrap. (29 de 09 de 2020). *Bootstrap.* Obtido de Bootstrap: https://getbootstrap.com/

Both, D. (23 de 07 de 2020). *The central processing unit (CPU): Its components and functionality.* Obtido de RedHat: https://www.redhat.com/sysadmin/cpu-components-functionality

Britannica Dictionaries. (24 de 08 de 2002). *software.* Obtido de Britannica Dictionaries: https://www.britannica.com/technology/software

Britannica Dictionaries. (12 de 09 de 2022). *declarative language.* Obtido de declarative language: https://www.britannica.com/technology/declarative-language

Britannica Dictionaries. (03 de 01 de 2022). *functional language.* Obtido de functional language: https://www.britannica.com/technology/functional-language

Britannica Dictionaries. (23 de 09 de 2022). *logic programming language.* Obtido de logic programming language: https://www.britannica.com/technology/logic-programming-language

Britannica Dictionaries. (26 de 09 de 2022). *numerical analysis.* Obtido de numerical analysis: https://www.britannica.com/science/numerical-analysis

Britannica Dictionaries. (26 de 10 de 2022). *World Wide Web.* Obtido de Britannica: https://www.britannica.com/topic/World-Wide-Web

Britannica Dictionary. (07 de 09 de 2022). *binary code.* Obtido de Britannica Dictionary: https://www.britannica.com/technology/binary-code

Britannica Dictionary. (20 de 10 de 2022). *Britannica Dictionary.* Obtido de software: https://www.britannica.com/technology/software

Britannica Dictionary. (15 de 10 de 2022). *computer programming language.* Obtido de Britannica Dictionary: https://www.britannica.com/technology/computer-programming-language

Browne, C. (13 de 04 de 2004). *Unix Shells - csh, ksh, bash, zsh, ...* Obtido de Christopher Browne: https://web.archive.org/web/20071108064620/http://www.linuxfinances.info/info/unixshells.html

Burak, A. (20 de 02 de 2022). *What Agile Software Development Team Structure Looks Like.* Obtido de Relevant Software: https://relevant.software/blog/what-agile-software-development-team-structure-looks-like/

Cambridge Dictionary. (15 de 10 de 2022). *impotent.* Obtido de Cambridge Dictionary: https://dictionary.cambridge.org/pt/dicionario/ingles/impotent

Caniuse. (30 de 09 de 2022). *Can I Use PWA.* Obtido de Can I Use: (Mozilla Developers, 2022)

Carlson, B. (09 de 08 de 2022). *Are Low-Code/No-Code Development Platforms Living Up to the Hype?* Obtido de CDP: https://cdp.com/articles/low-code-no-code-development/

Cassel, D. (20 de 5 de 2018). *Which Programming Languages Use the Least Electricity?* Obtido de The New Stach: https://thenewstack.io/which-programming-languages-use-the-least-electricity/

CERN Laboratories. (17 de 09 de 2022). *A short history of the Web.* Obtido de CERN: https://home.cern/science/computing/birth-web/short-history-web

Chawla, M. (16 de 07 de 2016). *Levels of Programming Languages*. Obtido de The Bit Theories: https://thebittheories.com/levels-of-programming-languages-b6a38a68c0f2

CHope. (30 de 06 de 2019). *Machine Language.* Obtido de CHope: https://www.computerhope.com/jargon/m/machlang.htm

CHope. (30 de 06 de 2019). *Procedural language.* Obtido de Procedural language: https://www.computerhope.com/jargon/p/proclang.htm

CHope. (07 de 06 de 2021). Computer Hardware. *Computer Hardware*, p. https://www.computerhope.com/ahardwre.htm.

Cocca, G. (16 de 03 de 2022). *What is Debugging? How to Debug Your Code for Beginners.* Obtido de Free Code Camp: https://www.freecodecamp.org/news/what-is-debugging-how-to-debug-code/

Codecademy. (13 de 09 de 2022). *What Is an IDE?* Obtido de Codecademy: https://www.codecademy.com/article/what-is-an-ide

Codecademy. (27 de 09 de 2022). *What Is an IDE?* Obtido de Codecademy: https://www.codecademy.com/article/what-is-an-ide

Concatenative.org. (15 de 09 de 2022). *Concatenative Language.* Obtido de Concatenative Language: https://concatenative.org/wiki/view/Concatenative%20language

Crucial. (27 de 10 de 2022). *What is a Form Factor?* Obtido de Crucial: https://www.crucial.com/articles/pc-builders/what-is-a-form-factor

Danciu, A. (26 de 04 de 2018). *Follow these key steps to start a successful software development project.* Obtido de Frecodecamp: https://www.freecodecamp.org/news/follow-these-key-steps-to-start-a-successful-software-development-project-163c838e8fe1/

Diffen Dictionaries. (30 de 09 de 2022). *Data vs. Information.* Obtido de Diffen: https://www.diffen.com/difference/Data_vs_Information

Dizdar, A. (22 de 02 de 2021). *Bright*. Obtido de SQL Injection Attack: Real Life Attacks and Code Examples: https://brightsec.com/blog/sql-injection-attack/

Esolangs. (01 de 01 de 2022). *Language list.* Obtido de Esolangs: https://esolangs.org/wiki/language_list

GDevelop. (29 de 09 de 2022). *GDevelop.* Obtido de GDevelop: https://gdevelop.io/

GFG. (16 de 04 de 2022). *Fourth Generation Languages and End Users – DP Management Attitudes.* Obtido de GeeksForGeeks: https://www.geeksforgeeks.org/generation-programming-languages/

Github. (01 de 06 de 2021). *Embedded scripting languages.* Obtido de Github: https://github.com/dbohdan/embedded-scripting-languages

GitHub. (29 de 09 de 2022). *Top IDE index.* Obtido de Github: https://pypl.github.io/IDE.html

GNU. (13 de 09 de 2022). *What are scripting and extension languages.* Obtido de What are scripting and extension languages: https://www.gnu.org/software/guile/docs/master/guile-tut.html/What-are-scripting-and-extension-languages.html

Google Corporation. (03 de 01 de 2022). *Android Studio.* Obtido de Android Developer: https://developer.android.com/studio

Governo Português. (30 de 09 de 2022). *Covid19 Estamos On.* Obtido de Covid19 Estamos On: https://covid19estamoson.gov.pt/

Halwai, S. (25 de 06 de 2021). *OpenXcell.* Obtido de System Software - Definition: https://www.openxcell.com/blog/system-software/

Harris, S. (s.d.). Hardware Description Languages. *ScienceDirect*. Obtido de Hardware Description Languages: https://www.sciencedirect.com/topics/computer-science/hardware-description-languages

Heaton, R. (27 de 03 de 2014). *How does HTTPS actually work?* Obtido de Robert Heaton: https://robertheaton.com/2014/03/27/how-does-https-actually-work/

HeidiSQL. (28 de 09 de 2022). *What's this?* Obtido de heidiSQL: https://www.heidisql.com/

IBM. (19 de 03 de 2021). *IBM.* Obtido de https://www.ibm.com/docs/en/iad/7.2.1?topic=c-clientserver-architecture

IBM Corporation. (03 de 09 de 2022). *How to squeeze billions of transistors onto a computer chip.* Obtido de IBM Portal: https://www.ibm.com/thought-leadership/innovation-explanations/mukesh-khare-on-smaller-transistors-analytics

IG Global. (02 de 07 de 2022). *What is Authoring Language.* Obtido de What is Authoring Language: https://www.igi-global.com/dictionary/authoring-language/1895

Intel Corporation. (13 de 05 de 2006). ATX Specification. *ATX Specification - Version 2.01*, pp. 1-31. Obtido de ATX Specification: https://web.aub.edu.lb/pub/docs/atx_201.pdf

Intel Corporation. (01 de 01 de 2012). *Making of a Chip.* Obtido de Intel: https://download.intel.com/newsroom/kits/chipmaking/pdfs/Sand-to-Silicon_22nm-Version.pdf

Intel Corporation. (8 de 06 de 2017). *X86: Approaching 40 and Still Going Strong.* Obtido de Intel: https://newsroom.intel.com/editorials/x86-approaching-40-still-going-strong/

Intel Corporation. (15 de 09 de 2022). *What Is Clock Speed?* Obtido de intel: https://www.intel.com/content/www/us/en/gaming/resources/cpu-clock-speed.html

Intelegain Technologies. (06 de 08 de 2019). *What are web frameworks and why you need them?* Obtido de Medium: https://intelegain-technologies.medium.com/what-are-web-frameworks-and-why-you-need-them-c4e8806bd0fb

Interaction Design Foundation. (23 de 10 de 2022). *Interaction Design.* Obtido de WIMP Definition: https://www.interaction-design.org/literature/book/the-glossary-of-human-computer-interaction/wimp

IONOS. (11 de 01 de 2020). *Imperative programming: Overview of the oldest programming paradigm.* Obtido de Imperative programming: Overview of the oldest programming paradigm: https://www.ionos.com/digitalguide/websites/web-development/imperative-programming/

Jacobson, D. (23 de 09 de 2019). *What was the first computer?* Obtido de The Conversation: https://theconversation.com/what-was-the-first-computer-122164

Jevtic, G. (15 de 05 de 2019). *What is SDLC? Phases of Software Development, Models, & Best Practices.* Obtido de PhenixNAP Global IT Services: https://phoenixnap.com/blog/software-development-life-cycle

Jilani, A. A. (2019). Advances in Applications of Object Constraint Language for Software Engineering. *ScienceDirect*. Obtido de https://www.sciencedirect.com/topics/computer-science/constraint-language

Johnston, J. (07 de 06 de 2021). *What is a CRUD app and how to build one | Ultimate guide.* Obtido de Budibase: https://budibase.com/blog/crud-app/

Juricic, V., & Radosevic, M. (01 de 01 de 2018). Performance Impact of Using Abstractions in Object-Oriented Programming. *Proceedings of the 29th International DAAAM Symposium*. Obtido de https://www.researchgate.net/publication/329460689_Performance_Impact_of_Using_Abstractions_in_Object-Oriented_Programming

Khanra, A. (29 de 04 de 2016). *Freelance Camp.* Obtido de What can a Full Stack developer do for a Small Business?: https://www.freelancermap.com/blog/what-can-a-full-stack-developer-do-for-a-small-business/

Lamprecht, E. (19 de 08 de 2022). *The Difference Between UX and UI Design – A Beginner's Guide.* Obtido de CareerFoundry: https://careerfoundry.com/en/blog/ux-design/the-difference-between-ux-and-ui-design-a-laymans-guide/

Laravel. (17 de 01 de 2022). Obtido de Laravel Get Started: https://laravel.com/docs/8.x/installation

Laravel. (15 de 09 de 2022). *Laravel Introduction.* Obtido de Laravel: https://laravel.com/docs/9.x

Laseur, D. (16 de 08 de 2021). *Simply explained: How software is made.* Obtido de LinkedIn: https://www.linkedin.com/pulse/simply-explained-how-software-made-dylan-laseur

Lee, X. (23 de 11 de 2013). *Python Indentation Syntax Terminology: "Off-Side Rule".* Obtido de Python Indentation Syntax Terminology: "Off-Side Rule": http://xahlee.info/comp/python_indentation_syntax_off-side_rule.html

Letendart, M. (28 de 03 de 2018). *What is Web Development?* Obtido de OpenClassRooms: https://blog.openclassrooms.com/en/2018/03/28/web-development-definition/

Lin, C. (20 de 07 de 2019). *Array Languages.* Obtido de SpringerLink: https://link.springer.com/referenceworkentry/10.1007/978-0-387-09766-4_25

Luerweg, T. (2015). Stack based programming paradigms. Concepts of Programming Languages. *Stack based programming paradigms. Concepts of Programming Languages*, p. 1.

Lutkevich, B. (02 de 03 de 2021). *Code Obfuscation.* Obtido de TechTarget: https://www.techtarget.com/searchsecurity/definition/obfuscation

Mack, C. A. (20 de 01 de 2011). Fifty Years of Moore's Law. *Fifty Years of Moore's Law*, pp. 1-2.

Madurai, V. (17 de 02 de 2018). *Web Evolution from 1.0 to 3.0.* Obtido de Medium: https://medium.com/@vivekmadurai/web-evolution-from-1-0-to-3-0-e84f2c06739

Malvik, C. (07 de 13 de 2020). *9 Programming Careers for Coding Connoisseurs*. Obtido de Rasmussen University: https://www.rasmussen.edu/degrees/technology/blog/programming-careers-for-coding-connoisseurs/

Malvik, C. (13 de 07 de 2020). *Rasmussen University - 9 Programming Careers for Coding Connoisseurs.* Obtido de Rasmussen University: https://www.daaam.info/Downloads/Pdfs/proceedings/proceedings_2018/130.pdf

Mashey, J. (01 de 01 de 2009). *The Long Road to 64 Bits.* Obtido de ACM: https://cacm.acm.org/magazines/2009/1/15667-the-long-road-to-64-bits/fulltext

Mendix. (26 de 09 de 2022). *Mendix.* Obtido de Mendix: https://www.mendix.com/

Merrian Webster. (23 de 09 de 2022). *WYSISWYG.* Obtido de Merrian Webster: https://www.britannica.com/dictionary/WYSIWYG

Metzner, J., & Barnes, B. (2014). *A minha biblioteca.* Academic Press.

Michael, S. S. (07 de 02 de 2019). *What Is a Microarchitecture?* Obtido de About Circuits: https://www.allaboutcircuits.com/technical-articles/what-is-a-microarchitecture-processor-register-files-ARM-core/

Microsoft. (25 de 02 de 2022). *Visual Studio Code*. Obtido de Visual Studio Code: https://visualstudio.microsoft.com/

Microsoft Corporation. (23 de 09 de 2022). *Code editing.* Obtido de Microsoft: https://code.visualstudio.com/

Microsoft Corporation. (23 de 09 de 2022). *Microsoft Power Apps.* Obtido de Microsoft Power Apps: https://powerapps.microsoft.com/pt-pt/

MMR. (12 de 09 de 2022). *Memory management in various languages.* Obtido de Memory management in various languages: https://www.memorymanagement.org/mmref/lang.html

Monteiro, T. (13 de 07 de 2022). *What is Abstraction in Programming – And Why is it Useful?* Obtido de FreeCodeCamp Courses: https://www.freecodecamp.org/news/what-is-abstraction-in-programming/

Moreau, J. (13 de 02 de 2019). *Overview of Computer Language Hierarchy*. Obtido de ScienceAid: https://scienceaid.net/Overview_of_Language_Hierarchy

Mozilla Developers. (07 de 10 de 2022). *High-level programming language.* Obtido de Mozilla Developers: https://developer.mozilla.org/en-US/docs/Glossary/High-level_programming_language?retiredLocale=pt-PT

Mozilla Developers. (30 de 09 de 2022). *Progressive web apps (PWAs).* Obtido de Mozilla Developers: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps

Mozilla Developers. (23 de 09 de 2022). *UTF-8.* Obtido de Mozilla Developers: https://developer.mozilla.org/en-US/docs/Glossary/UTF-8

Munoz, D. (02 de 03 de 2022). *After All These Years, the World is Still Powered by C Programming.* Obtido de TopTal: https://www.toptal.com/c/after-all-these-years-the-world-is-still-powered-by-c-programming

National Geographic Society. (20 de 04 de 2022). *Sep 9, 1947 CE: World's First Computer Bug.* Obtido de National Geograhoc: https://education.nationalgeographic.org/resource/worlds-first-computer-bug

NIST. (24 de 09 de 2022). *man-in-the-middle attack (MitM).* Obtido de NIST: https://csrc.nist.gov/glossary/term/man_in_the_middle_attack

NIST CSRC. (6 de 10 de 2022). *man-in-the-middle attack (MitM).* Obtido de Computer Security Resource Center: https://csrc.nist.gov/glossary/term/man_in_the_middle_attack

Ombeni, M., & Thomas, E. (01 de 01 de 2016). Web Programming - II (Server Side Scripting). *Web Programming - II (Server Side Scripting)*, p. 1. Obtido de https://oasis.col.org/items/e83b0a25-28a9-4f04-a1f9-e3a423947087

Oracle Corporation. (27 de 09 de 2022). *Oracle Business Process Management.* Obtido de Oracle: https://www.oracle.com/middleware/technologies/bpm.html

OurWorldInData Stats. (30 de 09 de 2022). *Our World in Data.* Obtido de Our World in Data: https://ourworldindata.org/coronavirus

OutSystems. (26 de 09 de 2022). *OutSystems.* Obtido de OutSystems: https://www.outsystems.com/

OutSystems Corporation. (12 de 09 de 2022). *OutSystems.* Obtido de What Is Low-Code? : https://www.outsystems.com/guide/low-code/

P., E. (07 de 09 de 2022). *What Are Programming Language Generations?* Obtido de EasyTech: https://www.easytechjunkie.com/what-are-programming-language-generations.htm

Papermaster, M. (26 de 07 de 2015). *Improving the Energy Efficiency of Computing as Moore's Law Slows .* Obtido de GTM: https://www.greentechmedia.com/articles/read/improving-the-energy-efficiency-of-computing-while-moores-law-slows

Patrizio, A. (23 de 09 de 2022). *Single-core vs. multi-core CPUs.* Obtido de Network World: https://www.networkworld.com/article/3673231/single-core-vs-multi-core-cpus.html

PHP. (14 de 01 de 2022). *PHP in_array()*. Obtido de PHP Documentation: https://www.php.net/manual/en/function.in-array.php

Pratt, M. (20 de 09 de 2022). *Definition.* Obtido de TechTarget: https://www.techtarget.com/searchsoftwarequality/definition/low-code-no-code-development-platform

RENTEC Digital. (24 de 12 de 2021). *Top Web Development Technologies and Frameworks to use in 2021*. Obtido de RENTEC Digital: https://rentechdigital.com/top-web-development-technologies-and-frameworks

RFC 9110. (01 de 06 de 2022). *RFC 9110.* Obtido de HTTP Semantics: https://www.rfc-editor.org/rfc/rfc9110.html

Roose, K. (02 de 09 de 2022). *An A.I.-Generated Picture Won an Art Prize. Artists Aren't Happy.* Obtido de New York Times: https://www.nytimes.com/2022/09/02/technology/ai-artificial-intelligence-artists.html

Rutgers School of Sciences. (02 de 07 de 2012). *Parallel/Concurrent Languages:.* Obtido de Parallel/Concurrent Languages:: https://people.cs.rutgers.edu/~venugopa/parallel_summer2012/ParallelConcurrentLanguages.html

Saylor Academy. (23 de 09 de 2022). *Generations of Programming Languages.* Obtido de Saylor Academy: https://learn.saylor.org/mod/book/view.php?id=30914&chapterid=6731

Scaler Academy. (11 de 01 de 2022). *Scaler Academy.* Obtido de Difference Between C and Python: https://www.interviewbit.com/blog/difference-between-c-and-python/

Schäferhoff, N. (10 de 08 de 2022). *How to Create a Website.* Obtido de WebsiteSetup: https://websitesetup.org/

Scott, M. L. (02 de 09 de 2009). *Science Direct.* Obtido de Assembly Language: https://www.sciencedirect.com/topics/computer-science/assembly-language

SCRUM.org. (06 de 01 de 2022). *What is Scrum?* . Obtido de Scrum: https://www.scrum.org/resources/what-is-scrum

Sensus Corporation. (28 de 09 de 2022). *SENSUS BPM ONLINE.* Obtido de Sensus: https://sensus-processmanagement.com/homepage/?lang=en

Shawn, J. (11 de 02 de 2021). *No Code / Codeless / Low Code testing: What's the difference?* Obtido de Testuim Company: https://www.testim.io/blog/low-code/

Shilov, A. (11 de 03 de 2022). *The Tech Behind Apple's M1 UltraFusion Chip Interconnect.* Obtido de Tom's Hardware: https://www.tomshardware.com/news/apple-uses-cowos-s-to-build-m1-ultra

Shopbox Software. (30 de 09 de 2022). *Shopbox Software.* Obtido de Shopbox Software: https://shopboss.net/

Shukla, S. (25 de 02 de 2021). *What Makes Laravel Framework the Best Choice for PHP Web Development?* Obtido de Net Solutions: https://www.netsolutions.com/insights/laravel-framework-benefits/

Simicart Compilations. (01 de 03 de 2022). *12 Best Examples of Progressive Web Apps (PWAs) in 2022.* Obtido de Simicart: https://www.simicart.com/blog/progressive-web-apps-examples/

Simon, S. (06 de 08 de 2019). *The Language of Computers Part 1 – A Beginner's Guide to the Binary.* Obtido de The Language of Computers Part 1 – A Beginner's Guide to the Binary: https://www.simonandsimon.co.uk/blog/the-language-of-computers-part-1-a-beginners-guide-to-the-binary

Smith, J. (27 de 08 de 2022). *Compiler Vs. Interpreter: What's the Difference?* Obtido de Guru99: https://www.guru99.com/difference-compiler-vs-interpreter.html

SoftwareTesting. (25 de 09 de 2022). *10 Best Low-Code Development Platforms in 2022.* Obtido de SoftwareTesting: https://www.softwaretestinghelp.com/low-code-development-platforms/

Sousa, T. B. (03 de 02 de 2021). Dataflow Programming - Concept, Languages and Applications. *Dataflow Programming - Concept, Languages and Applications*, p. 1.

Spotify. (30 de 09 de 2022). *Spotify.* Obtido de Spotify: https://open.spotify.com/

Statcounter. (10 de 09 de 2022). *Browser Market Share Worldwide.* Obtido de Statcounter: https://gs.statcounter.com/browser-market-share

Statista. (01 de 07 de 2022). *Most used web frameworks among developers worldwide, as of 2022* . Obtido de Statista: https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web/

Statista. (23 de 09 de 2022). *Most used web frameworks among developers worldwide, as of 2022* . Obtido de Statista: https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web/

Stephen, B. J. (05 de 03 de 2017). *What is an operating system?* Obtido de What is an operating system?: https://www.techtarget.com/whatis/definition/operating-system-OS

T, B., & S, P. (01 de 01 de 2013). *A Survey on Software Development Life.* Obtido de International Journal of Computer Science and Mobile Computing: https://d1wqtxts1xzle7.cloudfront.net/53623042/V2I5201384-with-cover-page-v2.pdf?Expires=1646311147&Signature=PvnW4JGVJABhxhq8HzLwhS2~g3No1FM2GonRC~wq3IZkOBrmvP5CTxn1R8Mpe709dlf4k54P7BFnvbQS~uXY13QN9XRqePCC0ArlleulUZdG5eYYp3ckv~365-Z1RPpeBCShIXA4sGhEMAIy2

Technical Study. (10 de 02 de 2022). *Why is Laravel the most popular framework of 2022 (Feature and Benefits).* Obtido de Why is Laravel the most popular framework of 2022 (Feature and Benefits): https://technicalistechnical.com/laravel-framework/

Techopedia. (17 de 09 de 2022). *What Does System Programming Mean?* Obtido de What Does System Programming Mean?: https://www.techopedia.com/definition/9616/system-programming

Tran, T. (17 de 12 de 2021). *Top 6 Object-Oriented Programming Languages.* Obtido de Top 6 Object-Oriented Programming Languages: https://www.orientsoftware.com/blog/list-of-object-oriented-programming-languages/

Transcend Corp. (14 de 10 de 2022). *Why is the actual storage capacity of the product slightly less than the volume specified on the product's package?* Obtido de Transcend: https://www.transcend-info.com/Support/FAQ-503

Triggs, R. (24 de 04 de 2022). *Arm vs x86: Instruction sets, architecture, and all key differences explained.* Obtido de AndroidAuthority: https://www.androidauthority.com/arm-vs-x86-key-differences-explained-568718/

Unadkt, J. (15 de 11 de 2019). *Understanding the Role of Rendering Engine in Browsers.* Obtido de Browser Stack: https://www.browserstack.com/guide/browser-rendering-engine

University of Rhode Island. (03 de 02 de 2005). *How Computers Work: The CPU and Memory.* Obtido de How Computers Work: The CPU and Memory: https://homepage.cs.uri.edu/faculty/wolfe/book/Readings/Reading04.htm

University of Texas. (20 de 01 de 2015). *University of Texas*. Obtido de What is a Compiler?: https://lambda.uta.edu/cse5317/notes/node3.html

University of Toronto. (23 de 09 de 2022). *What Is a Full Stack Developer & What Do They Do?* Obtido de Bootcamp: https://bootcamp.learn.utoronto.ca/blog/what-is-a-full-stack-developer/

Verma, S. (13 de 05 de 2020). *Difference between High Level and Low level languages*. Obtido de ProgrammerBay: https://programmerbay.com/difference-between-high-level-language-and-low-level-language/

Vodafone International. (23 de 09 de 2022). *What is codeless software?* Obtido de Vodafone: https://www.vodafone.com/business/news-and-insights/glossary/what-is-codeless-software

Vue.JS. (21 de 02 de 2022). *Introduction*. Obtido de Vue Oficial Documentation: https://vuejs.org/guide/introduction.html

W3C. (18 de 09 de 2022). *XML Essentials.* Obtido de XML Essentials: https://www.w3.org/standards/xml/core

W3Techs. (23 de 09 de 2022). *Usage statistics of client-side programming languages for websites.* Obtido de W3Techs: https://w3techs.com/technologies/overview/client_side_language

W3Techs. (03 de 09 de 2022). *Usage statistics of server-side programming languages for websites.* Obtido de W3Techs: https://w3techs.com/technologies/overview/programming_language

Wallack, M. (01 de 09 de 2019). *How to choose the best talent development tool.* Obtido de Gloat: https://gloat.com/blog/how-to-choose-the-best-talent-development-tool-for-your-employees/

WatEletronics Corporation. (05 de 01 de 2021). *What is a Programming Language and Different Types.* Obtido de WatEletronics: https://www.watelectronics.com/types-of-programming-languages-with-differences/

Wattanajantra, A. (08 de 07 de 2020). *Software ERP. Vertical ou horizontal?* Obtido de SAGE: https://www.sage.com/pt-pt/blog/software-erp-vertical-ou-horizontal/

Webflow. (02 de 03 de 2020). *What is no-code development?* Obtido de Webflow2020: https://webflow.com/no-code

Wikimedia. (02 de 10 de 2022). Obtido de Wikimedia: https://commons.wikimedia.org/wiki/File:ASCII-Table.svg

WordPress. (26 de 09 de 2022). *WordPress.* Obtido de WordPress: https://pt.wordpress.org/

Yale University. (03 de 04 de 2006). *x86 Assembly Guide.* Obtido de yale University: https://flint.cs.yale.edu/cs421/papers/x86-asm/asm.html

Yee, A. (02 de 09 de 2021). *How much RAM do you need in a laptop?* Obtido de PCWorld: https://www.pcworld.com/article/395018/how-much-ram-do-you-need-in-a-laptop.html