



Introduction to Software Design and Architecture

Part. 1

Fundamental

Software, like any other complex structure, should have a solid and stable foundation; otherwise it can fail to support and perform existing and future requirements, or create difficulties during the deployment or management in production environment. Current platforms and tools allow simplifying the process of building applications, but while designing a system it is important to take into consideration end-user, system and IT infrastructure as a whole, plus pre-set business objectives.

Software architecture

Software architecture is the process of determining a well-defined solution that complies with all the operational and technical requirements and on the other hand improves such common quality attributes as security, accessibility and performance. Whereas software design is the process of applying software solutions to one or more set of problems.

The major goal of application architecture is to analyze business and technical requirement, understand use cases and prepare the ways to apply those use cases in software. Good architecture determines requests that affect an application structure and, due to its flexibility, lessens business risks related to the development of technical solution. Apart from that software architecture deals with quality and functional requirement; realizes use-cases and scenarios and exposes the system structure while hiding the executing details.

Software architecture types

Architectural artifacts are introduced to define a system, solution or enterprise state. In a nutshell it is a set of scenarios, views, models, trends, goals, objectives, guidelines, attributes, application components, databases and other objects and architecture documents that guide architecture development process, outline typical architectural deliverables and manage architectural capabilities.

- Infrastructure
- Network
- Process
- Security
- Software
- Network
- Application
- Hardware
- Business
- Technical
- Enterprise
- Database
- Data
- Solution
- Integration

Viewpoints

Software architecture can be represented from a number of viewpoints that in return can be organized to form a holistic system view. The amount of architectural viewpoints ranges from 3 or 5 (“4+1” model by Kruchten) and up to 36 (“Zachman Framework”). Among other examples of architecture frameworks are, just to name a few: TOGAF, DoDAF, RM-ODP and others.

Requirements

- Functional requirements — combination of constraints to the system, specifically to its behavior or reaction to runtime stimuli and undertaken actions;
- Non-Functional requirements — a set of properties of the functional requirements (functional performance rate, level of fault tolerance to erroneous input, etc.) or of the product as the whole (product deployment time, maintenance costs limits, etc.);
- Constraints — fixed design decisions with no proposed means for customization or reuse. They are mainly specified formally or textually and play an important role in architecture validation and design decision documentation.

Non-Functional requirements

Non-Functional requirements (quality attributes) represent different areas of concern and influence runtime behavior, user experience and system design. They depend on a project and defined needs, but to name a few they are: availability, scalability, performance, security, testability, monitoring, extensibility and so on.

Architectural styles

- Approaches to software development are widely-used while planning complex software architecture and design. Such approaches are called architectural styles or patterns and they include a combination of principles for outlining an abstract framework for system family. Architectural style should be chosen separately for an appropriate application according to style's key principles, major benefits, overview and so on. The most common architectural styles are: DDD (Domain Driven Design), TDD (Test Driven Development) and BDD (Behavior Driven Development).

DDD vs TDD vs BDD

Domain Driven Design (DDD) is an object-based paradigm for software design on basis of business domain, its components and behaviors, and functional dependence between them;

Test Driven Development (TDD) is an innovative technology of applying automated unit tests to drive software design and force dependencies' decoupling;

Behavior Driven Development (BDD) is a methodology for designing software solutions that actually meet business requirements. Such results are achieved through the implementation of model driven approach that initiates the designing process with business strategy, objectives and constraints and converts them into a software solution.

Artifacts

Collecting all the necessary artifacts, listing the requirements, setting up technology specifications and planning a scenario, it is important to determine the appropriate application type that will meet all the pre-set constraints. Architectural archetypes are usually divided into four basic types that, in return, can be combined into other more specific application types. The most common architecture archetypes are:

- Web applications are mainly designed to support connected scenarios and various browsers running on a series of platforms and operational systems;
- Mobile applications can be designed as rich or thin (web) client applications; where rich client can support occasionally-connected or disconnected scenarios and web apps support only connected scenarios;
- Rich Client applications are usually developed as free-running apps with advanced user-interface functionality, improved responsiveness, better user experience and capability to support occasionally-connected or disconnected scenarios;
- Service applications are designed to loose coupling between server and a client, and support interoperability. This type of apps can be consumed by various and emergent apps, but it depends on network connectivity.



Architectural slicing

Software architecture of an object-oriented system determines its' high-level design structure. With increase in complexity and size of software systems and remarkable increase in importance of architectural design model the concept of “architectural slicing” becomes widely-used for partitioning large architectures into transparent portions. Such approach helps to make reliability prediction analysis report basing on architectural models, run impact analysis, facilitate complex architectures and even compute different types of metrics to define software architectures.

Software architecture layers

- Architecture patterns — to define the overall structure and shape of software app

Architecture relevant to software constraints and objectives.

- Modules and interconnections, like packages, components, classes and design patterns.


Object Oriented Design

The third layer is the basis for manageable and flexible software system. The process of planning a software system with interoperable objects is called Object Oriented Design (OOD) and it consists of techniques and principles that help to manage module dependencies, introduce changes to modules, without modifying the source code and in general make the system smart. OOD principles are usually categorized as: SOLID and GRASP.

S.O.L.I.D. stands for the first 5 OOD principles:

- S — Single-responsibility principle (SRP)
- O — Open/closed principle (OCP)
- L — Liskov substitution principle (LSP)
- I — Interface segregation principle (ISP)
- D — Dependency inversion principle (DIP)

GRASP



GRASP is an acronym for General Responsibility Assignment Software Patterns. GRASP uses principles and patterns, such as: Creator, Controller, High Cohesion, Indirection, Information Expert, Polymorphism, Low Coupling, Pure Fabrication, Protected Variations. These core patterns answer some software problems and such problems are of frequent occurrence to software development cycles.

Other basic principles

Other basic principles that can make software development process easier are:

- KISS — this abbr. stands for “Keep It Simple, Stupid!”;
- YAGNI — acronym that means “You Aren’t Gonna Need It”;
- DRY — stands for “Don’t Repeat Yourself”

Design patterns

Domain of modules and interconnections also contains design patterns that are aimed to solve widely occurring problems while designing a software application or system. These patterns are not ready-to-use design portions; it is more like a template or description of the way how the problem can be solved in various situations.

Design patterns can activate the development process, prevent hidden issues and enhance code readability.

The most common design patterns

Gang of Four (GoF) — consists of 23 design patterns and in return these patterns are categorized into 3 main sections: creational, structural, behavioral;

Patterns of Enterprise Application Architecture (PoEAA) — cover such patterns as: Domain Logic Patterns, Data Source Architectural Patterns, Object-Relational Behavioral Patterns, Object-Relational Structural Patterns and others;

Cloud Design Patterns & Practices — include 24 design patterns and 10 relevant guidance topics for cloud applications. To name a few: Cache-Aside Pattern, Circuit Breaker Pattern, Compensating Transaction Pattern, Gatekeeper Pattern and so on.

MV* Patterns — stand for Model-View-Controller (MVC), Model-View-Presenter (MVP) and Model-View-ViewModel (MVVM).

Command and Query Responsibility Segregation (CQRS) Pattern — can enhance scalability, security, and performance; preclude update commands from causing cohesion conflicts; support system evolution over time through better flexibility.

Software craftsmanship

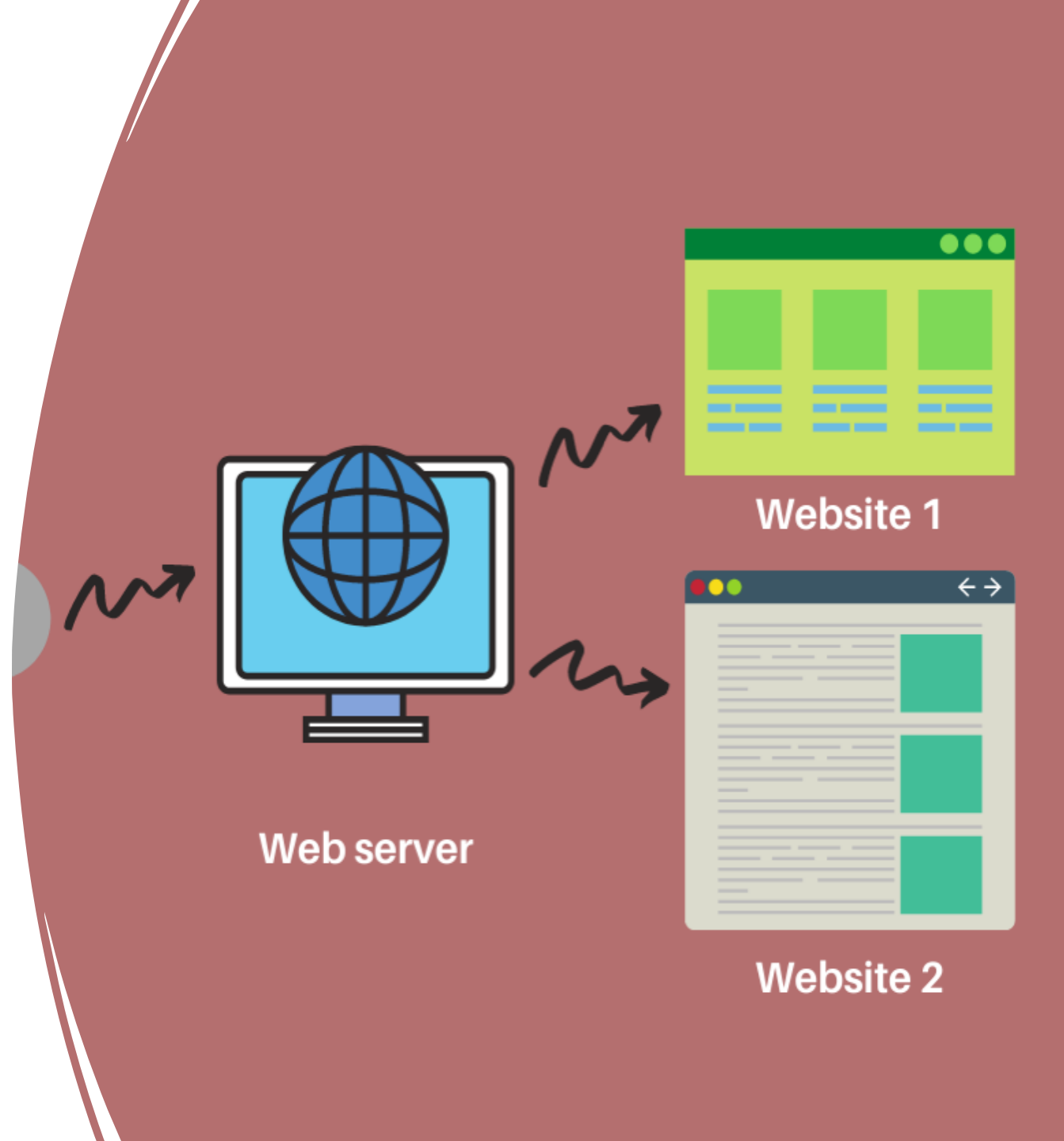
Nowadays software craftsmanship in combination of agile project management enhance the quality of software systems, they help to develop better software that meets business needs completely, manage time and budget limits and still there is a number of problems and issues that every project faces. Software architecture helps to create a well-defined picture and road map for the team, ensures better coordination, provides logicality of standards and approaches, outlines framework for specifying and leaning risks and much more.

Introduction to Web Development

Web development is basically the creation of website pages — either a single page or many pages. For instance, Facebook is a culmination of many webpages together. While single page websites often require endless scrolling and hence is relatively uncommon.

How do websites work?

- Just like the files that are stored in your computer, websites are basically files that are stored in another computer called the **server**. Many servers are together connected through the internet. And this is why, when you open a link, that particular file from that server gets loaded on your computer and you see that particular website.
In this case, since you are the one who is accessing the server, you are the client. So, your browser acts as the client in this scenario.



What are frontend and backend?

- Frontend and Backend basically depend on the kind of relationship **you** as a **client** has with the **server**.
- **Frontend**, as the name suggests, you will deal with what is in front of the screen, that is, when on the **client's** side.
- The same way, **backend** deals with what is behind the screen, as in the **server**, which you cannot see.

FRONTEND

- Users see it
- 20% of total effort

BACKEND

- Users don't see it
- 80% of total effort
- Repetitive



Uses markup and web languages like HTML, CSS, and Javascript

Uses programming and scripting languages like Python, Ruby, and Perl.



HTML



HTML + CSS



HTML + CSS
+ JAVASCRIPT

Learning the basics — HTML, CSS and JavaScript

- Three basic components make every website. Those are: HTML, CSS, and JavaScript.

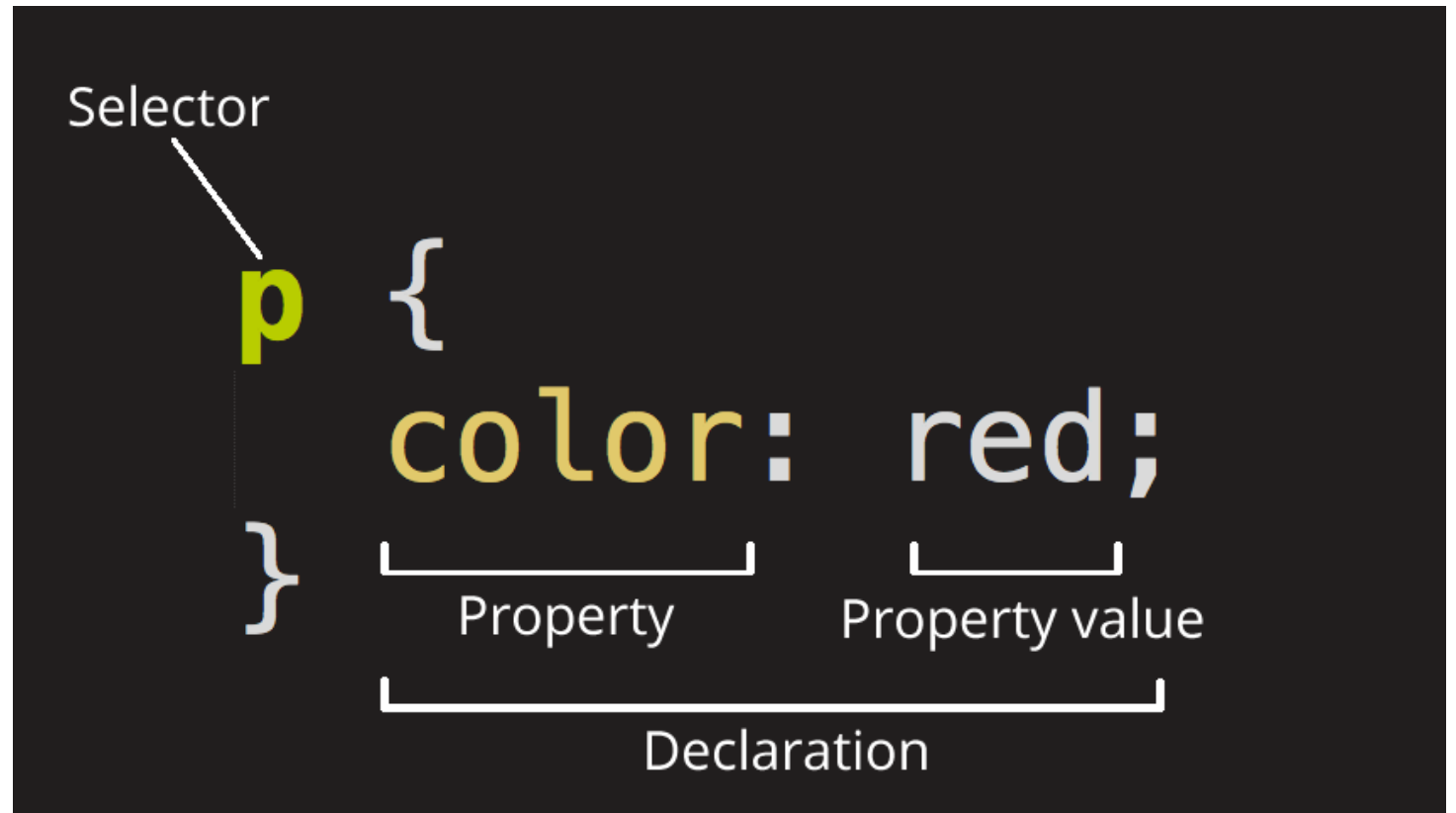
HTML

Hyper Text Markup Language, **HTML** is like the mother of all the websites. In other words, it is the main file that is loaded on your browser when you look at a website. The most basic website can be created by simply using just HTML and no other kind of file.

```
1 <!DOCTYPE html>
2 <html>
3     <head>
4         <title>Example</title>
5         <link rel="stylesheet" href="sty
6     </head>
7     <body>
8         <h1>
9             <a href="/">Header</a>
10        </h1>
11        <nav>
12            <a href="one/">One</a>
13            <a href="two/">Two</a>
14            <a href="three/">Three</a>
15        </nav>
```

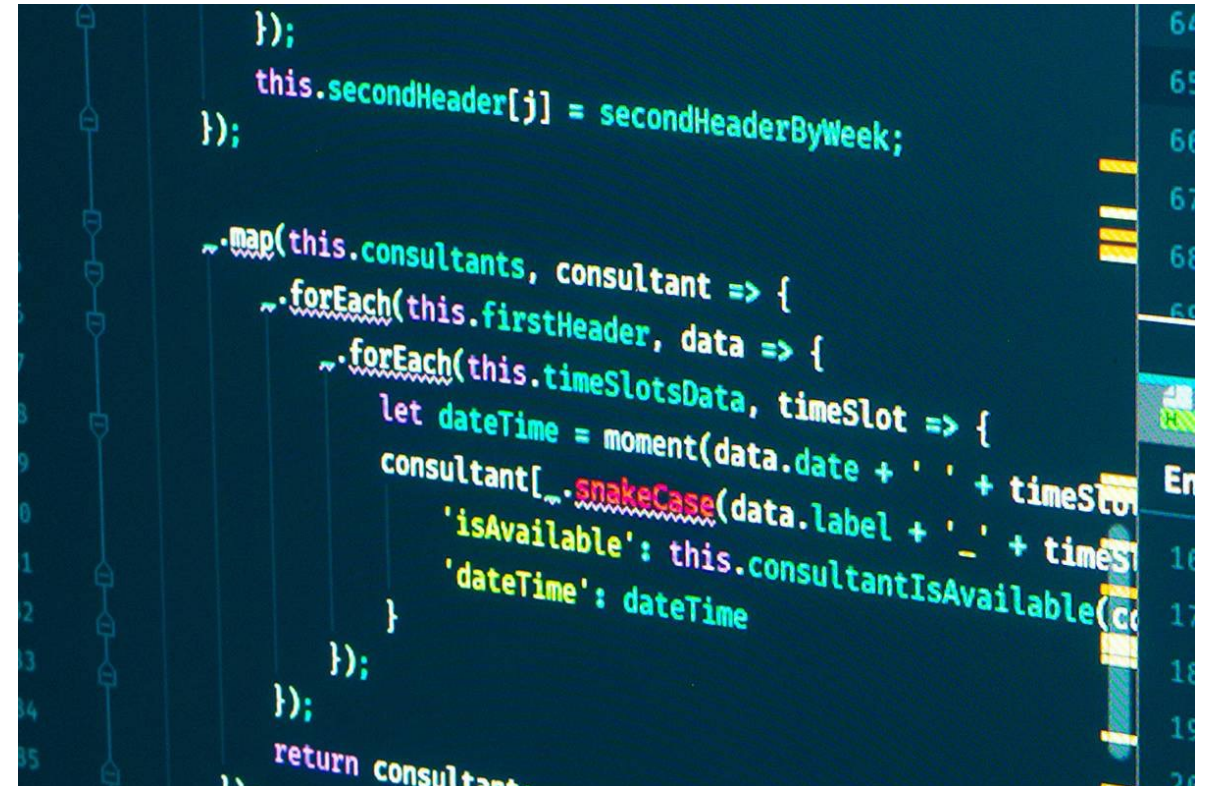
CSS

The Cascading Style Sheets, **CSS**, causes the website to appear more appealing. Without it, any website will look as simple as an MS Word Document. With it, you can add animations and drawings on your website and modify it — the way you want. It acts like the make-up that beautifies your web-page.



JavaScript

- **JavaScript** is a programming language that allows you to interact with certain elements on the website and change them according to your wishes. JavaScript acts like the accessories that are used to adorn the website, thereby making it more appealing.
- CSS adds style to the basic HTML. And the JavaScript adds interactivity with the website and gives a dynamic nature to the website.



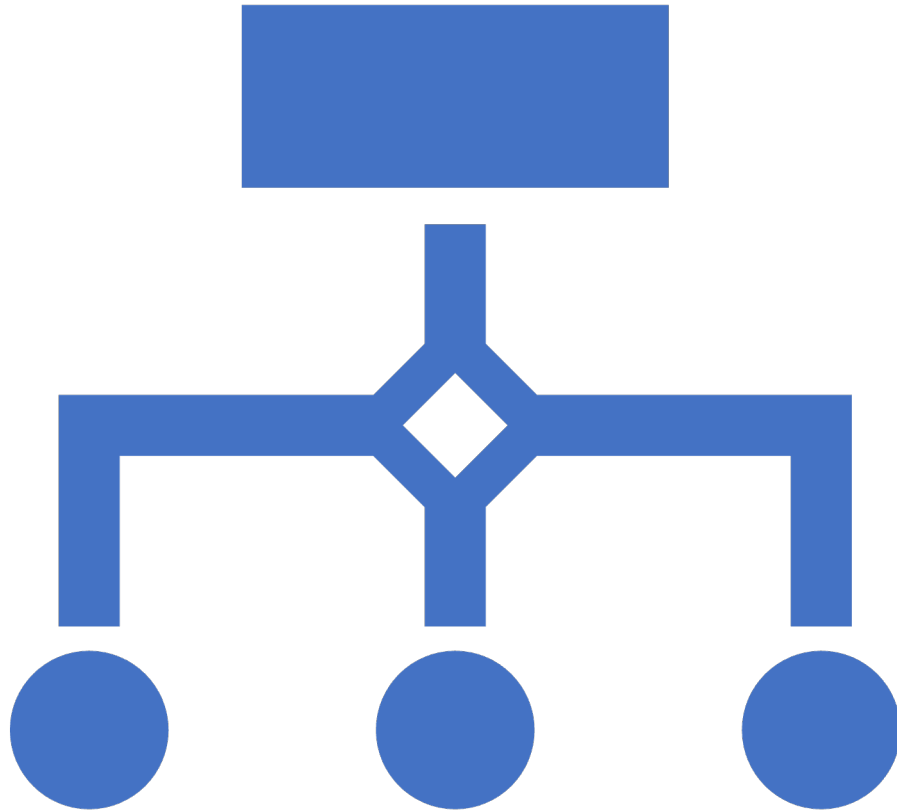
```
});  
this.secondHeader[j] = secondHeaderByWeek;  
});  
  
~.map(this.consultants, consultant => {  
  ~.forEach(this.firstHeader, data => {  
    ~.forEach(this.timeSlotsData, timeSlot => {  
      let dateTime = moment(data.date + ' ' + timeSlot);  
      consultant[~.snakeCase(data.label + ' ' + timeSlot)] = {  
        'isAvailable': this.consultantIsAvailable(consultant, dateTime),  
        'dateTime': dateTime  
      };  
    });  
  });  
});  
return consultant;
```



Code editor

To work with the above-mentioned three types of file, you will have to download a **program** in your computer called the **code editor**.

The type of coder editor you must use depends on the **kind** of code you want to write. For web development, the best-recommended editor, however, is the VS code.



Version control

There is also something called the **Version control**, which is source control. It keeps track of every code that you make in your project file. It enables you to go back to the previous change if you ever make a mistake. You can read more about Version control [here](#) and [here](#).

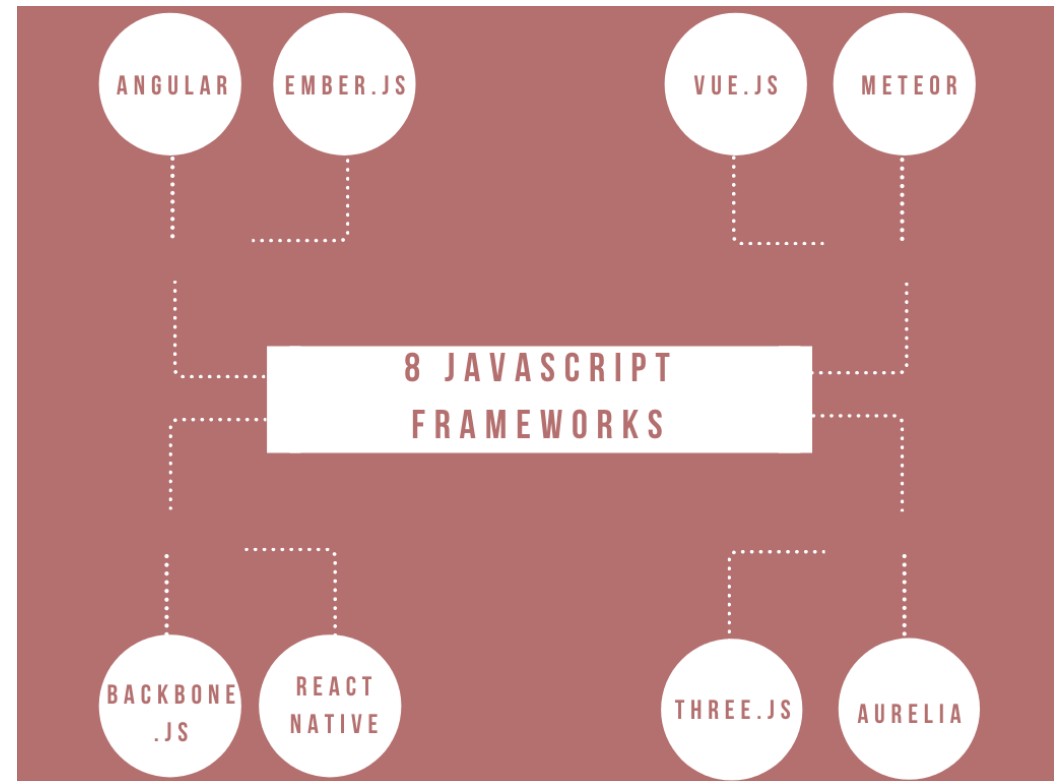
Git

Git is the most used version control system. You can store every file of yours and even change the history in collections.



Frameworks

- A **framework** with which you make a structure. They are like the building blocks created by someone else, and you use these to make your own website, and use them as and how you want.
- To use a framework, you will have to install it on your own website files. After that, you can then add commands on those structures to create the website according to your needs.



JavaScript frameworks and libraries

React, Vue and Angular are some examples of JavaScript front-end frameworks.

After you have understood the basics of working with JavaScript, you can easily learn either one of the JavaScript frameworks.

- **React** was first created by Facebook. It is the most popular framework right now. You can easily learn to react by going to [React.js](https://reactjs.org/) website.
- **Google created Angular**. It was the first big framework ever made.
- **Vue**, on the other hand, is a newer framework created by a former Angular developer, [Evan You](#). Though it is smaller in use as compared to the other two, it is relatively easy and fun to use.

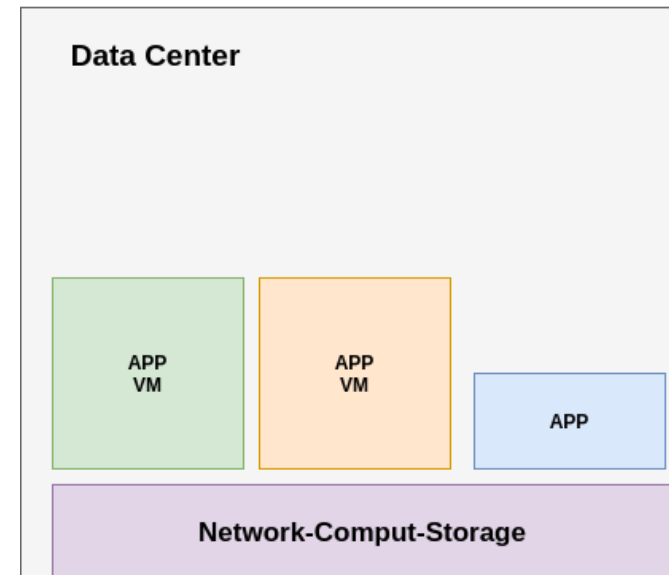
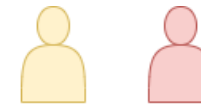
Introduction to Cloud Computing



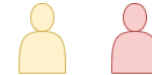
- Transitioning to the cloud is one of the most important things happening in IT today.
- It is a typical organization with a headquarters and some number of branch offices with users in them. Users rely on applications that run directly on the hardware that belongs to their organization.
- Probably running in some on-premise data center, those applications could run on bare metal servers or run inside virtual machines. All the hardware and software assets including network resources storage resources compute resources and the applications running on top of them belong to the organization and reside in a data center.

Data center

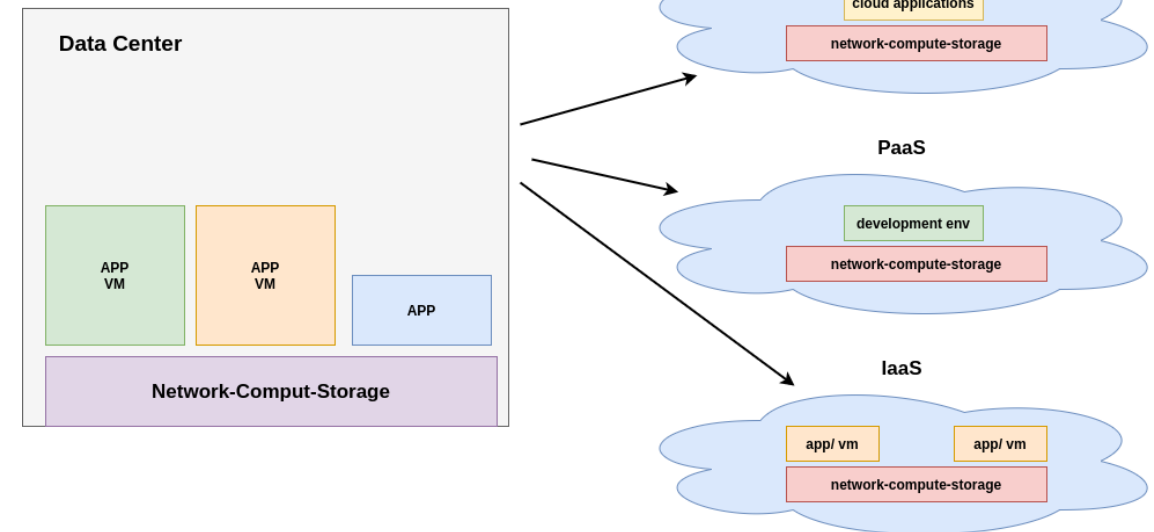
- The organization needs to provide and manage the required power cooling and real estate resources for maintaining the data center.
- The organization could be developing some apps in-house for this to happen, the developers need some development platforms to develop test, and run their software. This needs to be managed by the IT organization meaning new hardware and software assets to procure setup and manage.
- This familiar picture has been so common to us and in use around the world today. With the rising popularity of cloud computing in recent years, this picture is changing in multiple ways.



Benefit from the cloud without doing any upfront investment



- First, organizations can use cloud applications commonly called **Software As A Service (SaaS)** which really means applications running at data centers owned by a cloud operator and accessed via the internet.
- Organizations can also use cloud development platforms which are commonly called **Platform as a Service (PaaS)** for their software developers.
- Developers could consume the tools and the development environment they need from the cloud.
- The third way organizations can leverage the cloud is by running some of their applications on computing resources at data centers across the Internet. This is commonly called the **Infrastructure as a Service (IaaS)** consumption model.



Cloud provides agility and speed with economies of scale benefits

No need to say organizations will still have some applications that run locally. Not every service is consumed from the cloud. There could be different motivations behind that like data security, performance cost, and so on.

But still, the cloud is a viable alternative for writing some of the services and it is here to stay. The biggest promise of cloud computing is the promise of agility and effectiveness coupled with the economies of scale provided by large cloud providers.

Instead of waiting for weeks or months to roll applications users can now access information and applications from a cloud provider.

The organization might build new applications themselves on a cloud platform. At the same time, they could run those in-house built applications or the applications they license from a third party.

IaaS, PaaS, and SaaS are the 3 delivery models of cloud computing.

Cloud computing

- It's a type of internet-based computing that provides shared computer processing resources and data to any type of consumer device on demand.
- It is a model for enabling ubiquitous on-demand access to a shared pool of configurable computing resources. Computer resources could be computer networks, servers, storage, applications, and services that can be rapidly provisioned and released with minimal management efforts.

On-demand self-service

- The first one is having the On-Demand self-service capability, so a consumer can provision computing capabilities such as server network storage or an application by him or herself without requiring human interaction with the service provider.

Broad network access

- This emphasizes the fact that services should be available over the network and accessed through standard mechanisms. Any client platform available like mobile phones, tablets, laptops, or high-end servers could be used for that.

Resource pooling

- Resource pooling means the providers' computing resources are pooled to serve multiple consumers using a multi-tenant model. Different physical and virtual resources could be dynamically assigned and reassigned according to consumer demand.
- There is a sense of location independence in that the consumer generally has no control or knowledge over the exact location of the provided resources but may be able to specify a location at a higher level of abstraction.
- For example country state or data center. Examples of resources include storage, processing capacity, memory, and network bandwidth.

Rapid elasticity

- Meaning resources can be elastically provisions and released in some cases. This is automatic. This provides the ability to scale rapidly outward and inward along with the demand to the consumer.
- The capabilities available for provisioning often appear to be unlimited and can be changed in any quantity at any time.

Measured service

- Cloud systems automatically control and optimize resource usage by leveraging a metering capability at some level of abstraction appropriate to that type of service.
- Depending on the service metered, resources could be storage capacity, processing capacity, bandwidth and active user counts. Resource usage can be monitored controlled and reported, providing transparency for both the provider and consumer of the utility service in the end.

Conclusion

- Cloud computing provides individual users and organizations with various capabilities to store and process their data in third-party data centers that may be located anywhere far from the user ranging in the distance from across the city to across the world.
- Cloud computing relies on sharing of resources to achieve coherence and economy of scale similar to a utility like the electricity grid over an electricity network.