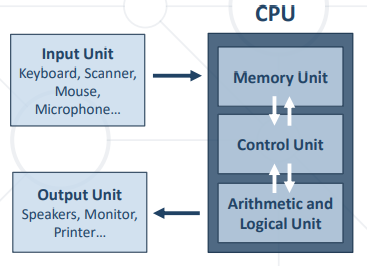
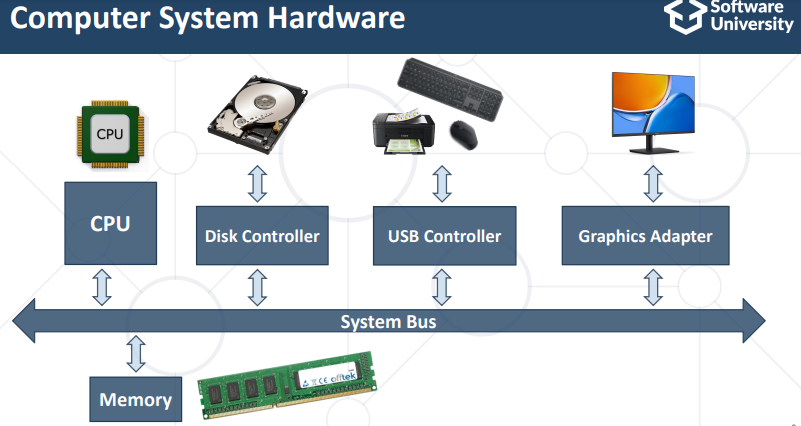
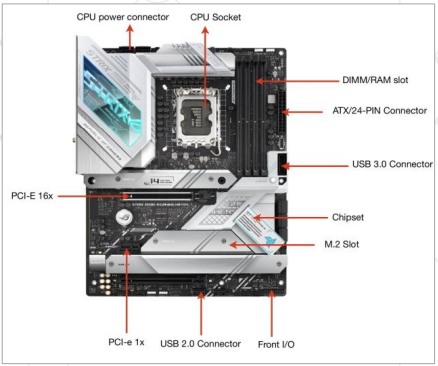
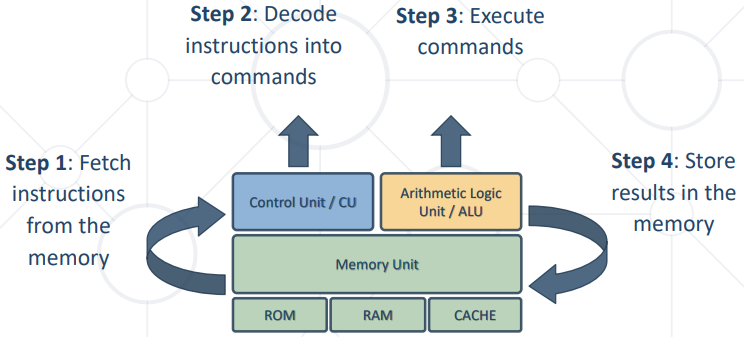
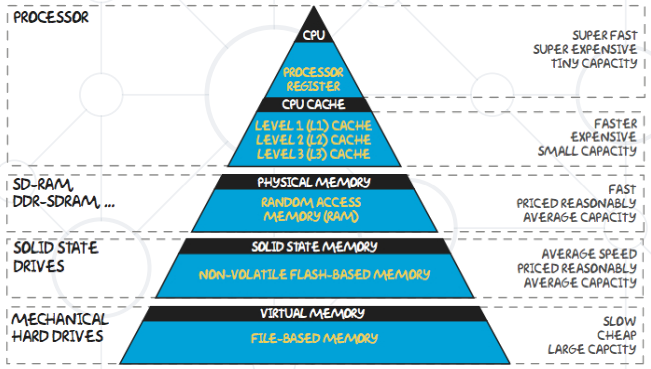
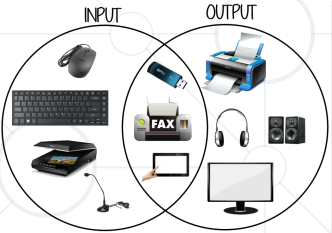
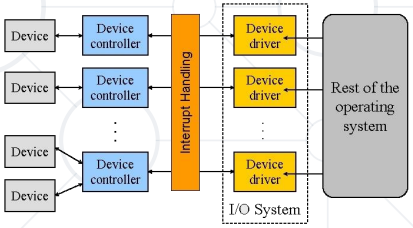
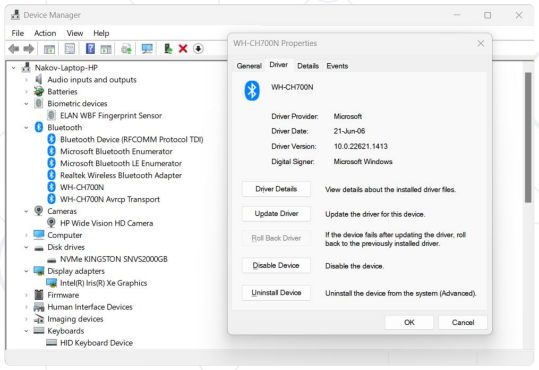
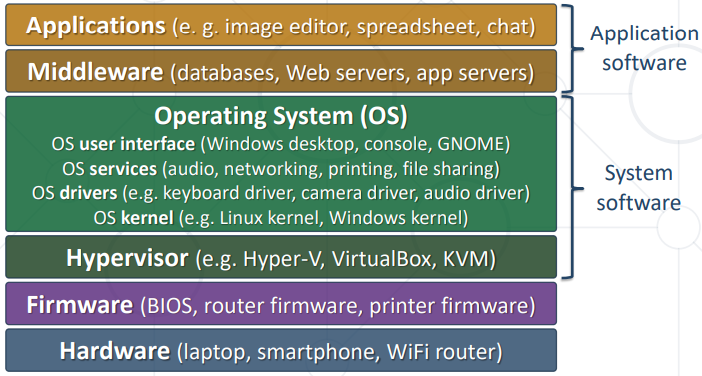
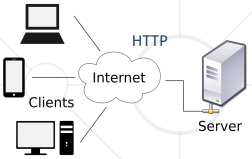
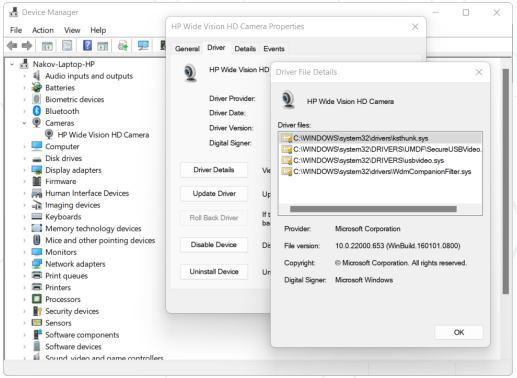
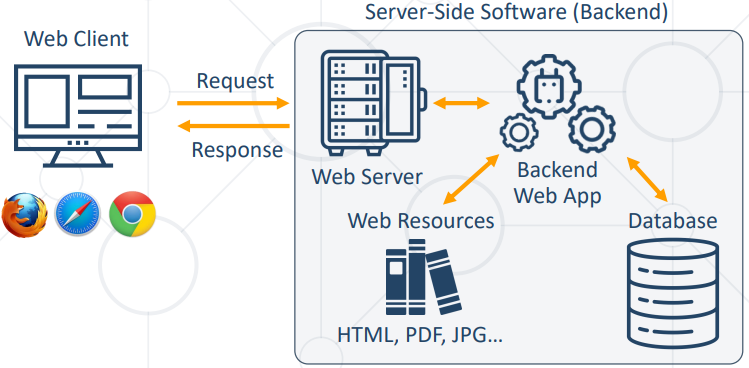
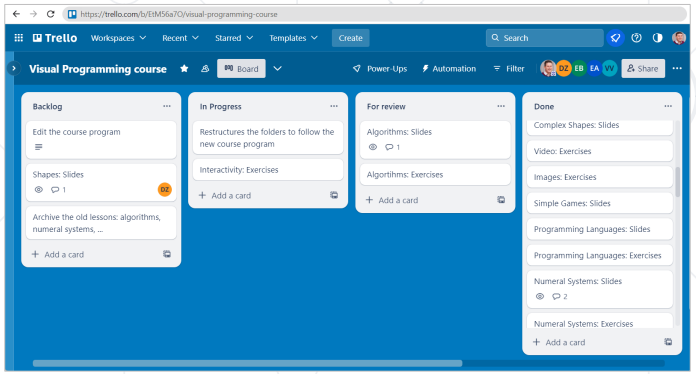
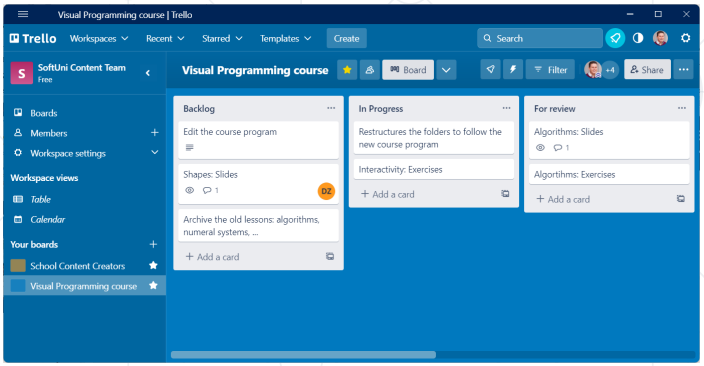
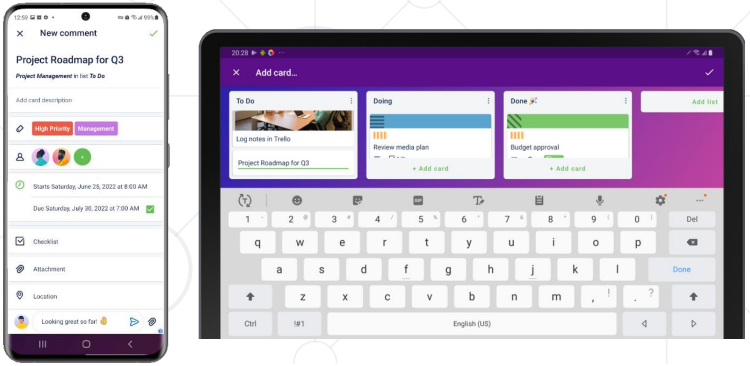
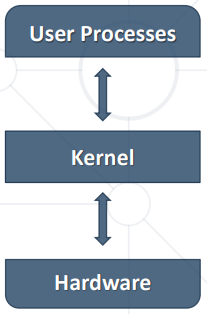
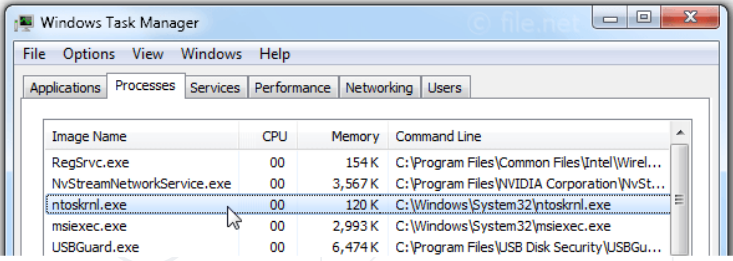
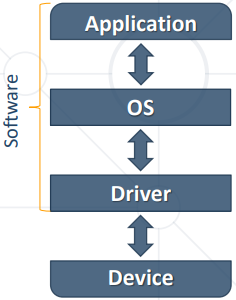
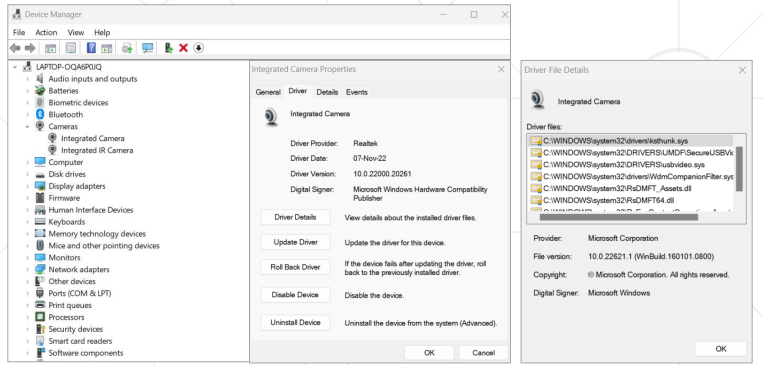
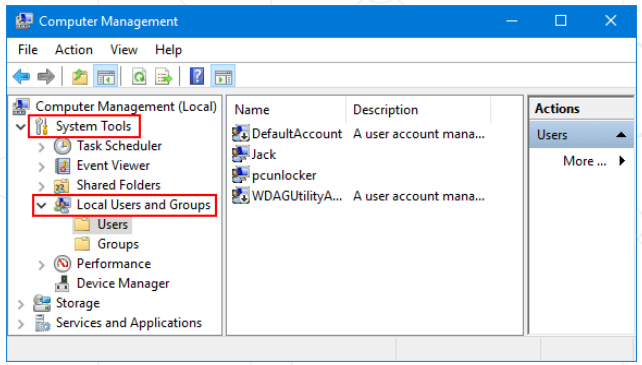
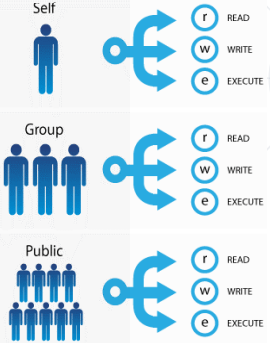
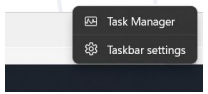
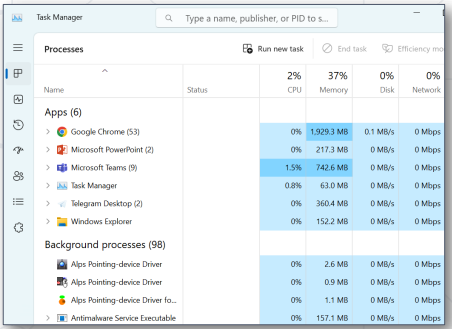
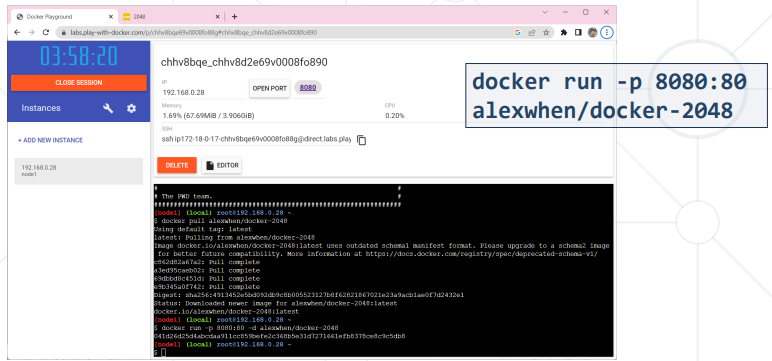
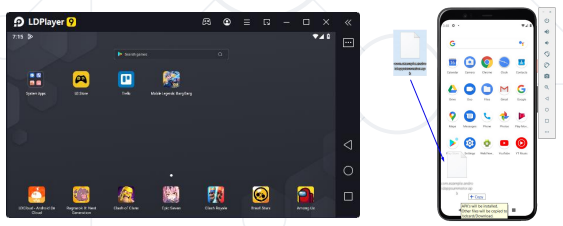
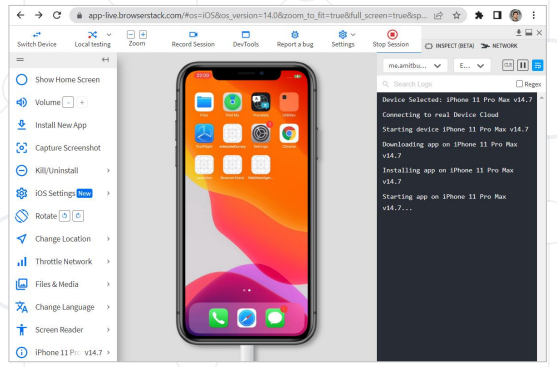
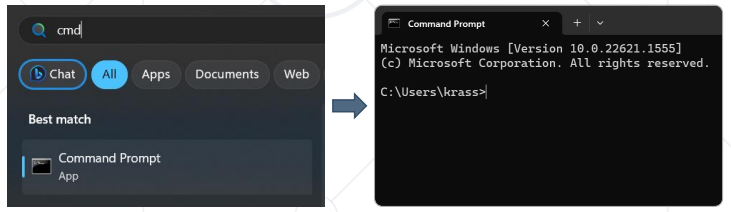
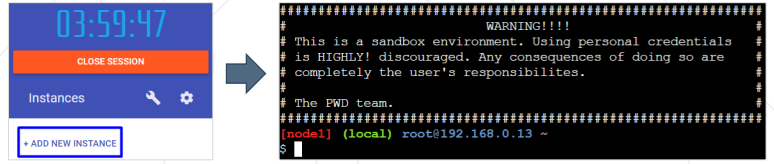
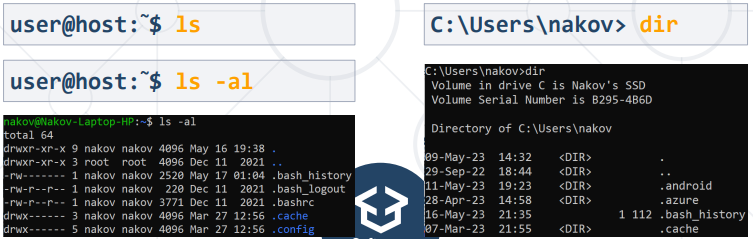
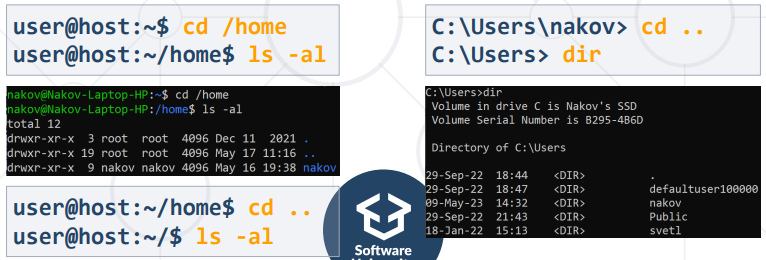
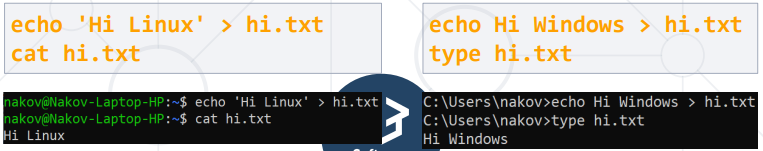
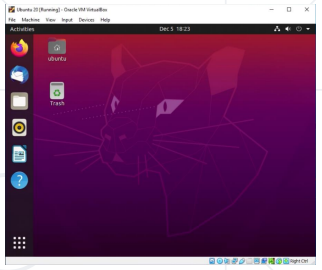
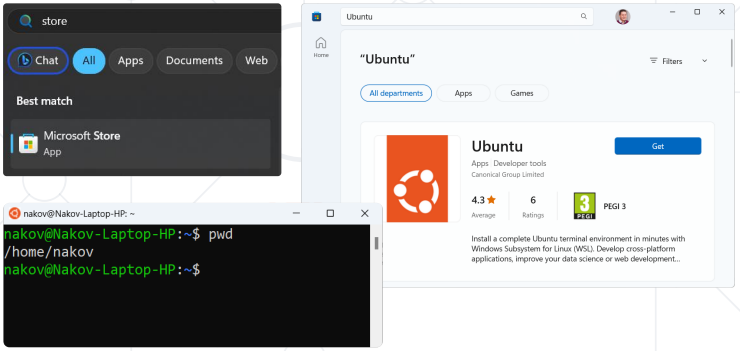
**Software Technologies Fundamentals**  
  
Computer Systems and Software

Hardware: Motherboard, CPU, RAM, Storage, Peripherals   
Software: Firmware, System, Server-Side, Applications  
  
▪ **Computer system**: an integrated bundle of hardware and software components, e. g. smartphone, POS terminal, laptop   
▪ Enables efficient data input, processing, and output   
▪ Comprises interconnected software and hardware components   
▪ Human-computer interaction for the end-users / APIs for machine-to-machine interaction   
▪ **Key elements:**   
 ▪ Hardware: RAM, input/output devices, storage devices, CPU   
 ▪ Software: operating systems, drivers, apps, games

▪ **Early computing**: mechanical and electromechanical devices (e.g., Abacus, Babbage's Analytical Engine, ENIAC)   
▪ **Advancements in technology**: transistors, integrated circuits, microprocessors (e.g., mainframe computers, minicomputers, personal computers)   
▪ **Modern era**: pervasive computing, IoT, cloud computing, edge computing, rise of AI and machine learning  
**Computer Hardware** - Motherboard, CPU, Memory, Storage, Peripherals  
▪ Hardware refers to the physical components of a computer   
▪ Central Processing Unit (CPU) – microprocessor   
▪ Executes the code (programs)   
▪ All data processing operations   
▪ Input devices ▪ Enter data   
▪ Output devices ▪ Get information  
 

  
  
▪ **Motherboard** == central hub for hardware connectivity   
 ▪ Communication between all hardware components   
▪ Compatibility considerations   
 ▪ Each motherboard is designed to work with specific types of processors and memory   
▪ Expansion slots for enhanced functionality   
 ▪ Video cards for improved graphics performance   
 ▪ Sound cards for enhanced audio capabilities   
 ▪ Network cards for better internet connectivity  
  
**Motherboard Components** ▪ CPU socket ▪ RAM slots ▪ Power connectors ▪ Chipset ▪ Expansion slots ▪ SATA connectors ▪ USB connectors ▪ Bluetooth module  


▪ **CPU – the brain of the computer**   
 ▪ Executes calculations, actions, and runs programs   
 ▪ Provides processing power and instruction control   
▪ Three core components   
 ▪ **Control Unit (CU)** ▪ Manages instruction flow and coordinates hardware functions   
 ▪ **Arithmetic and Logical Unit (ALU)** ▪ Performs arithmetic and logic operations   
 ▪ **Memory Unit (MU)** ▪ Stores data, programs, and information  
  
  
  
**Memory and Storage - Storing Information in a Computer**  
**▪ Primary memory**   
 ▪ RAM – read / write: stores data, required by the CPU during the execution of a program   
 ▪ ROM – read-only: stores crucial data for the system to operate, like the essential program for the computer boot   
**▪ Secondary memory**   
 ▪ Not accessed directly by the processor   
 ▪ Examples: hard drive, SSD, flash, optical drive, USB drive   
**▪ Cache memory**   
▪ Part of the CPU, very fast: temporarily stores frequently used instructions and data to speed-up access  
  
**Peripheral Devices** - Expanding Computer's Functionality  
▪ Any connected device that expands computer's capabilities with additional functionality   
▪ Three main categories:   
 ▪ **Input devices** → read data, e.g. keyboard, mouse, microphone   
 ▪ **Output devices** → write data, e. g. speakers, printer, monitor   
 ▪ **Input/output devices** → mixed, network card, hard drive, touchscreen monitor  
  
  
▪ **Device controller**   
 ▪ A physical device for connection between a peripheral device and the computer   
 ▪ E. g. USB controller   
**▪ Device driver**   
 ▪ System software, which enables the communication and data transfer between devices and the system  
  
  
**Computer Software** - Firmware, System Software, Applications  
▪ Computer software – definition   
 ▪ Computer programs, instructions, and data that enable a computer system to perform specific tasks   
▪ **Types of software**:   
 ▪ Application software: help the business to run, e.g. email software, spreadsheets, word processing, CRM systems, …   
 ▪ System software: interacts with and manages the hardware   
▪ Standalone apps vs. software systems (client + server)  
  
  
  
**Layers of Software**  
▪ **Firmware and embedded software**   
 ▪ Low-level software used to operate a hardware device   
▪ **System software**   
 ▪ Manages and controls hardware, platform for applications   
 ▪ Operating systems (OS) – Windows, Linux, macOS, Android   
 ▪ Hypervisors – runs virtual machines (VMs) in the host OS   
▪ **Application software**   
 ▪ Business applications, office apps, multimedia, communication   
 ▪ Several types: Web apps, desktop apps, mobile apps  
  
**Software Systems**  
▪ **Standalone apps**   
 ▪ Run locally, store their data locally, do not need Internet  
 ▪ Examples: Windows Calculator, Windows Explorer, Minesweeper   
▪ **Software systems**   
 ▪ Consists of several components (e. g. client + server)   
 ▪ Example: mail server (remote) + mail client app (local)   
▪ **Cloud apps:** hold all user data in the cloud + local client   
 ▪ Example: Google Docs, Discord, Trello, Canva  
  
▪ Front-end and back-end separate the modern apps into client-side (UI) and server-side (data) components   
▪ **Front-end** == client-side components (Desktop / mobile app / Web browser)   
 ▪ Implement the user interface (UI)   
▪ **Back-end** == server-side components (data and business logic APIs)   
 ▪ Implements data storage and processing Front-End and Back-End   
▪ **HTTP connects frontend with back-end**  
  
**Firmware** - Bridge between Hardware and Software  
▪ **Firmware** == permanent, low-level software, embedded in a device's read-only memory (ROM)   
 ▪ Controls device's basic functions and provides a stable foundation for higher-level software   
 ▪ Example: WiFi router's firmware, coffee machine firmware   
▪ **Functions of firmware**   
 ▪ Hardware initialization during the boot process   
 ▪ Management of low-level hardware operations (e. g. device initialization, hardware diagnostics, and system booting)  
▪ Examples of firmware applications   
 ▪ BIOS / UEFI in laptops and desktop computers   
 ▪ Firmware in routers, printers, scanners   
 ▪ Embedded systems, such as IoT devices   
▪ Firmware updates   
 ▪ Most devices allow firmware updates to improve functionality or fix issues   
 ▪ Can be critical for security and performance  
  
**System Software** - Foundation for Application Software  
▪ Software designed to manage and control computer hardware, providing a platform for application software   
▪ Examples of system software   
 ▪ **Hypervisors** – runs virtual machines (VMs) in the host OS   
 ▪ **Operating systems (OS)** – Windows, macOS, Linux, Android   
 ▪ **Device drivers** – software that enables communication between hardware and operating system), e. g. mouse driver   
 ▪ **System utilities** – tools for system maintenance and optimization, e. g. anti-virus, task manager, print spooler  
  
**Operating Systems**  
▪ Windows, macOS, Linux, Android, iOS   
▪ Manage the hardware and software resources   
▪ Manage processes (concurrently running apps)   
▪ Distribute the system resources between all processes   
▪ Manage file system and memory (RAM)   
▪ Manage users, security and access control   
▪ System updates and maintenance  
  
**Device Drivers**   
▪ In Windows, the "Device Manager" lists all devices, drivers, etc.  
  
  
**System Utilities**  
▪ Tools that help maintain and optimize a computer system   
 ▪ Antivirus and malware protection (e.g. Winows Defender)   
 ▪ System backup and recovery (e. g. Macrium Reflect)   
 ▪ Disk cleanup and defragmentation (e. g. CCleaner)   
 ▪ Performance monitoring and diagnostics (Task Manager)   
 ▪ Software updates and patches (e. g. Windows Update)   
 ▪ System hardware information (e. g. CPU-Z)   
 ▪ System logs viewer (e. g. Windows Events Viewer)  
  
**Server-Side Software (Backend) -** Facilitating Backend Operations and Web Services  
▪ Server-side software (backend software) runs on a remote server, processes requests and delivers data to client devices   
  
▪ Common types of **server*-*side software**   
 ▪ Web servers (e. g. Apache, Nginx, IIS)   
 ▪ Database servers (e. g. MySQL, PostgreSQL, MongoDB)   
 ▪ Application servers / runtimes (e. g. Tomcat, Node.js, .NET Core)   
 ▪ Mail servers (e. g. Microsoft Exchange Server, Postfix)   
 ▪ File servers (e. g. Windows File Server, Samba)   
 ▪ Authentication servers (e. g. FreeIPA, Active Directory)  
  
**▪ Server-side software (backend software):**  
 ▪ Executes on a remote server, rather than on the user's device   
 ▪ Handles data processing, storage, and retrieval   
 ▪ Powers Web applications, backend APIs, cloud services, etc.   
 ▪ Requires efficient resource management for optimal performance   
**▪ Graphical User Interface (GUI) / front-end apps:**   
 ▪ Executes on the user's device (desktop, mobile, or Web)   
 ▪ Providing seamless and visually appealing user experience   
 ▪ Can be Web apps, desktop apps, or mobile apps  
  
**Application Software** - Apps for the End Users  
▪ Application software is designed for users to perform specific business tasks, catered to their individual needs   
**▪ Examples of application software**   
 ▪ Productivity tools (Microsoft Office, Google Workspace)   
 ▪ Multimedia software (Adobe Photoshop, VLC Media Player)   
 ▪ Communication apps (Zoom, WhatsApp, MS Teams)   
 ▪ Web browsers (Google Chrome, Mozilla Firefox, Safari)   
 ▪ Games (Fortnite, League of Legends)  
  
**Web Apps** - Applications, Accessed from the Web Browser  
▪ What are Web apps?   
 ▪ Accessed through a Web browser with an active Internet connection   
 ▪ **Platform-independent**   
 ▪ Accessible on any device with a Web browser   
 ▪ Desktop/mobile Web browsers   
 ▪ **Automatic updates (always up-to-date)**   
 ▪ No need for manual installation or updating

▪ **Benefits of Web apps**   
 ▪ Scalability: easily accommodate a growing user base   
 ▪ Centralized data storage: simplifies data management and backup   
 ▪ Lower device requirements: minimal hardware needed (processing is done on the server-side)   
 ▪ Easier collaboration: real-time collaboration   
 ▪ Cross-platform compatibility: works across various operating systems and devices  
  
▪ **Compatibility:** if the app works consistently across different Web browsers and different screen sizes (responsive design)   
▪ **Usability:** testing for accessibility, intuitive use on different devices, and ease of navigation   
▪ **Network conditions:** Web apps rely on an active internet connection→ testing under different network conditions   
▪ **Security:** Web apps deal with sensitive data → testing for vulnerabilities such as XSS attacks and SQL injection   
▪ **Performance:** performance can be affected by network speed / server load / browser capabilities → testing for scalability / load capacity  
  
  
  
**Desktop Apps** - Applications Running Locally on Your Laptop  
▪ What are desktop apps?   
 ▪ **Installed and run locally on a user's computer**   
 ▪ Store their data locally or remotely (depends)   
 ▪ **Offline access**   
 ▪ Can be used without an Internet connection   
 ▪ **More features**   
 ▪ Often more feature-rich than Web apps   
 ▪ Better integrated with the host OS  
  
  
▪ Benefits of desktop apps   
 ▪ **Performance:** faster processing and response time, as tasks are executed locally   
 ▪ **Customization:** easily tailored to individual user preferences and needs   
 ▪ **Integration:** compatible with other locally installed software and hardware   
 ▪ **Cost-effective:** one-time purchase or licensing fees, instead of recurring subscription costs (depends)  
  
▪ Installation / uninstallation including any dependencies or prerequisites   
▪ Performance testing on different hardware configurations – processors, memory, and graphic cards ▪ Compatibility testing for different operating systems and their different versions   
▪ User interface testing: desktop apps often have complex UI that need to be thoroughly tested   
▪ Integration testing with other desktop applications  
  
  
  
**Mobile Apps** - Applications Running Locally on Mobile Device  
▪ What are mobile apps?   
 ▪ Designed specifically for smartphones and tablets   
 ▪ Accessible through dedicated app stores (e.g., Google Play, Apple App Store)   
 ▪ Optimized for touchscreen interfaces and mobile device features (adaptable UI design for different screen sizes)   
 ▪ Can work offline, online or mixed  
**▪ Benefits of mobile apps**   
 ▪ **Portability:** access apps and data on-the-go, anytime, anywhere   
 ▪ **Push notifications:** real-time updates and alerts for improved user engagement   
 ▪ **Device-specific features:** leverage device capabilities like GPS, camera, and sensors   
 ▪ **Offline functionality**: some apps can operate without an Internet connection   
 ▪ **Streamlined user experience:** tailored for smaller screens and touch-based interactions  
  
  
  
**Testing Challenges for Mobile Apps**  
▪ **Compatibility** across different devices and OS versions is crucial for mobile apps (many different devices and versions in use)   
▪ **User interface testing** – design and layout has significant impact on the user's experience on a smaller screen   
▪ **Performance testing** – performance may be affected by limited processing power and memory on the user's device   
▪ **Battery life testing** – to ensure that the app does not significantly drain the user's device battery  
  
  
  
**Summary**  
▪ **Hardware** is the physical part, whereas software is a set of instructions for the computer   
 ▪ Main computer parts are the motherboard (ties together all components), CPU (code execution), input / output devices   
▪ **Software** – programs, running in the computer   
 ▪ Firmware and system software (OS, hypervisors)   
 ▪ Server-side software (back-end) vs. GUI / front-end apps   
 ▪ Application software (end-user apps): Web apps, desktop apps, mobile apps   
 ▪ Software systems (client + server) and cloud apps  
  
  
  
  
  
  
  
  
  
  
  
Operating Systems  
OS Overview, Linux Shell, VM and Containers  
  
**Operating Systems (OS) – Overview** - OS Purpose and Structure  
▪ The **operating system (OS)** manages applications (processes), users, file system and resources in a device   
▪ The OS is loaded into a device through a process called booting   
▪ OS enables applications to interact with the device's hardware and software resources   
▪ Applications make requests for services through a defined interface called an application program interface (API)   
▪ At least one OS must be installed in a device to run basic programs, e. g. Web browser, file explorer, video player  
  
**OS Main Functions**  
▪ **Booting** – turning on the device and loading the OS   
▪ **App loading and execution** – load and run programs (processes, apps), start / view / pause / terminate apps   
▪ **Process management** – allocates resources to OS processes, share data between processes, protects, and synchronizes them   
▪ **Memory management** – controls and coordinates the memory allocation for the applications running in the OS   
▪ **Disk management** – manages storage (hard drives, SSD disks, optical disk drives, flash drives) and file systems  
▪ **Device controlling** – controls the access to physical devices (like disk drives, CD/DVDs, USBs) and virtual devices (like random)   
▪ **Networking** – communication over the network and Internet   
▪ **Printing controlling** – takes control of printers connected and manages the printing process   
▪ **User interface (UI)** – provides UI for the users to interact with the computer by commands or visual UI elements   
▪ **Data security** – isolate apps, users and files to keep data secure (e. g. using file system / resource permissions)  
  
▪ **Kernel** == core component of the OS   
▪ **The OS "heart"** – bridges hardware and software components   
▪ Facilitates communication between different system components   
▪ Provides complete control over the system   
▪ Always stays resident in memory   
▪ Essential for running any operating system  
  
  
  
▪ **Drivers** == set of system programs that enable hardware components to function   
▪ Drivers connect the OS and devices   
 ▪ Enable hardware components or peripherals to operate properly   
▪ Drivers are low-level software programs without a user interface (UI)   
▪ All hardware components require a driver (e. g. disk drives, printers, keyboards)  
  
  
  
▪ **Shell** == user interface (UI) to the OS   
 ▪ Outermost layer of the operating system, located between the kernel and the apps   
 ▪ Provides a UI and tools to control processes, files, installed software, users, etc.   
▪ Two types of shells:   
 ▪ **Command-line (CLI) shells** – require knowledge of commands, syntax, and concepts about the shell-specific scripting language (e. g. bash)   
 ▪ **Graphical (GUI) shells** – intuitive, easy to use (e. g. Windows Desktop)   
▪ Most GUI-enabled OS provide also CLI shells for advanced users  
**Users in Operating Systems**  
▪ **Users in the OS** == individuals or entities who interact with the system by logging in and performing tasks   
▪ A user often has a user account and is identified to the system by a username   
▪ Users may have privileges over processes, folders and files, devices, services, network and other resources   
 ▪ Users are typically isolated from each other   
▪ OS can be single-user (e. g. DOS) or multi-user (e. g. Linux, macOS, Windows)  
  
▪ **User accounts** allow access to a system's resources   
▪ Authentication is the process of verifying a user's identity   
 ▪ Through credentials (like passwords / keys)   
▪ Authorization determines what resources a user can access based on their authenticated identity   
▪ User accounts in the OS are important for accounting, security, logging, and resource management  
  
  
**Authentication vs. Authorization**  
▪ **Authentication** verifies the identity of a user or service   
▪ **Authentication** answers the question:   
 ▪ Who are you?  
▪ **Authorization** determines the user's access rights   
▪ **Authorization** answers the question:   
 ▪ What are you allowed to do?  
  
**User Permissions**  
▪ OS controls the use of system and network resources   
 ▪ Through authentication and authorization   
 ▪ Based on user permissions over resources (e. g. file permissions)   
▪ The OS determines if an authenticated user has the correct permissions to access a resource   
 ▪ Using built-in authorization and access control technologies  
  
**User Roles (Groups)**  
▪ User roles (groups) are permission sets that control access to resources (files, folders, processes, services)   
 ▪ Simplify permission assignments, e. g. in a hosting company, all customers may use the group "web"   
▪ Each user account may have multiple roles   
▪ Examples of user roles **in MS Windows**: Administrator, User, Power User, Guest   
▪ Examples of user groups **in** **Linux**: root, user, nobody  
  
**Access Permissions in OS**  
▪ **Access permissions** determine a user's ability to perform a specific action, or access a feature or object   
▪ Set access permissions to specify which users, groups, or roles can access your content   
▪ The most common permissions are read, write and execute  
  
  
**Processes in OS**  
▪ A process is a program in action (a running app)   
 ▪ Consume CPU time, RAM memory, file handles and other OS resources   
▪ It's the basic unit of work in the operating system   
▪ Unlike files, which are passive, processes are an active entity   
▪ For example, when you open a browser to search the web, that's a process  
  
**Task Managers**  
▪ In OS, a task manager is a system monitoring app   
 ▪ View processes, users, consumed resources, etc.   
 ▪ View RAM, CPU, GPU, disk, network load   
 ▪ Start / terminate (kill) processes   
▪ Examples:   
▪ Windows Task Manager in MS Windows ▪ top and htop in Linux ▪ Activity Monitor in macOS  
  
**Windows Task Manager**▪ Open the Task Manager in MS Windows:   
 ▪ [Ctrl + Alt + Delete] → select [Task Manager] from the menu   
 ▪ Right click on the task bar → [Task Manager]  
  
  
  
**Popular Operating Systems -** Windows, Linux, macOS, Android, iOS  
  
**Most Popular Operating Systems**  
▪ Five major operating system:  
▪ Microsoft Windows ▪ Apple macOS ▪ Google's Android OS ▪ Apple iOS ▪ Linux (open source)  
  
**Microsoft Windows**▪ Proprietary OS, developed by Microsoft   
▪ One of the most popular OS   
 ▪ Typically preinstalled on new PC   
▪ Several versions: Windows 95 / 98 / Vista, Windows 7 / 8 / 10 / 11   
 ▪ Has been around since the 1980s   
▪ Easy-to-use, intuitive GUI shell   
 ▪ Many apps and games  
  
**Apple macOS**  
▪ Apple and Macintosh computers run on macOS and OS X   
 ▪ Proprietary OS developed by Apple   
▪ macOS is a Unix-based OS   
 ▪ Released over 20 years ago   
▪ In 2020, Apple began transitioning to its own 64-bit ARM-based Apple M CPU   
 ▪ Apple M1 / M2 CPU: powerful and silent  
  
  
  
**Android OS**  
▪ Mobile OS, designed for touchscreen mobile devices   
▪ Based on a modified version of the Linux kernel and other open-source software   
▪ Core OS is called Android Open-Source Project (AOSP)   
 ▪ Free and open-source software   
 ▪ Developed and maintained by Google   
▪ Many distributions (by Samsung, Xiaomi)  
  
**Apple iOS**  
▪ Mobile OS, developed by Apple   
 ▪ Exclusively for its hardware devices: iPhone, iPad and iPod Touch   
▪ Closed ecosystem, dominated by Apple   
▪ iOS UI uses multi-touch gestures: swipe, tap, pinch, and reverse pinch   
▪ iOS runs on Apple hardware only   
 ▪ Might run on PC emulators, but is illegal  
  
**Linux**  
▪ Linux is Free and open-source family of operating systems   
▪ Linux's popularity comes from its ease of customization and open license   
▪ Offers CLI shell and many GUI desktops   
▪ Many distributions: Ubuntu, CentOS, Debian, Mint, openSUSE, Alpine, …   
▪ It offers a variety of options for those who understand how to use it  
  
**Virtual Machines & Containers -** Remote Instances & Emulators  
**Virtual Machines (VM)**  
▪ A virtual machine (VM) is a software-based computer resource, used to run an OS inside another OS ▪ Digital version of a physical computer that can run programs and OS, store data, connect to networks, and other computing functions   
▪ **Virtualization** == running a virtual machine (VM) / virtual environment inside a physical hardware system   
 ▪ E. g. run Android VM or Linux inside a Windows host   
 ▪ Storage, networking, desktops can also be virtual  
  
**Containers and Docker**  
▪ **Container image** == software, packaged with its dependencies, designed to run in a virtual environment (like Docker)   
 ▪ E. g. WordPress instance (Linux + PHP + Apache + WordPress)   
 ▪ Simplified installation, configuration and deployment   
 ▪ **Lightweight** – containers use shared OS kernel with the host   
▪ Docker is the most popular containerization platform   
 ▪ Runs containers from local image or downloaded from the Docker Hub online repository   
 ▪ Open-source, runs on Linux, Windows, Mac  
  
  
  
**Docker Containers**  
▪ A Docker container image is a lightweight, standalone executable package of software   
 ▪ Contains everything needed to run an app: code, runtime, libraries, tools, and settings   
▪ **Container** == running Docker image   
 ▪ App, running inside the Docker Engine   
▪ Containers provide fast and simple way to run apps, without installing them on the host OS   
▪ Containers are isolated from the host and other containers → security  
  
**Remote VM Instances and Docker Playground**▪ Containers allow for customizable and replicable instances of an application   
▪ Without interfering with anything else on a user's system (no conflicts)   
▪ Docker Playground is an interactive and fun way to learn Docker   
▪ Provides free Linux + Docker VMs   
▪ Accessible for 4 hours, for learning   
▪ <https://labs.play-with-docker.com>  
  
  
  
**Device Emulators**  
▪ **Device emulators** run Android / iOS / other OS in a virtual machines (VM) and simulate device functions (e. g. rotation)   
▪ BlueStacks, LDPlayer, Android Emulator - run Android apps in Windows and simulate mobile devices  
  
  
**BrowserStack – App & Browser Testing**  
▪ **BrowserStack** – manual and automated online mobile testing for Web sites and mobile apps   
▪ Test on remote physical devices: iPhone, iPad, Samsung, Xiaomi, Google smartphones / tablets   
▪ Modern devices, modern Web browsers   
▪ Android, iOS, Windows, macOS   
▪ **BrowserStack Live** offers 3000+ devicebrowser-OS combinations for testing  
  
  
**Shell & Shell Commands -** Shell Command Execution on Linux and Windows  
Opening the CLI Shell in MS Windows  
 1. Click [Start] -> [Run] or press [Windows + R] key   
 2. Type "cmd"   
 3. Click on [Command Prompt]  
  
  
**Linux Shell in Docker Playground**  
▪ Starting a Docker Playground session   
 ▪ Open Docker Playground, register and log in   
 ▪ Press [Start] and add a new instance   
 ▪ Now you have a Linux VM + Docker environment to experiment with  
  
**Commands: ls & dir**  
▪ ls list files and directories in Linux / UNIX / macOS  
▪ dir lists the files and folders in Windows  
  
  
**Commands: cd**  
▪ cd changes the current working directory in Linux  
▪ cd works the same way in Windows  
  
  
**Commands: pwd / cd**  
▪ pwd prints the current working directory in Linux  
▪ cd works the same way in Windows  
  
  
**Commands: echo and cat / echo and type**  
▪ echo '…' > filename prints a text to a file in Linux   
▪ cat displays the content of given file  
▪ echo … > filename prints a text to a file in Windows   
▪ type displays the content of given file  
  
  
**Can I Run Linux Commands on Windows PC?**  
▪ You can run Linux in Windows through a virtual machine   
 ▪ E. g. Ubuntu Linux in Virtual Box   
▪ You can run Linux in Windows Subsystem for Linux (WSL)  
  
  
  
**Install WSL and Ubuntu Linux in Windows 11**  
 **Summary**  
▪ Operating Systems (OS) manage processes, users, files and other resources   
▪ OS Examples: Windows, macOS, Linux, Android, iOS   
▪ Virtual machine (VM) == OS inside another OS   
▪ Container == app image, running in Docker   
▪ Shell commands == execute commands from the console (Linux / Windows shell)

**Network Fundamentals**

**OSI Model, MAC Address, IP Address, TCP and Ports**

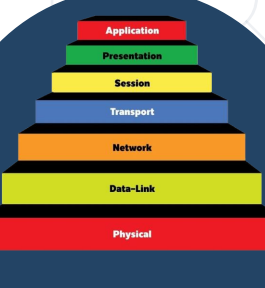
**Network Protocol** - a set of rules that determine how data is transmitted between different devices on the same network  
▪ enable standardized communication between devices / programs

▪ Typically, one party sends a request (command / question / other) and receives a response from the other party  
▪ Network protocols govern aspects of data transmission, addressing, routing, flow-control, and error handling

**Network Layering Models**

* Layers organize networking into a structured framework
* Facilitate the understanding, design, and management of complex networks
* Simplifies network communication and troubleshooting

**Examples : OSI model (7 layers) and TCP (4 layers)**

**OSI Model**  


▪ **OSI Model** – consists of 7 layers each layer stacks on the previous and adds functionality to the data transmitted

1. **Physical Layer** - cables and radio.

* Converts digital data into electrical impulses, radio signals, or optical signals for transmission. Devices : hubs, repeaters, antennas

1. **Data Link Layer** – MAC address, frames.

* Manages data transmission, error detection / correction
* Devices: switches, bridges, network interface cards (NICs)

1. **Network Laye**r - hosts and IP address, packets.

* Packet routing: host → router → router → … → end host,
* Devices : routers, layer 3 switches

1. **Transport Layer** – ports.

* Error checking, flow control, congestion control, multiplexing

1. **Session Layer**

* dialog control, token management, synchronization

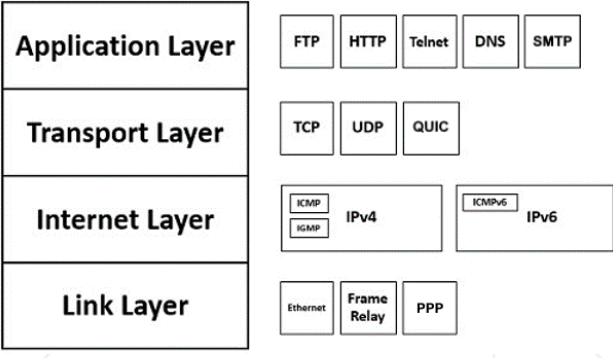
1. **Presentation Layer** - data formats

* data representation, encryption, decryption, compression, decompression

1. **Application Layer** – applications

* Networking for applications, e. g. Web browsers use DNS, HTTP and HTTPS to open a Web site
* Protocols - **HTTP, HTTPs, FTP, SMTP, IMAP, DNS**

**TCP Model**



**TCP/IP Layers**

1. **Link layer** – Combines the functionalities of OSI Physical and Data Link layers
2. **Internet Layer** - Corresponds to the OSI Network Layer
3. **Transport Layer** - Closely resembles the OSI Transport Layer
4. **Application Layer** - Merges the functionalities of OSI Session, Presentation, and Application layers

**MAC, IP, Netmask, Gateway**

**Media Access Control (MAC) Address**

* MAC address is a unique hardware identifier assigned to network interface cards (NICs)
* Format: 48-bit (6 hex numbers), e. g. 9c-93-4e-3f-14-f7

**Internet Protocol (IP) Address**

* IP address == 32-bit identifier (e. g. 192.168.0.61) assigned to devices in a network for addressing and routing purposes
* Netmask (e. g. 255.255.255.0) is a 32-bit number, used to masks out the network part of an IP address
* Gateway (e. g. 192.168.0.1) is the router IP used to access Internet
* IPv6 address == 128-bit address for the modern Internet

**Ports**

**1. Ports Overview -** Numerical identifiers used to distinguish specific processes or services running on a device within a network

* Facilitate end-to-end communication between applications on different devices

**2. Types of Ports**

- **TCP ports** - Used for connection-oriented communication, ensuring reliability and data integrity

**- UDP port**s - Used for connectionless communication, providing faster data transmission with minimal overhead

**3. Port Numbers** - Used to identify a network service

**Networking: Summary**

* Communication in Internet uses networking protocols
* IP: host-to-host communication in local networks and Internet
* TCP: implements reliable transport of data streams; uses ports to distinguish connections
* UDP: transports single packets, connectionless, faster, has no error checking; uses ports to distinguish connections
* DNS: maps hosts to IP addresses (e. g. softuni.org → 172.67.168.4)
* HTTP: request-response text-based protocol for the Web

**Web Fundamentals**

1. **Domain Name System (DNS)**

* A hierarchical, distributed system (part of Internet) that translates domain names into IP addresses
* Facilitates the resolution of human-readable domain names to machine-readable IP addresses

1. **Domain name**

* a unique, human-readable name for Internet host / machine / web site
* Simplify navigation to websites, easier to remember and share

1. **Uniform Resource Locator (URL)**

* a unique address pointing to a website, a web page, or a document on the Internet

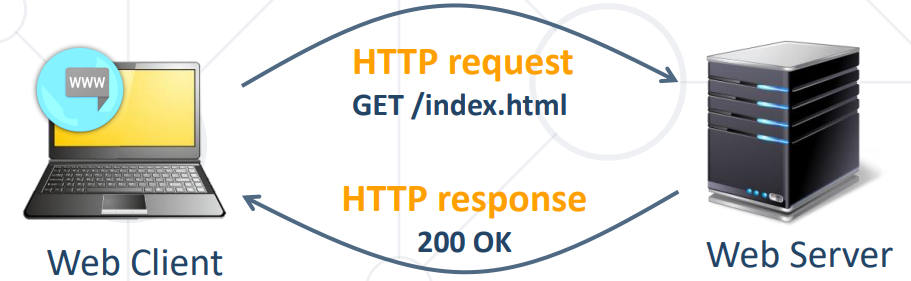
1. **WWW (World Wide Web)**

* A global, interconnected system of documents, images, and other resources, accessed through the Internet using Web browsers

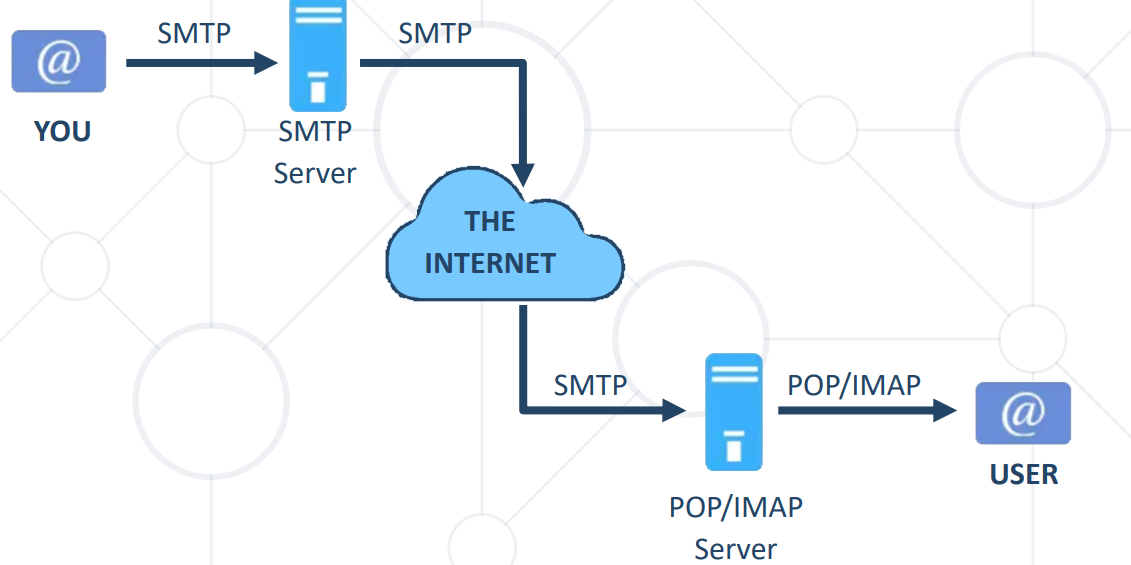
**HTTP Protocol**

**HyperText Transfer Protocol (HTTP)**

* Text-based client-server protocol for the Internet
* For transferring Web resources (HTML files, images, styles, etc.)
* Request-response based



**Email Protocols: SMTP and IMAP**



**SMTP Protocol (Simple Mail Transfer Protocol)**

* Send / receive email messages between mail servers

**IMAP (Internet Message Access Protocol)**

* Retrieve email messages from server mailbox
* Allows management of email messages on the server from different devices (sync and delete)
* More popular and flexible

**POP (Post Office Protocol)**

* Once downloaded to a client, the message is removed from the server (download and delete)
* Difficult to access email messages from different devices or locations