Seminar content

What is Web development

Imagine you want to build a house. You need to design the layout, construct the walls, install the plumbing and electricity, and finally, decorate the interior. Building a website is similar, but instead of physical materials, you work with digital tools and code.

Web Development is the process of creating a website or web application that can be accessed over the internet. It involves several stages, just like building a house:

1. **Planning and Design:** Decide what the website should look like, what features it should have, and how it should work. This is like creating a blueprint for your house.
2. **Front-end Development:** Build the user interface (UI) and user experience (UX) of the website using programming languages like HTML, CSS, and JavaScript. This is like constructing the walls and installing the doors and windows of your house.
3. **Back-end Development:** Create the server-side logic, database integration, and API connectivity using programming languages like Python, Ruby, or PHP. This is like installing the plumbing and electricity in your house.
4. **Deployment:** Upload the website to a server or hosting platform, making it accessible to the public. This is like moving into your newly built house.
5. **Maintenance and Updates:** Regularly update the website with fresh content, fix bugs, and ensure it remains secure and fast. This is like maintaining your house, fixing leaky faucets, and repainting the walls.

Web Development Roles:

1. **Front-end Developer:** Focuses on the client-side, building the UI and UX using HTML, CSS, and JavaScript.

2. **Back-end Developer:** Focuses on the server-side, building the logic, database integration, and API connectivity using programming languages like Python, Ruby, or PHP.
3. **Full-stack Developer:** Works on both front-end and back-end development, handling the entire website development process.
4. **Web Designer:** Focuses on the visual design and user experience, creating wireframes, prototypes, and high-fidelity designs.

Web Development Technologies:

1. **HTML (Hypertext Markup Language):** Used for structuring content on the web.
2. **CSS (Cascading Style Sheets):** Used for styling and layout.
3. **JavaScript:** Used for adding interactivity and dynamic effects.
4. **Frameworks and Libraries:** Like React, Angular, or Vue.js, which simplify development and improve efficiency.
5. **Databases:** Like MySQL, MongoDB, or PostgreSQL, which store and manage data.
6. **Server-side Languages:** Like Python, Ruby, or PHP, which handle server-side logic and database integration.

FrontEnd

Why?
Market trend - 2015-2025 graph

What is web development

Front-end web development is the process of designing and building a website or application's user interface and experience. This includes:

- Layout: Designing and coding the layout
- Interactive features: Creating interactive features
- Responsiveness: Ensuring the site works well on various devices and browsers.

Types of Employers and Industries

1. **Tech Companies**
   - **Examples**: Google, Facebook (Meta), Microsoft, Apple.
   - **Focus**: Developing cutting-edge web applications, platforms, and user interfaces.
2. **E-Commerce**
   - **Examples**: Amazon, Shopify, eBay.
   - **Focus**: Building user-friendly and high-performance online stores and shopping experiences.
3. **Finance and Banking**
   - **Examples**: JPMorgan Chase, Goldman Sachs, PayPal.
   - **Focus**: Developing secure and efficient web applications for financial transactions and services.
4. **Media and Entertainment**
   - **Examples**: Netflix, Spotify, Disney.
   - **Focus**: Creating engaging and interactive media platforms and streaming services.
5. **Healthcare**
   - **Examples**: Cerner, Epic Systems, Johnson & Johnson.
   - **Focus**: Building interfaces for healthcare applications, patient portals, and medical data systems.

6. **Startups**
   - **Examples**: Various tech startups across different sectors.
   - **Focus**: Often involve working on innovative projects with a fast-paced environment and varied responsibilities.

What a web developer do?

Web developers create functional, user-friendly websites and web applications. They may write code, develop and test new applications, or monitor site performance and traffic. Front-end developers focus on the user-facing side of their work and the front-end portion of websites and web applications the part users see and interact with.

Blocks in frontend

1. HTML
2. CSS
3. JAVASCRIPT

Explain each blocks

HTML(House Items)

In this haunted house, the HTML represents the physical items that make up the house. Each item is a tangible object that has a specific purpose, just like how HTML elements have specific meanings and uses.

Furniture (Divs, Spans, etc.): The couch, chairs, tables, and other furniture in the house are like the div, span, and other HTML elements that provide structure and organization to the content.

Decorations (Images, Icons, etc.): The paintings, vases, and other decorative items in the house are like the img, svg, and other HTML elements that add visual appeal and interest to the content.
Doors and Windows (Links, Forms, etc.): The doors and windows of the house are like the a, form, and other HTML elements that provide entry and exit points for the user to interact with the content.

Css(Household Items' Color and Smoke Effects)

In this haunted house, the CSS represents the color and smoke effects that bring the house to life. Just as CSS styles and layouts can enhance the appearance of HTML elements, the color and smoke effects in the house add an eerie and mysterious atmosphere.

Color Schemes (Background Colors, Text Colors, etc.): The color schemes used in the house are like the background colors, text colors, and other CSS properties that define the visual aesthetic of the HTML elements.
Smoke and Fog Effects (Transitions, Animations, etc.): The smoke and fog effects in the house are like the transitions, animations, and other CSS effects that create a sense of movement and drama.

JS (The Ghost)

In this haunted house, the JS represents the ghost that brings the house to life. Just as JS code can manipulate and interact with HTML elements, the ghost can control the household items and effects to create a spooky and dynamic experience.

Ghost's Helpers (Fetch, SetTimeout, etc.):
Fetch: The ghost's ability to fetch new items or information from the outside world is like the fetch API, which allows JS to retrieve data from external sources.

SetTimeout: The ghost's ability to set traps or schedule events is like the setTimeout function, which allows JS to execute code after a specified delay.
LocalStorage: The ghost's ability to store secrets and memories is like the localStorage API, which allows JS to store data locally in the user's browser.
The ghost uses its helpers to control the household items and effects, creating a dynamic and interactive experience for anyone who dares to enter the haunted house!

Common types on websites

1.Search Engines

A search engine is a software program that helps people find the information they are looking for online using keywords or phrases. Search engines are able to return results quickly—even with millions of websites online—by scanning the Internet continuously and indexing every page they find**.**

Usecases:

· **Google** (google.com): The most widely used search engine for finding information on the internet.

· **Bing** (bing.com): Microsoft's search engine, offering an alternative to Google with unique features like rewards for searches.

· **Yahoo!** (yahoo.com): A search engine and web portal offering news, email, and other services.

2. Social Media

Online platforms where users create, share, and interact with content, information, or other users. It Connect with others Share experiences and ideas, Stay informed about news and events, Entertain and be entertained,Build personal or professional brands

Usecases:

· **Facebook** (facebook.com): The largest social networking site, used for connecting with friends, sharing updates, and joining communities.

· **Instagram** (instagram.com): A photo and video sharing platform owned by Facebook, popular for visual content and influencers.

· **Twitter** (twitter.com): A microblogging platform where users share short updates and engage in discussions.

· **LinkedIn** (linkedin.com): A professional networking site used for career networking, job searching, and professional development.

3. E-Commerce

Buying and selling goods or services online,shop from anywhere,access to global products and services, compare prices and find deals, secure online transactions

Usecases:

- **Amazon** (amazon.com): The largest online retailer, offering a vast selection of products, including electronics, books, clothing, and more.
- **eBay** (ebay.com): An online auction and shopping website where individuals and businesses buy and sell a wide variety of goods.
- **Alibaba** (alibaba.com): A global e-commerce platform connecting buyers with manufacturers and suppliers, particularly for wholesale goods.

4. News and Media

The dissemination of information and entertainment content to the public through various channels Stay informed about current events and issues, Be entertained and educated

Usecases:

- **BBC** (bbc.com): A leading global news outlet, providing comprehensive news coverage, analysis, and documentaries.
- **CNN** (cnn.com): A major news network offering real-time coverage of global events, politics, business, and more.
- **The New York Times** (nytimes.com): An influential newspaper known for its in-depth journalism and coverage of a wide range of topics.

Commonly used components and with its use cases

Navigation Components

- **Header/Nav Bar**: A top section of a webpage containing links to other parts of the website, such as Home, About, Contact, etc.

  Use case: Primary navigation for a website or application, often featuring a logo, menu items, and search functionality.

- **Sidebar**: A vertical menu, often on the left or right side of the page, used for navigation or additional content.

  Use case: Providing quick access to related content or features, often used in dashboards, admin panels, or documentation websites.

- **Footer**: The bottom section of a webpage, usually containing links to legal information, contact details, and social media.

  Use case: Secondary navigation providing additional links and information, often including copyright details, social media links, and contact information.

2. Input Components

- **Text Fields/Input Boxes**: Fields where users can input text, such as in forms or search bars.
- Use case: Collecting short, single-line text input from users, such as names, emails, or search queries.
- **Buttons**: Clickable elements that trigger an action, like submitting a form or opening a menu.
- Use case: Allowing users to select one option from a list, such as gender, language, or payment method.
- **Checkboxes/Radio Buttons**: Used in forms for selecting options; checkboxes allow multiple selections, while radio buttons are for single selections.
- Use case: Allowing users to select one or more options from a list, such as terms and conditions, newsletter subscriptions, or feature toggles.
- **Dropdown Menus**: A list of options that appears when the user clicks on a button or input field.

- Use case: Providing a compact way to select one option from a list, such as country, language, or category.
- **Sliders**: Allow users to select a value from a range by moving a handle along a track.
- Use case: Allowing users to select a value from a range, such as pricing, ratings, or quantity.

3. Display Components

- **Cards**: Container elements used to display content in a grouped manner, often with images, text, and buttons. Commonly used for product listings, blog previews, etc.
- Use case: Displaying a concise summary of information, such as a product, article, or user profile.
- **Tables**: Grid layouts used for displaying data in rows and columns, often with sortable headers.
- Use case: Displaying structured data, such as lists, datasets, or reports.
- **Modals/Dialogs**: Overlay components that appear on top of the main content, used for alerts, forms, or additional information.
- Use case: Displaying temporary, contextual information, such as alerts, warnings, or additional details.
- **Tooltips**: Small pop-up boxes that provide additional information when hovering over an element.
- Use case: Providing additional information or context about an element, such as a button or icon.

4. Media Components

- **Images**: Used to display pictures, illustrations, or icons. Can be static or dynamically loaded.
- Use case: Displaying visual content, such as product images, logos, or icons.
- **Videos**: Embedded media players to stream or play video content
- Use case: Displaying video content, such as tutorials, product demos, or promotional videos.

5. Forms and User Input

- **Forms**: Collections of input fields, buttons, and other controls that users fill out and submit, typically used for registrations, logins, and data collection.
- Use case: Validating user input to ensure it meets certain criteria, such as email format or password strength.
- **Validation Messages**: Error or success messages displayed in response to user input in forms, ensuring the input meets specific criteria.
- **Date Pickers**: Input components that allow users to select dates from a calendar interface.
- Use case: Collecting date or time input from users, such as selecting a birthdate or appointment time.

6. Interactive Components

- **Tabs**: A component that allows users to switch between different views or sections of content within the same space.
- Use case: Organizing content into separate sections or views, such as displaying different types of content or settings.
- **Pagination**: Navigation controls used to divide content across multiple pages, often found in search results or product listings.
- Use case: Navigating through a large dataset or list, such as navigating through search results or a list of products.

7. Layout Components

- **Grid Systems**: A flexible structure that divides the page into rows and columns, helping to align and organize content.
- Use case: Organizing content into a structured grid, such as displaying a list of products or images.
- **Containers**: Wrapper components that hold other components and provide structure to a webpage, often with padding or margin settings.
- Use case: Grouping elements together, such as wrapping a header or footer section.

8. User Authentication Components

- **Login/Signup Forms**: Forms used for user authentication, often including fields for username/email and password.
- Use case: Allowing users to log in to their account, such as accessing a dashboard or protected content.
- **Password Reset Forms**: Components that allow users to reset their password by providing their email or username.
- Use case: Allowing users to reset their password, such as when they forget their password.

Popular frameworks and libraries

1. React

- **Companies**: Facebook (Meta), Instagram, Airbnb, Netflix, WhatsApp, Atlassian
- **Why It's Used**: React's component-based architecture and virtual DOM provide high performance and flexibility, making it ideal for

building complex and dynamic user interfaces. It also has a large ecosystem and strong community support.

2. Angular

- **Companies**: Google, Microsoft, IBM, Intel, Upwork, Santander
- **Why It's Used**: Angular offers a full-featured framework with built-in solutions for routing, state management, and form handling. Its strong TypeScript integration and robust tooling make it a good fit for enterprise-level applications.

3. Vue.js

- **Companies**: Alibaba, Xiaomi, Laravel, GitLab, Adobe, Behance
- **Why It's Used**: Vue.js is known for its ease of integration and flexibility. It's lightweight and easy to learn, making it suitable for both small projects and large-scale applications. Its reactive data-binding and component-based architecture are highly valued.

4. Svelte

- **Companies**: Square, Bloomberg, The New York Times, Rakuten

Why It's Used: Svelte compiles components into highly efficient imperative code, offering excellent performance and a simplified development experience. It's gaining traction for its innovative approach to front-end development.

Most famous site build by popular frameworks and libraries

Vue.js-A popular framework that uses component-based architecture and visual DOM to simplify the work of developers.

React-An open-source framework that allows developers to create prototypes on screen and describe user interfaces. It's also used to build online apps, including single-page applications and multi-page services.

Ember.js-A JavaScript framework that uses the MVVM (Model-View-ViewModel) architecture pattern for rapid development. It also has a command line interface tool that can help with productivity.

Svelte-A modern and scalable framework that uses JavaScript libraries to render a simple framework. It transcribes code to update the DOM in sync with the application.

Backbone.js-A JavaScript framework that can be used to develop single-page applications. It allows server-side functions to flow through an API.

About WordPress

WordPress is an open-source content management system (CMS) used for building and managing websites. It began as a blogging platform but has since evolved into a versatile CMS capable of powering a wide range of websites, from simple blogs to complex e-commerce sites and enterprise-level applications.

1. Free and Open-Source: WordPress is free to use, modify, and distribute.

2. Highly Customizable: Thousands of themes, plugins, and widgets to personalize your site.

3. User-Friendly: Easy to use, even for beginners, with a intuitive dashboard and drag-and-drop functionality.

4. Extensive Plugin Library: Over 50,000 plugins to extend functionality, including SEO, security, and e-commerce tools.

5. Responsive Design: WordPress sites are mobile-friendly and responsive, ensuring a great user experience on any device.

6. Search Engine Optimization (SEO): WordPress has built-in SEO features and plugins to improve search engine rankings.

7. Multilingual Support: WordPress supports over 160 languages, making it a great choice for global audiences.

8. Security: WordPress has a strong focus on security, with regular updates and a large community of developers working to prevent vulnerabilities.

9. Scalability: WordPress can handle large amounts of traffic and is used by many high-traffic websites.

10. Community Support: WordPress has a massive community of developers, designers, and users, ensuring there's always help available.

Roles and responsibilities of web developer

The web developer is responsible for planning and developing software solutions and web applications, supporting and maintaining a company's websites and digital products. The day-to-day work of the web developer highly depends on constantly evolving internet innovations.

Job availability

The job outlook for front-end developers is expected to grow 16% from 2022 to 2032, which is much faster than the average for other industries. This isabout 19,000 job openings per year over the next decade, with many openings expected due to retirements and other job changes.

Average salary for front-end developers

₹6,80,000 per year

Frontend Developer Salaries in India

The average salary for Frontend Developer is ₹6,80,000 per year in the India. The average additional cash compensation for a Frontend Developer in the India is ₹80,000, with a range from ₹23,000 - ₹1,10,000.

Feature of web developer

Proficiency in HTML, CSS, and JavaScript to build responsive and interactive web pages.

Understanding of user interface and user experience principles to create visually appealing and user-friendly interfaces.

Ability to create responsive designs that adapt to different screen sizes and devices.

Knowledge of accessibility guidelines to ensure that web pages are accessible to people with disabilities.

Understanding of different browsers and their compatibility issues to ensure that web pages work across multiple browsers.

Developer tools and software

Visual Studio Code (VS Code): A popular, open-source code editor with a wide range of extensions.

Sublime Text: A fast and feature-rich code editor with a large user base.

Atom: A customizable, open-source code editor with a large community of developers.

Brackets: A free, open-source code editor specifically designed for front-end development.

WebStorm: A commercial IDE with advanced features for front-end development.

How to become a front-end developer?

As a new front end developer, start by mastering HTML,CSS, and Javascript. These core languages form the foundation of front end development. Learn to create structured content with HTML, style it with CSS, and add interactivity with JavaScript

Backend

1)What is the backend?

The backend of a software application refers to the server-side part that handles the logic, database interactions, authentication, and other behind-the-scenes operations.

It's responsible for :

- processing requests from the frontend (the user interface),
- managing data, and
- performing the core functions of the application.

In web development, the backend typically involves:

1. **Server**: The computer or service where the backend code runs and where data is processed.
2. **Database**: Where data is stored and retrieved from. This can be SQL (like MySQL, PostgreSQL) or NoSQL (like MongoDB, Redis).
3. **Application Logic**: The code that performs tasks such as data processing, authentication, and business logic. This code is often written in languages like Python, Java, Ruby, or Node.js.

The backend interacts with the frontend (client-side) through APIs (Application Programming Interfaces) or direct data exchanges to ensure that user requests are properly handled and that the application functions as expected.

2)Why do we need a backend?

1. **Data Management**: The backend handles the storage, retrieval, and manipulation of data. For example, when you make a transaction on a banking app, the backend processes and stores this information securely.
2. **Business Logic**: It executes the core logic of the application. This includes calculations, data processing, and decision-making that drive the functionality of the application.
3. **Security**: The backend manages authentication and authorization, ensuring that only authorized users can access certain data or functionalities.
4. **Performance**: It helps manage and optimize resources, handle complex computations, and deliver responses efficiently, offloading these tasks from the client-side.
5. **Scalability**: The backend can be designed to handle increased loads by scaling up or out, ensuring that the application can grow and handle more users or data as needed.
6. **Integration**: It enables integration with other services, APIs, or external systems, allowing for more complex functionalities and interactions.
7. **Consistency**: It ensures that data and application logic are consistent across different users and devices, providing a unified experience.

Overall, the backend is crucial for ensuring that the application operates smoothly, securely, and efficiently, providing a solid foundation for the frontend and user experience.

3)When do we need it?

1. **Data Storage and Management**: When your application requires persistent data storage, such as user profiles, transaction history, or content management.
2. **Complex Business Logic**: When you need to implement and execute complex business rules or processes that can’t be handled efficiently on the client side.
3. **User Authentication and Authorization**: When you need to manage user accounts, ensure secure login processes, and control access to different parts of your application.
4. **Interactivity**: When your application needs to interact with other services or APIs, such as payment gateways, third-party integrations, or real-time data feeds.
5. **Scalability**: When you expect your application to handle a large number of users or high volumes of data, and you need a robust system to manage and scale this load.
6. **Security**: When you need to handle sensitive data securely, implement encryption, and protect against vulnerabilities that can’t be managed solely on the client side.
7. **Performance**: When you need to offload heavy computational tasks from the client device to the server to enhance performance and user experience.

In summary, a backend is necessary when your application needs to manage data, execute complex logic, handle user authentication, integrate with other services, scale effectively, ensure security, or enhance performance.

Explain each blocks

Backend (Kitchen)

In this culinary analogy, the backend represents the kitchen where the food is prepared and served. Just as a kitchen receives orders, prepares meals, and serves them to the diners, the backend server receives requests, processes data, and sends responses to the users.

Kitchen Staff (Server-side Languages): The kitchen staff, such as chefs and cooks, are like the server-side languages like Node.js, Python, Ruby, etc. that execute the recipes (code) to prepare the meals (data).

Recipes (APIs and Endpoints): The recipes used in the kitchen are like the APIs and endpoints that define how the data is prepared and served. Each recipe has a specific set of instructions (code) that the kitchen staff follows to prepare the meal.

Cooking Utensils (Frameworks and Libraries): The cooking utensils, such as pots, pans, and utensils, are like the frameworks and libraries that provide tools and functionality to the kitchen staff to prepare the meals.

Database (Storeroom): In this culinary analogy, the database represents the storeroom where the ingredients and supplies are stored. Just as a storeroom provides the necessary ingredients and supplies for the kitchen to prepare meals, the database provides the necessary data for the backend server to process and serve.

Ingredients (Data): The ingredients stored in the storeroom are like the data stored in the database. The kitchen staff retrieves the necessary ingredients (data) to prepare the meals (responses).

Shelves and Racks (Database Schema): The shelves and racks in the storeroom are like the database schema that organizes and structures the data for efficient retrieval and storage.

Users (Diners in Each Room): In this culinary analogy, the users represent the diners in each room, each representing a different country. Just as diners place orders and receive their meals, users send requests and receive responses from the backend server.

Orders (Requests): The orders placed by the diners are like the requests sent by the users to the backend server. Each order specifies what meal (data) the diner wants to receive.

Meal Delivery (Response): The meal delivered to the diner is like the response sent by the backend server to the user. The response contains the prepared data (meal) that the user requested.

Request-Response Cycle

Here's how the request-response cycle works in this analogy:

- ❖ A diner (user) in a room (country) places an order (request) for a meal (data) to the kitchen (backend server).
- ❖ The kitchen staff (server-side language) receives the order and retrieves the necessary ingredients (data) from the storeroom (database).
- ❖ The kitchen staff prepares the meal (processes the data) according to the recipe (API and endpoint) and cooking utensils (frameworks and libraries).
- ❖ The kitchen staff serves the meal (sends the response) to the diner (user) in their room (country).

And that's how the backend server, database, and users interact in this culinary analogy!

4)An Example of E-Commerce project and Explain How its implemented

Project Overview

Project: Online E-commerce Platform

Key Features:

- User registration and login
- Product catalog
- Shopping cart
- Order processing
- Payment integration
- User reviews and ratings

User Groups:

1. **Buyers**: Customers who can browse products, add them to their cart, and make purchases.
2. **Sellers**: Users who can list their products, manage inventory, and view their sales.

1. Project Setup

Block 1.1: Initialize Project and Environment

- **Objective**: Establish the project environment and configuration.
- **Steps**:
    1. **Install Django**: Use a package manager to install Django.
    2. **Create Project**: Generate a new Django project using Django's command-line tool.
    3. **Create App**: Within the project, create an application (e.g., `store`) to handle e-commerce functionality.
    4. **Configure Settings**: Add the new app to the project's settings to make it available.

2. Define Models

Block 2.1: Create Models

- **Objective**: Design the database schema to support the e-commerce functionality.
- **Components**:
    1. **User Model**: Extend Django's default user model to differentiate between buyers and sellers.
        - **Buyers**: Users who can purchase products.
        - **Sellers**: Users who can list and manage products.
    2. **Product Model**: Represents products available for sale, including attributes such as name, description, price, and stock.
    3. **Cart and CartItem Models**: Manage shopping carts for buyers, including the products and quantities in the cart.
    4. **Order and OrderItem Models**: Handle orders placed by buyers, including the products and quantities ordered.

Block 2.2: Migrations

- **Objective**: Apply the model definitions to the database.
- **Steps**:
    1. **Create Migrations**: Generate migration files based on the models.
    2. **Apply Migrations**: Update the database schema to reflect the model changes.

3. Implement Views

Block 3.1: Develop Views

- **Objective**: Implement the logic to handle user requests and interact with the database.
- **Components**:
    1. **Product Management**: Views to display a list of products and detailed information about individual products.
    2. **Cart Management**: Views to add products to the cart, view cart contents, and manage cart operations.
    3. **Order Processing**: Views to handle the checkout process, including order creation and confirmation.

Block 3.2: Templates

- **Objective**: Create HTML templates to render the user interface.
- **Components**:
    1. **Product List Template**: Shows all available products, typically with options to view details or add to cart.
    2. **Product Detail Template**: Displays detailed information about a single product.
    3. **Cart Template**: Lists products in the cart with options to modify quantities or proceed to checkout.
    4. **Checkout Template**: Provides an order summary and confirmation after a purchase.

4. Define URLs

Block 4.1: URL Configuration

- **Objective**: Map URLs to the appropriate views.
- **Components**:
    1. **Store URLs**: Define URL patterns for accessing product lists, product details, cart management, and checkout.
    2. **Project URLs**: Include the store URLs in the main project URL configuration to integrate the e-commerce functionality into the overall project.

5. Implement Authentication

Block 5.1: User Management

- **Objective**: Handle user authentication and account management.
- **Components**:
    1. **Custom User Model**: Customize the user model to include roles (buyer or seller) and additional attributes.
    2. **Authentication Views**: Provide views for user registration, login, and logout.

Block 5.2: Registration and Login

- **Objective**: Enable users to create accounts and access the system.

- **Components**:
    1. **Registration View**: Allows new users to register as buyers or sellers.
    2. **Login/Logout Views**: Manage user sessions, allowing users to log in and out of their accounts.

6. Testing and Validation

Block 6.1: Test Functionality

- **Objective**: Ensure that all features work as expected and the application is free of bugs.
- **Components**:
    1. **Unit Tests**: Write tests for individual components such as models, views, and forms.
    2. **Integration Tests**: Test interactions between different components of the system to ensure they work together correctly.

7. Deployment

Block 7.1: Prepare for Production

- **Objective**: Make the application ready for deployment on a live server.
- **Components**:
    1. **Settings Configuration**: Update settings for security, performance, and production environment requirements.
    2. **Static Files Management**: Ensure that static files (CSS, JavaScript) are properly managed and served.
    3. **Web Server Setup**: Configure a web server (e.g., Gunicorn) and reverse proxy (e.g., Nginx) to handle requests and serve the application.

By following these blocks, you can systematically build and deploy an e-commerce website with Django that supports both buyers and sellers, ensuring each component is implemented and integrated effectively.

5)Working Flow and Process of Backend Development

1. Planning and Design

- Define project requirements and goals to ensure a clear understanding of the objectives.
- Create a detailed design document that outlines the architecture, technology stack, and database schema.

2. Coding

- Write clean and modular code following best practices to ensure maintainability.
- Implement the necessary business logic and algorithms to meet the project's requirements.

3. Testing

- Develop and execute unit, integration, and end-to-end tests to validate functionality and quality.
- Utilize testing tools like JUnit or Postman to identify and fix bugs.

4. Deployment

- Configure the production environment, including servers and databases, for smooth deployment.
- Implement CI/CD strategies to automate deployment and minimize downtime.

5. Maintenance

- Monitor application performance and logs to optimize resources and identify improvement areas.
- Regularly apply updates, patches, and security fixes to keep the application secure.

6. Debugging and Troubleshooting

- Use debugging tools and logging to pinpoint and resolve code issues.
- Analyze error logs and system metrics to diagnose problems and enhance system reliability.

The Backend Process

When a user interacts with a web application, the following process occurs:

1. **Request**: The user's web browser sends an HTTP request to the server, which can be a GET, POST, PUT, DELETE, etc.

2. **Server:** The server receives the request and passes it to the backend application.
3. **Backend Application**: The backend application processes the request, which involves:
    - **Routing**: The request is routed to the appropriate controller or handler function.
    - **Authentication and Authorization**: The backend checks if the user is authenticated and authorized to access the requested resource.
    - **Database Interaction**: The backend interacts with the database to retrieve or update data.
    - **Business Logic**: The backend applies business logic to the data, such as calculations, validations, and transformations.
    - **Response Generation**: The backend generates a response to the request, which can be in the form of HTML, JSON, XML, etc.
4. **Response:** The backend sends the response back to the server.
5. **Server:** The server sends the response back to the user's web browser.
6. **Browser:** The web browser receives the response and renders the updated page to the user.

Backend interacts with frontend and database :

Request-Response cycle:

1. **Frontend Request**: The frontend sends a request to the backend, typically using HTTP methods like GET, POST, PUT, or DELETE.
2. **Backend Processing**: The backend receives the request, processes it, and determines what action to take.
3. **Database Interaction**: If necessary, the backend interacts with the database to retrieve or update data.
4. **Response Generation**: The backend generates a response based on the request and database interaction.
5. **Response Sent to Frontend**: The backend sends the response back to the frontend, which then renders the updated UI.

6)What are the tools backend developers use ?

Backend developers rely on a range of tools to build, test, and maintain web applications. Here's an overview of some common tools categorized by their functions:

1. Programming Languages

Backend programming languages are used to write the server-side logic that powers web applications. These languages enable developers to handle requests, interact with databases, and perform various backend operations.

Python: Known for its readability and simplicity, often used with frameworks like Django and Flask.

PHP: A widely-used language for web development, especially in content management systems like WordPress.

JavaScript (Node.js): JavaScript is used on the server-side with Node.js, allowing developers to use a single language for both frontend and backend.

Ruby: Known for its elegant syntax, used with the Ruby on Rails framework.

Java: A powerful, object-oriented language often used in large-scale enterprise applications, frequently paired with the Spring framework.

C#: A language developed by Microsoft, commonly used with the ASP.NET framework for web applications.

2. Frameworks

Frameworks provide a structured environment for building web applications, offering pre-built components and libraries that streamline development.

Django: A Python framework that encourages rapid development and clean design.

Laravel: A PHP framework that simplifies tasks like routing and authentication.

Spring: A comprehensive Java framework for building scalable enterprise applications.

Ruby on Rails: A Ruby framework that emphasizes simplicity and convention over configuration.

Node.js: A runtime environment for executing JavaScript on the server, often used with Express.js.

Flask: A lightweight Python framework for small to medium-sized applications.

3. Databases

Databases store and manage the data that web applications rely on. They can be relational or non-relational, depending on the type of data and how it needs to be accessed.

SQL (Relational)

MySQL: A popular relational database known for its reliability and performance.

PostgreSQL: An open-source relational database known for its advanced features and compliance with SQL standards.

SQLite: A lightweight, file-based database often used in smaller projects or for development purposes.

NoSQL (Non-Relational)

MongoDB: A NoSQL database that stores data in flexible, JSON-like documents.

Redis: An in-memory data structure store often used as a database, cache, and message broker.

4. Servers:

Servers host the backend applications and manage incoming requests from clients, serving responses based on the application's logic.

Apache: A widely-used web server known for its flexibility and performance.

NGINX: A high-performance web server and reverse proxy server known for its ability to handle large amounts of concurrent connections.

Microsoft IIS: A web server created by Microsoft, commonly used in conjunction with ASP.NET applications.

Lighttpd: A lightweight web server optimized for speed-critical environments.

5. Testing and Debugging Tools:

These tools help ensure the quality of the code by finding and fixing bugs.

Postman: A tool for testing APIs by sending HTTP requests and viewing responses.

PyTest: A Python testing framework used for writing simple as well as scalable test cases.

JUnit: A testing framework for Java, commonly used for unit testing.

Sentry: An error tracking tool that helps developers monitor and fix crashes in real time.

6. Deployment Tools

These tools are used to deploy applications to servers, manage environments, and automate deployment processes.

Docker: A platform for automating the deployment of applications inside lightweight containers.

Kubernetes: An open-source system for automating the deployment, scaling, and management of containerized applications.

Jenkins: A continuous integration tool that automates the process of building, testing, and deploying code.

7. API Development Tools

APIs are a crucial part of backend development, enabling communication between different software components.

Swagger: A tool for designing, building, and documenting RESTful APIs.

OpenAPI: A specification for building APIs that are easy to understand and consume.

Backend Developer

1)Roles and Responsibilities of a Backend developer?

A backend developer plays a crucial role in building and maintaining the server-side of web applications. Their responsibilities focus on ensuring that the server, application, and database interact correctly and efficiently. Here’s a breakdown of their key roles and responsibilities:

1. Server-Side Logic and Application Development

- **Design and Implementation**: Create and maintain the core functionality of the application, including APIs, server-side logic, and integrations with databases and other services.
- **Business Logic**: Implement the rules and processes that drive the application's behavior, ensuring that the server-side code correctly processes data and handles user requests.

2. Database Management and Integration

- **Schema Design**: Design and implement database schemas that efficiently store and retrieve data.
- **Queries and Optimization**: Write and optimize database queries to ensure fast performance and scalability.
- **Data Integrity**: Ensure that data is accurate, consistent, and securely managed.

3. API Development and Management

- **API Design**: Develop RESTful or GraphQL APIs that allow the frontend to interact with the backend.

- **Documentation**: Provide clear and comprehensive API documentation for other developers and users.
- **Integration**: Ensure that APIs are properly integrated with frontend applications and third-party services.

4. Server and Infrastructure Management

- **Server Configuration**: Set up and configure servers, including choosing and maintaining the appropriate hardware and software.
- **Deployment**: Deploy applications to production environments and manage continuous integration and deployment (CI/CD) pipelines.
- **Monitoring**: Monitor server performance, identify issues, and perform troubleshooting as needed.

5. Security

- **Data Protection**: Implement security measures to protect data from unauthorized access, breaches, and other threats.
- **Authentication and Authorization**: Develop and manage authentication and authorization mechanisms to ensure that users can only access data and functionality they are permitted to.

6. Performance Optimization

- **Code Efficiency**: Optimize backend code to ensure fast execution and minimal latency.
- **Load Balancing**: Implement load balancing techniques to distribute traffic and improve application performance.

7. Collaboration and Communication

- **Team Coordination**: Work closely with frontend developers, designers, and other team members to ensure cohesive development and integration of features.
- **Stakeholder Communication**: Communicate technical details and progress to stakeholders and provide support as needed.

8. Maintenance and Support

- **Bug Fixes**: Identify, troubleshoot, and fix bugs or issues in the backend code.
- **Updates and Upgrades**: Apply updates and upgrades to backend systems to keep them current and secure.

9. Documentation

- **Code Documentation**: Write and maintain documentation for backend code, including inline comments and external documentation.
- **System Documentation**: Document system architecture, APIs, and server configurations.

10. Testing

- **Unit Testing**: Write and execute unit tests to verify that individual components of the backend function correctly.
- **Integration Testing**: Test the interaction between different components of the application to ensure they work together as expected.

By handling these responsibilities, a backend developer ensures that the server-side of the application is robust, efficient, secure, and capable of meeting the needs of the frontend and its users.

2)Job Availability and payscale in industry?

The job availability for backend developers is strong and continues to grow across various sectors. Here's an overview of the current job market and trends for backend developers:

1. Industry Demand

- **Technology Sector**: There is high demand from major tech companies (e.g., Google, Amazon, Microsoft) and startups. Technology companies often require backend developers to build and maintain scalable systems.
- **E-Commerce**: Companies in this sector, such as Shopify, eBay, and Amazon, need backend developers to handle transactions, user data, and product management.
- **Finance and Banking**: Financial institutions and fintech startups need backend developers for secure transaction processing and data management.
- **Healthcare**: Healthcare organizations require backend developers to manage patient data, electronic health records (EHRs), and telemedicine systems.

2. Job Titles and Roles

- **Backend Developer**: Focuses on server-side development, including database management and API development.
- **Full-Stack Developer**: Handles both frontend and backend tasks, providing versatility and broader skill sets.
- **API Developer**: Specializes in creating and managing APIs for integration with other services or applications.
- **Database Administrator (DBA)**: Manages database systems, often with a strong backend development background.

3. Work Environments

- **Full-Time Positions**: Common across many industries, offering stability and benefits.
- **Contract and Freelance Work**: Available for specific projects or short-term needs, offering flexibility.
- **Remote Work**: Many companies offer remote positions for backend developers, allowing for geographic flexibility.

4.Backend Developer Salary and Career Growth

In India, the payscale for backend developers can vary significantly based on experience, location, and the size of the company. Here's a general overview:

1. Entry-Level (0-2 Years of Experience)

- **Average Salary**: ₹400,000 - ₹700,000 per year
- **Typical Range**: ₹300,000 - ₹800,000 per year

2. Mid-Level (3-5 Years of Experience)

- **Average Salary**: ₹700,000 - ₹1,200,000 per year
- **Typical Range**: ₹600,000 - ₹1,500,000 per year

3. Senior-Level (5+ Years of Experience)

- **Average Salary**: ₹1,200,000 - ₹2,500,000+ per year
- **Typical Range**: ₹1,000,000 - ₹3,000,000+ per year

4. Specialized Roles and Locations

- **Tech Hubs** (e.g., Bangalore, Hyderabad, Mumbai): Salaries can be higher in major tech hubs due to higher demand and cost of living. Senior backend developers in these cities might earn ₹1,500,000 - ₹3,000,000+ per year.
- **Specialized Skills**: Expertise in high-demand technologies (e.g., cloud computing, big data, machine learning) can lead to higher salaries.

5. Freelance and Contract Work

- **Freelance Rates**: Backend developers working as freelancers or contractors may earn between ₹500 - ₹2,000 per hour, depending on their expertise and the project's complexity.

Additional Factors Affecting Pay

- **Company Size**: Large tech companies, multinational firms, and startups with substantial funding often offer higher salaries and additional perks.
- **Education and Certifications**: Advanced degrees or certifications in specialized areas can influence salary levels.
- **Experience and Expertise**: Extensive experience and expertise in emerging technologies or specific frameworks can lead to higher compensation.

➢ **Career Progression:**

  ○ Start as a junior developer, progress to mid-level, then senior developer, and potentially to roles like lead developer or CTO.

➢ **Factors Influencing Salary:**

  ○ Experience, location, industry, and technical expertise.

3)Roadmap

Steps	Descriptions
Requirement gathering	1. gathering business requirements from business team 2. create modules based on the requirements 3. create an project architecture(folder setup) 4. list what are the 3rd party modules or subscriptions required 5. create API endpoints documentation based on the UI(include frontend team)
Initial Setup	1. list common security practices 2. create initial DB schema and add pre-addable data 3. add containerization setup using docker 4. list and create reusable functions used in the project using django management command 5. install all debugging related 3rd party modules 6. creating and managing 3rd party tokens and keys
Development	1. create a endpoint 2. do validations on received fields 3. list performance monitoring like logging, request response timing 4. handling background tasks

	5. triggering 3rd party service or db rollback on request failures
Testing	1. writing test cases based on business logic
Deployment	1. getting familiar with aws services like ec2,s3, sms,mailing 2. getting familiar with linux
study about	1. low level designing 2. high level designing 3. creating reusable packages 4. list most used packages in particular technology 5. managing a doc for listing all available resource links about particular technology

4)What is Django?

Django :

Django is a free and open-source web framework that provides an architecture, templates, and APIs to build web applications quickly and efficiently. It's built on top of Python and provides many built-in features, including:

Key Features of Django :

➢ **ORM (Object-Relational Mapping)**: Allows you to interact with databases using Python code instead of writing SQL queries.

➢ **Admin Interface:** Automatically generates an admin interface for managing application data.

➢ **Form Handling:** Simplifies form creation, validation, and processing.

➢ **Security:** Comes with built-in protections against common web vulnerabilities like SQL injection, XSS, and CSRF attacks.

➢ **Scalability:** Capable of handling high-traffic websites.

5)Why is django used?

Django is used for web development because it provides a comprehensive set of tools and features that streamline the process of building robust, scalable, and secure web applications. Here’s why Django is favored by developers:

1. Rapid Development

- **Built-in Features**: Django comes with many built-in features (like authentication, admin interface, and form handling) that allow developers to quickly build and deploy applications without needing to reinvent the wheel.
- **Scaffolded Framework**: Its “batteries-included” approach provides a lot of functionality out of the box, which accelerates development.

2. Security

- **Protection Against Common Vulnerabilities**: Django includes built-in protection against common web vulnerabilities such as SQL injection, cross-site scripting (XSS), cross-site request forgery (CSRF), and clickjacking.
- **Secure Authentication**: Django’s authentication system is robust and includes password hashing, user session management, and other security features.

3. Scalability

- **Efficient Scaling**: Django is designed to scale with applications, whether you’re building a small project or a large, complex application. It supports caching, database optimization, and load balancing.
- **Modular Design**: Django’s architecture encourages modular design, making it easier to manage and scale applications as they grow.

4. Maintainability

- **Clean and Readable Code**: Django’s design promotes clean, readable, and maintainable code, following best practices and the DRY (Don’t Repeat Yourself) principle.
- **Structured Projects**: The framework enforces a structured approach to organizing code, which helps in maintaining and updating projects over time.

5. Built-in Admin Interface

- **Automatic Admin Interface**: Django automatically generates a powerful and customizable admin interface based on your models, allowing for easy management of application data without additional coding.

- **Ease of Use**: This interface is user-friendly and helps developers and administrators manage content and user data efficiently.

6. ORM (Object-Relational Mapping)

- **Database Abstraction**: Django's ORM allows developers to interact with the database using Python code instead of raw SQL. This abstraction simplifies database queries and management.
- **Database Migration**: Django provides built-in tools for managing database schema changes and migrations, making it easier to evolve the database schema over time.

7. Community and Ecosystem

- **Active Community**: Django has a large and active community that contributes to its development, provides support, and creates a wealth of third-party packages and extensions.
- **Extensive Documentation**: The framework offers thorough and well-organized documentation, which helps developers understand and utilize its features effectively.

8. Versatility

- **Wide Range of Use Cases**: Django can be used to build various types of web applications, including content management systems, e-commerce platforms, social networks, and RESTful APIs.
- **Customization**: While Django provides many built-in features, it also allows for extensive customization to meet specific application requirements.

9. Testing and Debugging

- **Testing Framework**: Django includes a built-in testing framework that supports unit tests, integration tests, and other types of testing. This helps ensure that applications are reliable and function correctly.
- **Debugging Tools**: The framework provides useful debugging tools and error reporting, which aid in diagnosing and fixing issues during development.

Overall, Django is used because it provides a comprehensive, secure, and efficient framework for building web applications, helping developers to create high-quality software quickly and maintainably.

6)When do we need Django?

1. **You Need to Build a Website Quickly**: If you have a tight deadline and need to get a web application up and running fast, Django’s built-in features and tools help you build and deploy quickly.
2. **Your Project is Complex**: If your application needs lots of features like user accounts, admin interfaces, and data management, Django provides a lot of these out of the box.
3. **Security is Important**: When your application handles sensitive information or requires strong security measures, Django’s built-in protections help keep your data safe.
4. **You Need a Custom Content Management System**: If you want a system to manage content easily (like articles, products, or user data), Django’s admin interface can be customized to fit your needs.
5. **You’re Building an Online Store**: For creating an e-commerce site with shopping carts, product catalogs, and payment systems, Django can handle these complex requirements.
6. **You Want to Create APIs**: If you need to build APIs that other applications or services will use, Django, along with Django REST framework, makes it easier to create and manage these APIs.
7. **You Want a Well-Organized Codebase**: If you prefer a structured approach to web development that separates data, logic, and presentation, Django’s framework helps you organize your code cleanly.

In short, use Django when you need a robust, secure, and scalable web framework that helps you build complex applications efficiently and with best practices in mind.

Cloud computing

When We need it

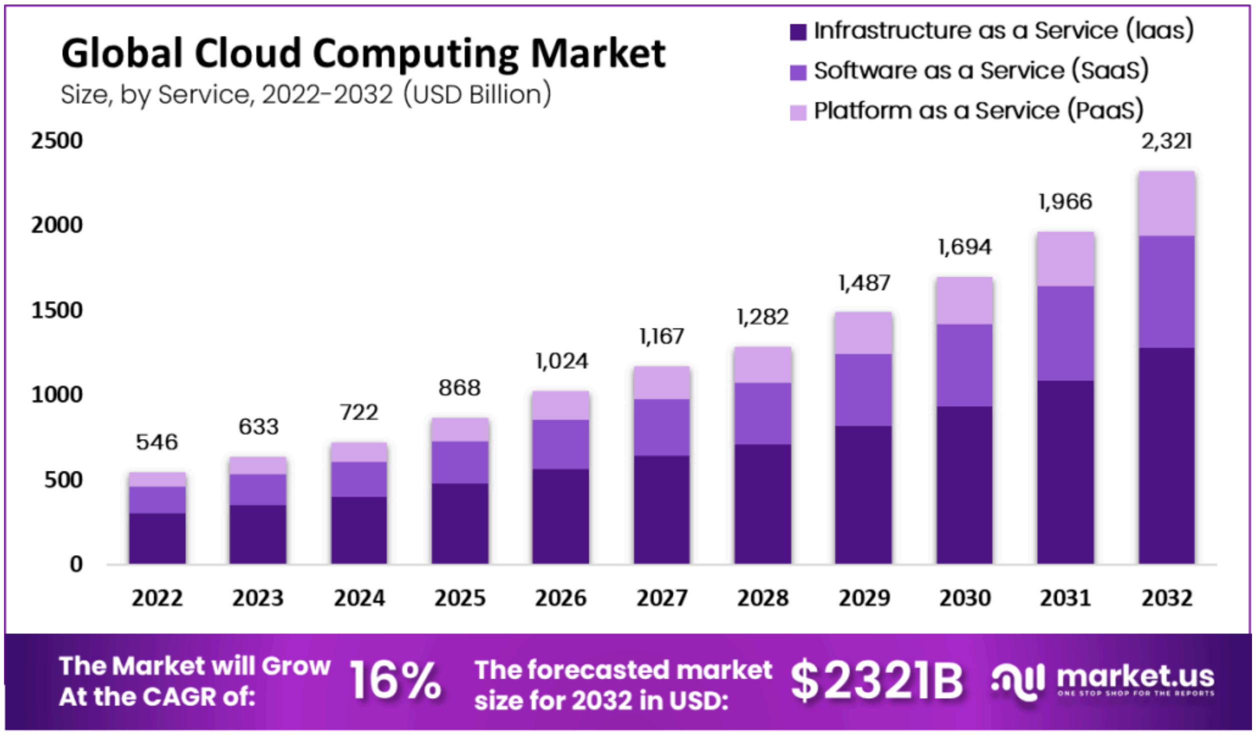
- Cloud computing gives your business more flexibility.
- You can quickly scale resources and storage up to meet business demands without having to invest in physical infrastructure.
- Companies don't need to pay for or build the infrastructure needed to support their highest load levels.

What is cloud computing

- Cloud computing is the delivery of computing services—including servers, storage, databases, networking, software, analytics, and intelligence—over the Internet (“the cloud”) to offer faster innovation, flexible resources, and economies of scale.
- Instead of buying, owning, and maintaining physical data centers and servers, one can access technology services, such as computing power, storage, and databases, on an as-needed basis from a cloud server.
- Cloud computing offers platform independency, that means software is not required to be installed in a local PC.

Example: data storage, servers, databases, networking, software over the internet.

Statistics



Cloud architecture

Components

- Infrastructure as a service (IaaS) - Empty Rental House

For IaaS models, the service provider hosts, maintains, and updates the backend infrastructure, such as compute, storage, networking, and virtualization. You manage everything else including the operating system, middleware, data, and applications.

IaaS examples: Amazon Web Services (AWS) Elastic Compute Cloud (EC2), Microsoft Azure, Google Compute Engine (GCE)

- Platform as a service (PaaS) - Partially Furnished Rental House

Like IaaS models, for PaaS models, the service provider delivers and manages the backend infrastructure. However, PaaS models provide all the software features and tools needed for application development. You still have to write the code and manage your apps and data but do not have to worry about managing or maintaining the software development platform.

PaaS examples: AWS Elastic Beanstalk, Google App Engine, and Adobe Commerce

- Software as a service (SaaS) - Fully Furnished Rental House

With SaaS service models, the service provider delivers the entire application stack—the complete application and all the infrastructure needed to deliver it. As a customer, all you have to do is connect to the app through the internet—the provider is responsible for everything else.

SaaS examples: Gmail, Slack, and Microsoft Office 365

Types of cloud computing

Public cloud

Description: Refers to services provided by cloud service providers (CSPs) that are available to the general public over the Internet.
Benefits: Cost-effective, scalable, and offer a wide range of services.

Private cloud

Description: Involves dedicated infrastructure and resources that a single organization solely uses.
Benefits: Provides greater control, security, and customization options

Hybrid cloud

Description: A combination of both public and private cloud environments.
Benefits: Enables seamless data and workload portability between different cloud environments.

Advantage of cloud computing

1.Flexibility

Cloud computing provides flexibility in terms of accessing and managing resources, applications, and data from anywhere, at any time, and on any device with an internet connection. This allows users to work remotely, collaborate with others, and access information from different locations.

2. Low Cost

Cloud computing reduces the need for upfront capital expenditures on hardware, software, and infrastructure. Instead, users pay only for the resources they use, which can lead to significant cost savings. Additionally, cloud providers handle maintenance, upgrades, and support, reducing the need for in-house IT staff and resources.

3. Speed and Scale

Cloud computing enables rapid deployment and scaling of resources, applications, and services. This means that users can quickly respond to changing business needs, deploy new applications, and scale up or down to match demand. Cloud providers offer a vast pool of resources that can be quickly provisioned and de-provisioned as needed.

4. Easier Management of Data and Information

Cloud computing provides a centralized platform for managing data and information, making it easier to store, process, and analyze large amounts of data. Cloud-based data management tools and services help to simplify data integration, reduce data silos, and improve data governance.

5. Data Diversity

Cloud computing enables users to store and process diverse types of data, including structured, unstructured, and semi-structured data. This allows for more comprehensive data analysis, better decision-making, and improved business outcomes.

6. Increased Storage Capacity

Cloud computing provides virtually unlimited storage capacity, allowing users to store large amounts of data without worrying about running out of space. This is particularly useful for organizations that generate large amounts of data, such as those in the fields of science, engineering, and finance.

7. Easy to Learn and Understand

Cloud computing is designed to be user-friendly and accessible, with intuitive interfaces and minimal technical requirements. This makes it easier for non-technical users to learn and understand cloud computing concepts and technologies.

8. Automatic Updating

Cloud providers handle software updates, patches, and maintenance, ensuring that users always have access to the latest versions of applications and services. This reduces the burden on in-house IT staff and minimizes the risk of security breaches and downtime.

9. Customize Settings

Cloud computing allows users to customize settings, configurations, and applications to meet their specific business needs. This includes setting up custom workflows, integrating with other systems, and tailoring security and access controls to meet organizational requirements.

Disadvantage of cloud computing

1. Dependency

Cloud computing makes organizations dependent on the cloud provider's infrastructure, services, and support. This means that if the provider experiences downtime, outages, or service disruptions, the organization's operations may be severely impacted. Additionally, if the provider changes their terms of service or pricing, the organization may be forced to adapt or find a new provider.

2. Risk

Cloud computing introduces new risks, such as:

- Data breaches and cyber attacks
- Unauthorized access to sensitive data
- Data loss or corruption
- Compliance and regulatory issues

- Vendor lock-in (difficulty switching to a different provider)

These risks can be mitigated with proper security measures, data encryption, and due diligence in selecting a cloud provider.

3. Requires a constant internet connection

Cloud computing requires a stable and fast internet connection to access and use cloud-based resources. This can be a problem for organizations with:

- Unreliable or slow internet connectivity
- Remote or mobile workers with limited internet access
- Critical applications that require low latency and high availability

In such cases, cloud computing may not be the best fit, and alternative solutions like on-premises infrastructure or edge computing may be more suitable.

4. Security

Cloud computing security is a major concern, as sensitive data is stored and processed outside the organization's premises. Key security concerns include:

- Data encryption and access controls
- Identity and access management
- Network security and firewalls
- Compliance with regulatory requirements (e.g., GDPR, HIPAA)

To address these concerns, organizations must implement robust security measures, such as multi-factor authentication, encryption, and regular security audits.

5. Migration issues

Migrating to the cloud can be a complex and challenging process, especially for large or complex applications. Common migration issues include:

- Data migration and integration
- Application compatibility and refactoring
- Network and infrastructure changes
- Training and adoption for end-users

To overcome these challenges, organizations should develop a clear migration strategy, assess application readiness, and plan for potential roadblocks and contingencies.

These disadvantages highlight the importance of carefully evaluating the benefits and drawbacks of cloud computing, assessing organizational readiness, and developing a comprehensive cloud strategy that addresses potential risks and challenges.

World wide Top 13 cloud providers

1. Amazon Web Services (AWS) – Infrastructure-as-a-Service (IaaS)
2. Microsoft Azure – Hybrid cloud and enterprise cloud services
3. Google Cloud Platform – AI, ML, and Kubernetes
4. Alibaba — Largest cloud service provider in Asia
5. IBM Cloud – Multi-cloud CSP
6. DigitalOcean Cloud – Cloud computing tools for SMBs and developers
7. Salesforce Cloud – Cloud-hosted services
8. Tencent Cloud
9. Oracle Cloud Infrastructure (OCI) – All-around CSP with a focus on databases, custom apps
10. Huawei Cloud – Going Global
11. Dell Technologies Cloud – Virtualization Via VMware
12. Cisco Cloud Solutions – Best Used For Hybrid Cloud Strategies
13. Rackspace – Best For Unpredictable Cloud Needs

Jobs in Cloud computing

1. Cloud administrator

These experts manage a company's cloud presence and infrastructure. They develop, enforce and update policies for how employees and users access cloud services; establish security protocols and policies; monitor and ensure uptime; and assess the need for technology updates.

Salary range: $68,051 to $84,571.

Average salary: $75,795.

2. Cloud support engineer

Where there's technology, problems inevitably arise. Cloud support engineers are troubleshooting experts who assist customers, typically B2B clients and not the ultimate end user. In addition to providing on-demand assistance, cloud support engineers are often responsible for writing user and training manuals, tutorials, FAQs and help guides. They should have excellent communication skills and prior experience in tech support and debugging.

Salary range: $77,000 to $116,000.

Average salary: $87,974.

3. Cloud security analyst

Cloud security analysts have the responsibility of ensuring the integrity and security of a company's cloud presence. They do this by assessing threats and shoring up defenses against them, preventing data breaches, securing data and eliminating security gaps if a breach occurs.

Salary range: $93,000 to $147,000.

Average salary: $98,228.

4. Cloud network engineer

In this role, IT professionals design and maintain an organization's cloud services across a network in a certain division or even for an entire company. Cloud network engineers' duties might overlap with cloud architects and engineers in that they are sometimes called upon to identify new cloud providers, assess business needs and make recommendations for cloud-based platforms.

Salary range: $83,846 to $102,123.

Average salary: $91,984.

5. Cloud software engineer

Cloud software engineers work with programmers and related computer scientists to develop software that operates in the cloud, often as SaaS or

IaaS systems. These individuals are usually also responsible for upgrading, repairing and maintaining the software they develop and the databases it powers.

Salary range: $131,000 to $195,000.

Average salary: $131,237.

6. Cloud automation engineer

As the world becomes increasingly automated, cloud automation engineers are necessary to build, implement and maintain this automation technology as it migrates to the cloud. This automation frees human workers from repetitive tasks.

Salary range: $90,000 to $132,000.

Average salary: $101,415.

7. Cloud engineer

Cloud engineers are responsible for the managerial aspects of a company's cloud strategies. Engineers often work alongside architects to ensure a company's cloud strategies are implemented, but they also do the administrative work of negotiating with clients and vendors to keep everyone on task and within budget.

Salary range: $125,498 to $147,958.

Average salary: $136,475.

8. Cloud consultant

Cloud consultants have broad knowledge of cloud technologies and provide guidance to companies looking for cloud-based tools. Typically, these experts assess a company's needs and suggest software and devices to best meet its technical and budgetary requirements. Consultants might also help transition to the cloud by designing migration policies and selecting appropriate platforms. Consultants might sometimes be asked to help customize a company's cloud presence, so they should have both general and in-depth knowledge of the major cloud platforms.

Salary range: $87,000 to $135,000.

Average salary: $98,900.

9. Cloud data scientist

Data scientists already analyze tremendous amounts of data. With the rapid expansion of big data and data generated by IoT applications, these professionals will be in even higher demand. Cloud data scientists develop logical management and organization systems for otherwise unwieldy amounts of data in cloud infrastructure environments. As organizations complete their digital transformation from physical to cloud-based servers, they'll need skilled cloud data scientists to assess and organize that data in a way that is sensible and usable. Note when searching for jobs that this position is also sometimes referred to as a *cloud data engineer*.

Salary range: $119,000 to $176,000.

Average salary: $119,584.

10. Cloud architect

Think of cloud architecture as the framework within which all other cloud technologies operate. It's the frame of the house, and all the cloud-specific subspecialties are like flooring, plumbing and drywall. Cloud architects, similar to general contractors, are the ones who design and implement a company's cloud computing strategies. They ensure everything stays on track and on budget and that the company's transition to cloud operations goes smoothly.

Salary range: $127,800 to $158,202.

Average salary: $143,665.