


Home > Course > Design Your Software Architecture Using Industry-Standard Patterns > Client-Server Architecture


Design Your Software Architecture Using Industry-Standard Patterns

4 hours  Hard

Last updated on 6/29/20



Client-Server Architecture

 [Log in](#) or [subscribe](#) for free to enjoy all this course has to offer!

In this chapter, we are going to cover the client-server architecture, what it is, and when you should use it. At the end of the chapter, I will show you a real-world example (IrisGold mining company), and then you will solve a similar case applying what you've learned.

What Is Client-Server Architecture?



Have you ever gone into a restaurant and ordered your food directly from the cook? Probably not. You sit at the table and wait until the waiter comes to take your order. You tell them what you want to eat, and they go into the kitchen and tell the cook. It makes sense. You don't necessarily use the right jargon for the cook, and the cook might not know what table to send your food to when it's done. The intermediary - the waiter - is effective here. Especially since the most important thing for you is that your order is correct. The cook doesn't care who you are as long as they get the instructions they need.

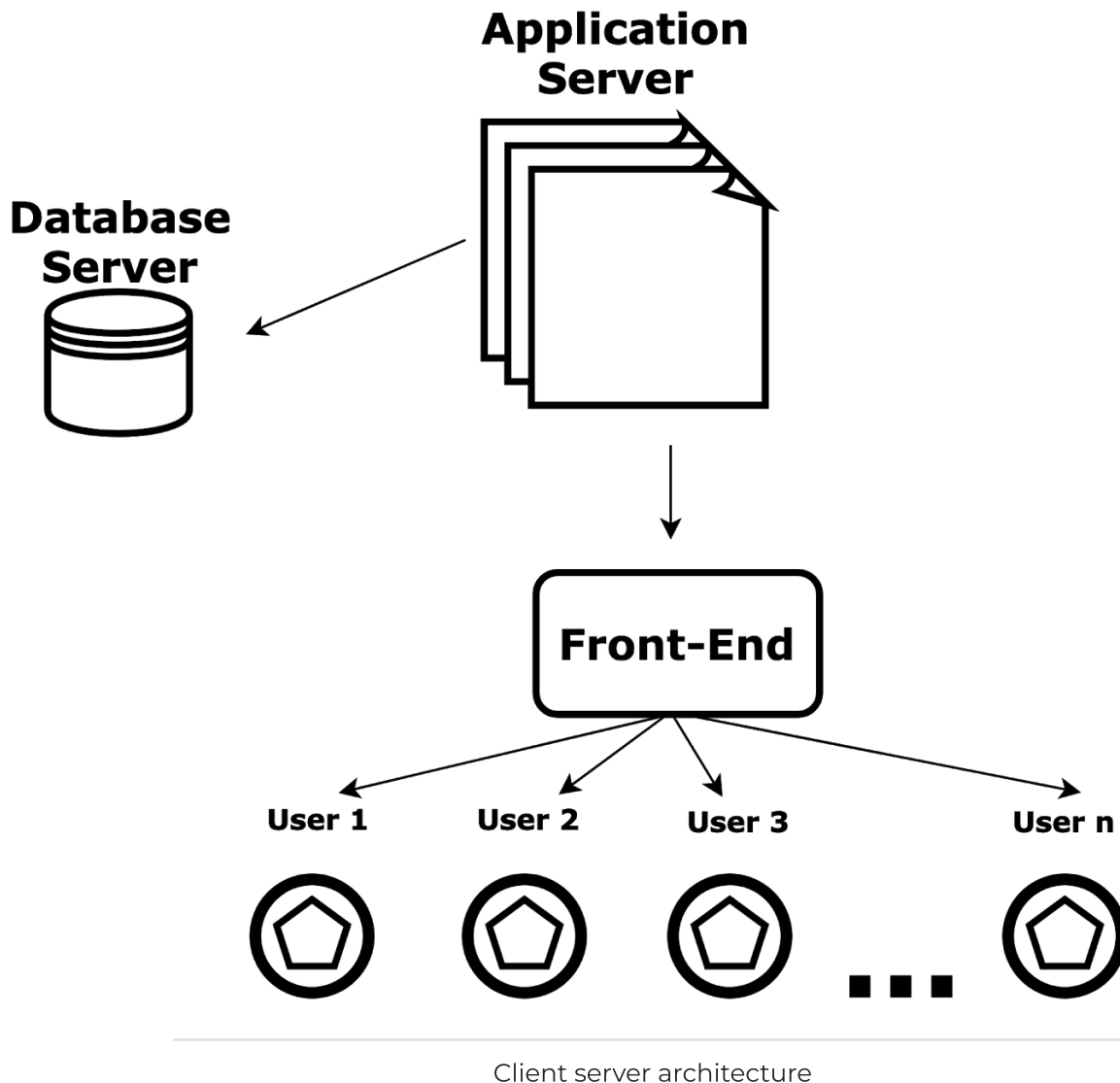
The client-server architecture works under the same principle: it partitions tasks between the providers of a service, called **servers**, and service requesters, called **clients**. A server is like the cook in our restaurant metaphor, and the client is the customer.

Just like a customer, clients usually initiate communication sessions with servers, which await incoming requests. The device (laptop, tablet, smartphone) a client uses to ask for service from the server is like the waiter in our example: it mediates between the service requester and the service provider and knows how to speak with both of them.

What Is the Structure of the Client-Server Architecture? ✓

Let's see what this looks like:

Client-Server Architecture High-Level Diagram



As you can see in the diagram above, a standard client-server architecture has three parts:

- **Front-End:** This is the piece of software that interacts with users, even if they are on different platforms with different technologies. Any front-end module in a client-server architecture is designed to interact with all existing devices on the market. This level contains the login screens, menus, data screens, and reports that give and take information to/from users. For example, most development tools and frameworks allow the creation of one version of a program that works for PCs, tablets, and phones.
- **Application server:** This is the server where the software modules of the application are installed. It connects to the database (called back-end) and interacts with users (called

front-end). The application server is like the waiter on our restaurant example.

- **Database server:** This server contains the tables, indexes, and data managed by the application. Searches and insert/delete/update operations are executed here.

How does this work in practice?

1. The user makes a request via the front-end, for example, “Bring me all customer invoices from January 1st until today, for customers that have bought product X.”
2. The application server gets this request and sends it to the database. The database server executes this request and sends it to the application server, which sends the result to the client using the front-end module.

There are several **advantages** to using client-server architecture:

- As you can see in the example, the client-server architecture separates hardware, software, and functionality of the system. For instance, if a software adaptation is needed in a particular country, i.e., a change in functionality is necessary. It can be adapted in the system without having to develop a version for phones, tablets, or laptops.
- Since it separates among hardware, software, and functionality of the system, only the front-end must be adapted to communicate with different devices.

There are also a few key **disadvantages**:

- If all the clients simultaneously request data from the server, it may get overloaded.
- If the server fails for any reason, then no user can use the system.

Now that you know how it works, let's look at how to use it!

When Would I Use the Client-Server Architecture?

▼

Here are some real-life situations that this would be useful for:

Example Name	Definition	Real-World Example	Advantages
Enterprise resource planning (ERP) system	An ERP is a big software system that contains all of the typical administrative functionalities most of the organizations use: accounts payable, accounts receivable, stock management, HR management, production management, provider management, purchasing, treasury, finance, accounting, etc. These	<ul style="list-style-type: none">• SAP (sap.com).• Oracle Business Suite (oracle.com).• Microsoft Dynamics (microsoft.com.• Infor (infor.com).	ERPs are usually built on a client-server architecture since they have a core module that manages the functionalities and communicates with the database.

modules contain standard, predefined functionalities that are adapted when the software is implemented.

- Epicor (epicor.com).

They are clients of the system (the users) scattered over the whole organization. These users have different devices, hardware, and channels of communication, and they all need to speak with the server.

Print server

A print server is a device that connects users of a network to a group of printers. Users send their print requests to the print server, not to the printers directly. The print server manages authentications, permissions, and job priorities.

Any network software has a print server included.

Examples:

- Microsoft Windows Homegroup.
- Linux Network Manager.
- HP Network Management Center.

Print servers usually use a client-server architecture, since they can connect different clients that have different devices or operating systems, to a group of printers that also can be equally diverse.

Mail server

A mail server is a software system that manages incoming and outgoing emails in an organization.

- IBM Lotus Domino.
- Sendmail.
- Microsoft Exchange Server.

The client is user-friendly and hides from the user the technical aspect of email.

Clients may have different devices to read mail: iPhones, Android phones, tablets, PCs, notebooks, etc.

Case-Study: Solving a Business Problem With Client-Server Architecture



Let's apply what you've learned about client-server architecture and see how it was used in a real organization.

IrisGold is a gold mining company. Here are some **quick facts**:

- IrisGold operates on three continents, with more than 21,000 employees.
- The company's mines are mostly located in remote places like the Amazonas in Brazil, the Andes mountain range, the Ural mountains in Russia, and eastern South Africa.
- The company is selecting an Enterprise Resource Planning (ERP) system package. How does this inform your decision?

What's the Business Problem?

IrisGold wants to deploy and operate its new ERP package securely. But they have two main constraints:

1. Its users are in remote places in the world with different kinds of devices (laptops, notebooks, phones, tablets).
2. Client devices must be light: they must be a simple notebook computer, tablet, or phone with few processing and storage capabilities, able to communicate to a remote server.

It's up to us to design an architecture that will support these business needs.

Got it! But where do we start?

Let's start with the facts!

First, let's look at **employee numbers and technology use**. The organization is massive and international. Big firms with lots of people in various places typically mean there are diverse practices in technology use. We can already infer from this that there are likely a diversity of platforms, operating systems, hardware, software, and communication components needed to make the administrative and production systems work. This is confirmed in the business problem: worldwide employees all use different devices.

Therefore, we need a system that will ensure that **all devices can speak with a front-end module** and that this module communicates with the ERP system. Additionally, the devices need to be **lightweight**, with few storage and processing capabilities. Hence, clients need only to be able to connect with a central server. A client-server model is a perfect solution for both situations.

Next, let's review **the nature of the ERP system**. This kind of system will need to manage business resources like cash, raw materials, human resources, production capacity, and the status of business commitments like invoices, purchase orders, and payroll. In practice, some modules of the system will only be accessed by a select group of employees at the HR department.

So how can we **manage user access**? We can ensure there is a table in a central database that specifies what menus in the system a particular user can access.

This is very typical of ERP systems: not everyone can see everything. A matrix of users/menus (who can see what) is defined when the system is implemented.

Finally, let's take another look at the **locations** where the system will be used. The logistics are key here: most of the locations are in remote, hard-to-reach places, meaning they will most likely have communication and accessibility problems. But the kind of ERP data we're working with must be secure, even with these issues.

How can we address this requirement? By locating most of the ERP system components at the IrisGold headquarters. Users can communicate to this central system through their devices, but these devices barely store any data. This also addresses the need for lightweight devices!

So to **sum up**:

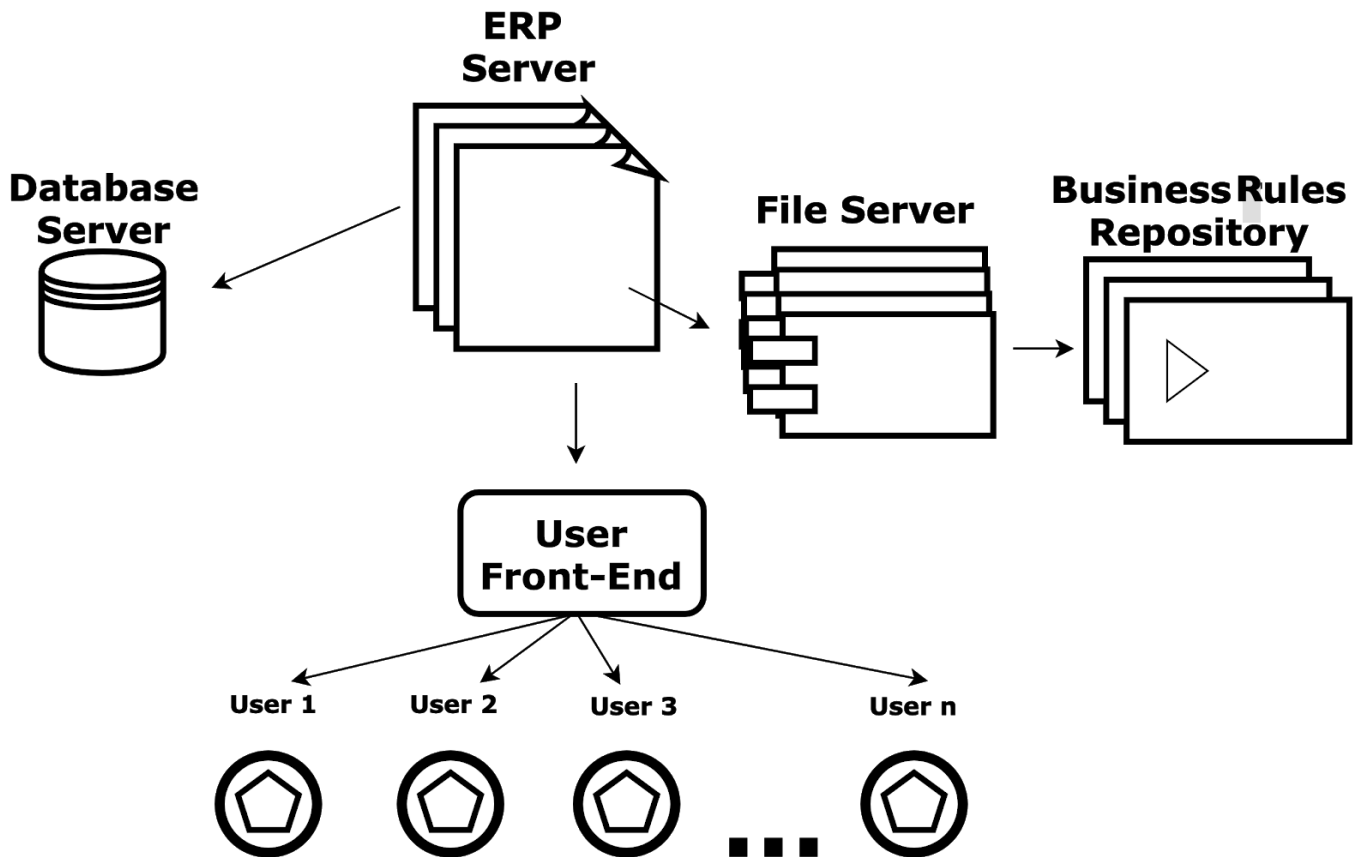
- We know that users are scattered over the world and use different devices that need to be lightweight.
- We know that users have deficient infrastructure capabilities and work in remote places.
- Clients need only to be able to connect with a central server.
- The ERP system installed in the central server must cover all the business rules. In essence, it must manage all modules for all countries internally, and it must answer client requests by communicating with a central database.

What's the Solution?

Now that we've walked through the thought process, let's see the detailed architecture diagram:

Client-Server Architecture

IrisGold



IrisGold client-server architecture

Let's explain each component of the ERP system:

- **Front-End:** This is the piece of software that interacts with ERP users, even if they are in different countries.
- **ERP server:** This is the server where the ERP software is installed.
- **File server:** The ERP server requests files from the file server to fulfill user requests. Examples: an invoice printed in PDF format, a report, a data file that the user needs. It is good practice to install a file server when the application has many users that read, update, or write files frequently.
- **Business rules repository:** A repository of business procedures, methods, and regulations for every country (all different), to make the ERP work according to each country's needs. This repository is often separated from the software to make customizations more agile and secure. It is a good practice introduced by ERPs in the '90s that has spread over many kinds of non-ERP applications.
- **Database server:** This server contains the tables, indexes, and data managed by the ERP system. Examples: customer table, provider table, invoice list, stock tables, product IDs, etc.

All **heavy processing is done at IrisGold's headquarters**. Clients are thin; that's why they are called "thin clients." They do not process or store large amounts of data. They just speak with the server.

The essence of client-server architecture is that every component shown in the diagram is located in the **central data center** (at the IrishGold headquarters). All users connect to these components through their devices.

Try it out for yourself!

Now that you've seen one solution, see if you can apply what you've learned to another!

Context: Imagine you're a software architect at a major bank. Your boss is John, the Chief Information Officer (CIO). The bank has 2,400 branches within the country, and more than 16,000 employees.

Your mission: You have been asked to develop a software system for managing the human resources aspect of the bank. This system includes modules such as:

- Employee administration: manages the creation of new employees in the system, updates existing employee data as well as those no longer working f
- or the bank.
- Performance evaluation: manages the result and analysis of semi-annual employee performance reviews. All areas in the bank hold review sessions where feedback is given to employees by their bosses, and where employees express their goals for the upcoming term.
- Career path planning: the HR Department develops a desired future career path for the employee. This path is discussed at the end of each performance review session.
- Training programs: the HR department analyzes training requirements for each area of the Bank and develops the training programs accordingly, sometimes hiring external providers.
- Payroll: the HR department is responsible for correctly paying the salaries of all bank employees. This module manages the salaries, bonuses, commissions, vacations, leave of absence, and payments of all employees' wages.

Now, it's your job to create an architecture for them! Here are some **questions you can ask yourself** to get started:

1. Computers and workstations at the bank have many different technologies, platforms, and operating systems. How are you going to solve this problem?
2. The ERP data must be secure even if users are scattered all over the country. How will you address this requirement?
3. Some modules of the system can be accessed only by a select group of employees at the HR department. How will you solve this problem?

Can you produce an architecture diagram for this business setting?

Once you've written out your version, check it against **my solution**.

Let's Recap!



Architecture Model	Description	Advantages	Disadvantages	When to use it
Client-server	A distributed application structure that partitions tasks between the providers of a service, called servers, and service requesters, called clients. Often clients and servers communicate over a computer network using different hardware.	Encapsulation of hardware, software, and functionality. Seamless combination of clients and servers on different platforms.	If all the clients simultaneously request data from the server, it may get overloaded. If the server fails for any reason, then none of the requests of the clients can be fulfilled.	When you have different user devices. When you need to encapsulate system functionalities.

[◀ GET THE MOST OUT OF THIS COURSE](#)[EVENT-DRIVEN ARCHITECTURE ▶](#)

Teacher

José Esterkin

Software engineer, project manager and trainer. Director of Positive, a project management consulting firm based in Buenos Aires.



OPPORTUNITIES

▼

SUPPORT

▼

FOR BUSINESS

▼

MORE

▼

 English



