



INCEPTIONU

ExpressJS

& Introduction to Backend
Web Development

- Client-Server Model
- How to make a Backend with Javascript
- Exercises with Express

Client-Server Model

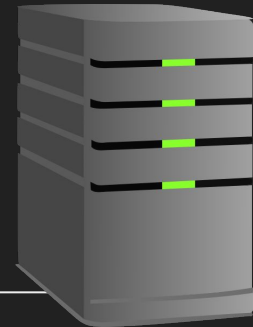
- The Client-Server Model is a software architecture concept where tasks, resources, and services are divided between two applications

Client-Server Model

- The Client-Server Model is a software architecture concept where tasks, resources, and services are **divided between two applications**:
 - **Client** - the application that **requests service**
 - **Server** - the application that **fulfills a service request**

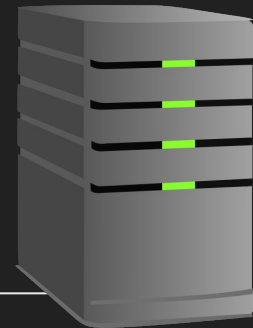
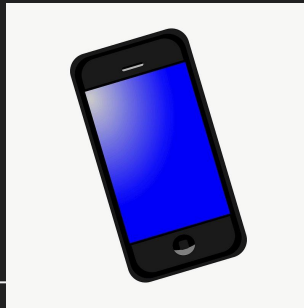
Client-Server Model

- The Client-Server Model is a software architecture concept where tasks, resources, and services are **divided between two applications**:
 - **Client** - the application that **requests service**
 - **Server** - the application that **fulfills a service request**



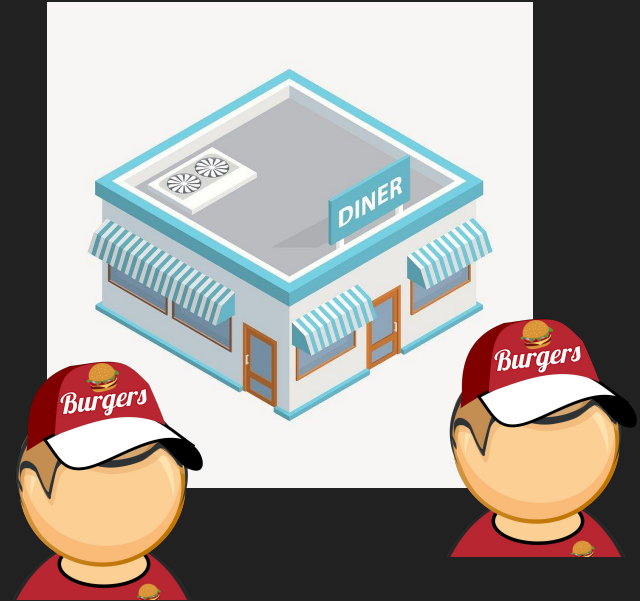
What's the point of the Client-Server distinction?

- The Client-Server Model is a software architecture concept where tasks, resources, and services are divided between two applications:
 - **Client** - the application that requests service
 - **Server** - the application that fulfills a service request



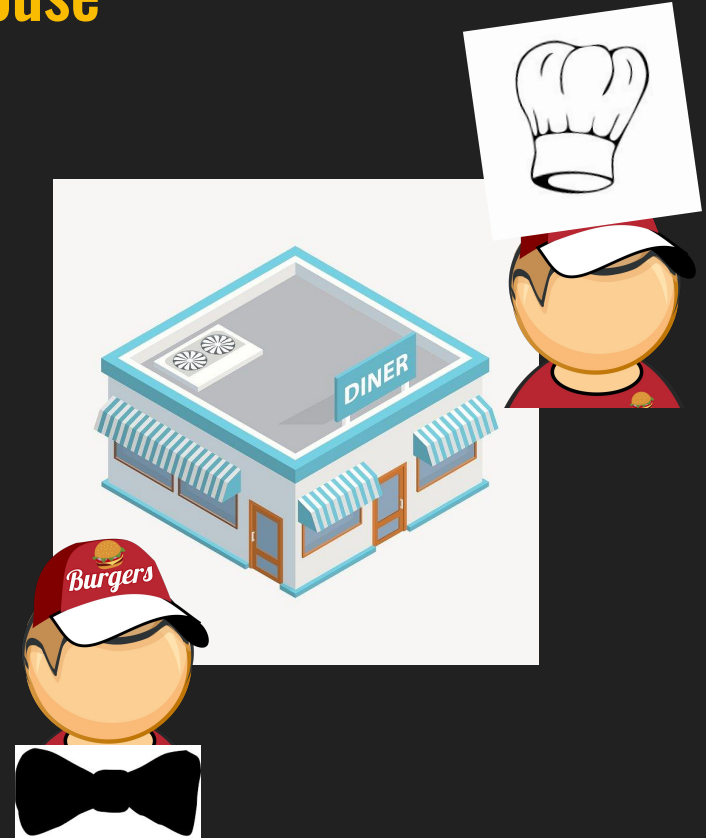
Restaurant Analogy

- Suppose a restaurant had two employees.
- Their tasks are as follows:
 - Taking orders
 - Making fries
 - Making burgers
 - Handing out orders
- Both employees are to do all of the tasks above
- What are the limitations of this model?



Restaurant: **Front of House** and **Back of House**

- Suppose a restaurant had two employees.
- Employee 1 has the following tasks:
 - Taking orders
 - Handing out orders
- Employee 2 has the following tasks:
 - Making fries
 - Making burgers
- Similarly, client and server apps can **focus on their specific tasks**.
- What are some advantages of this model?



One Back, Multiple Fronts

- If a restaurant wants to serve drive thru customers, it doesn't need a second kitchen.
- Similarly, an app may have **multiple user interfaces** (clients) but **one server**.



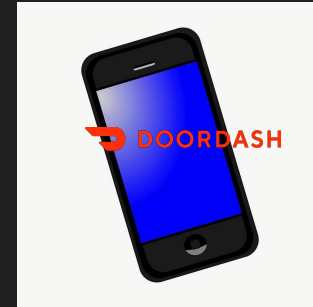
One Back, Multiple Fronts

- If a restaurant wants to serve drive thru customers, it doesn't need a second kitchen.
- Similarly, an app may have **multiple user interfaces** (clients) but **one server**.
- When might an app NEED one server but multiple clients?*



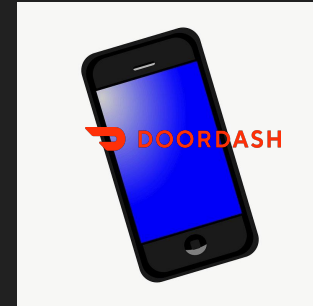
One Front, Multiple backs

- If viewed from the perspective of the app, DoorDash is one “Front End” for multiple “Back End” kitchens.



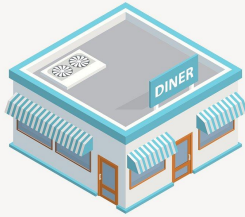
One Front, Multiple backs

- If viewed from the perspective of the app, DoorDash is one “Front End” for multiple “Back End” kitchens.



One Front, **Multi-layered backs**

- What happens if a restaurant runs out of buns?
- In a way, food suppliers and distribution another layer of back end.



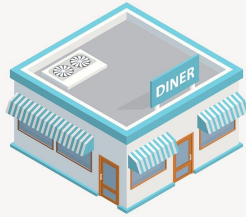
One Front, Multi-layered backs

- What happens if a restaurant runs out of buns?
- In a way, food suppliers and distribution another layer of back end.

(Client)



(Server)



(Server)

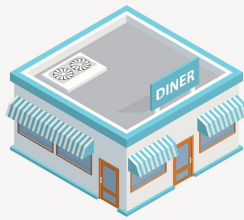


One Front, Multi-layered backs

- What happens if a restaurant runs out of buns?
- In a way, food suppliers and distribution another layer of back end.
 - In the perspective of the food distributor, the restaurant is actually a client

(Client)

(Server)



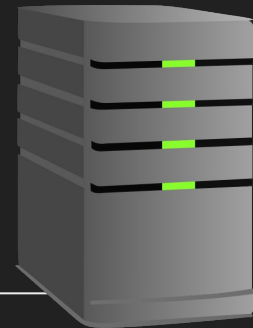
Now let's go
back to the
original
question.

What's the point of the Client-Server distinction?



What's the point of the Client-Server distinction?

- The Client-Server Model is a software architecture concept where tasks, resources, and services are **divided between two applications**:
 - **Client** - the application that **requests service**
 - **Server** - the application that **fulfills a service request**



User changes their profile picture

User



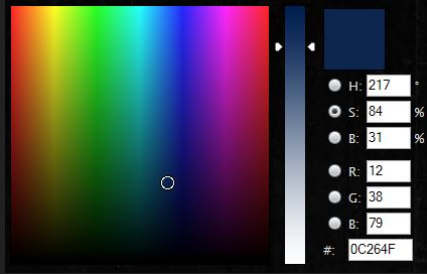
Front End



Back End



User changes their profile picture



Selects a
new color

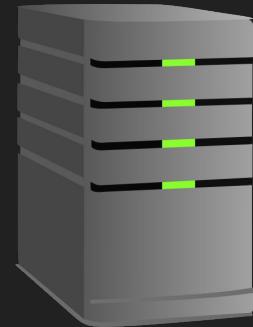
User



Front End



Back End



User changes their profile picture



Front end tells
back end about
the new color,
and to save it

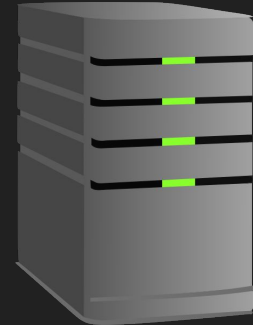
User



Front End



Back End



User changes their profile picture

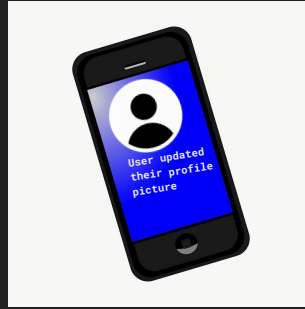


User changes their profile picture



User

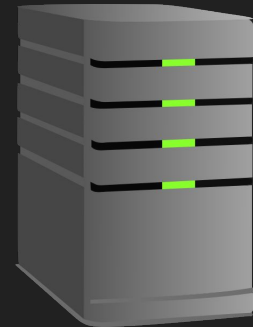
Since the change happened in the server, other users will also be able to see the change



Front End



Back End



New Terminology

User



Front End



Back End



New Terminology



New Terminology



New Terminology

- APIs are how two or more applications interact with (ie. talk to) one another.
- Just like with UIs, there are design principles, best practices, and standards to learn about with APIs.



API

(Application
Programming
Interface)






- Client-Server Model
- How to make a Backend with Javascript
- Exercises with Express

Making Back Ends (server side code) with Js is **NEW**

- Javascript used to be a **programming language only used to write client-side code**. This changed because of the popularity of NodeJS.
- Being able to use the same language for both front and back is convenient for learners.



Original author(s)	Ryan Dahl
Developer(s)	OpenJS Foundation
Initial release	May 27, 2009; 14 years ago ^[1]
Stable release	21.1.0 ^[2]  / October 24, 2023; 13 days ago
Repository	github.com/nodejs/node  
Written in	JavaScript, C++, Python
Operating system	z/OS, Linux, macOS, Microsoft Windows, SmartOS, FreeBSD, OpenBSD, IBM AIX ^[3]
Type	Runtime environment
License	MIT License ^[4] ^[5]
Website	nodejs.org 

Making Back Ends (server side code) with Js is **NEW**

- Javascript used to be a **programming language only used to write client-side code**. This changed because of the popularity of NodeJS.
- Being able to use the same language for both front and back is convenient for learners.
- However, what are some problems that can come with this?*



Original author(s)	Ryan Dahl
Developer(s)	OpenJS Foundation
Initial release	May 27, 2009; 14 years ago ^[1]
Stable release	21.1.0 ^[2]  / October 24, 2023; 13 days ago
Repository	github.com/nodejs/node  
Written in	JavaScript, C++, Python
Operating system	z/OS, Linux, macOS, Microsoft Windows, SmartOS, FreeBSD, OpenBSD, IBM AIX ^[3]
Type	Runtime environment
License	MIT License ^[4] ^[5]
Website	nodejs.org 

Javascript 10 Year Challenge

2009



2019



Can you tell which frameworks are for front, and which for back?*

2009



2019



//CodeTrace

*Me neither

Enough Blabbering, let's learn about Express...

From Wikipedia, the free encyclopedia

Express.js, or simply **Express**, is a [back end web application framework](#) for building [RESTful](#) APIs with [Node.js](#), released as [free and open-source software](#) under the [MIT License](#). It is designed for building [web applications](#) and [APIs](#).^[3] It has been called the [de facto standard](#) server framework for [Node.js](#).^[4]

- You will learn what RESTful means later
 - For now, know that [REST is a design standard](#)

- Client-Server Model
- How to make a Backend with Javascript
- Exercises with Express

Creating a new NPM project

Intro to *npm*

- NPM is a set of command-line tools that ships with “node” and it is a repository of *modules* (code built by other developers) available from npmjs.com
- To initialize a new project:
 - a. Make a new directory: **mkdir intro-to-expressjs**
 - b. Change to that new directory: **cd intro-to-expressjs**
 - c. Create a new NPM project: **npm init** (or **npm init -y**)
- Once created, start VS Code: **code .**

Creating a new NPM project

Takeaways

- Create a new project with `npm init` (or other tools as we'll see when we get to React)
- `package.json` is updated by the `npm` command or by editing the file manually (name, version, scripts, etc.)
- Add `package.json` and `package-lock.json` to your *git repository*.
- Add `node_modules` to your `.gitignore` file. This directory stores all your dependencies (often lots of files) which should not be committed to your repository.
- *Do not* manually edit `package-lock.json`.

ExpressJS - Getting started

Set Up ExpressJS

- Edit *package.json* setting **"main": "server.js"**
- Install the *express module* to your project: **npm install express**
- ExpressJS app example: start and console.log()
- ExpressJS app example: Add "Hello World!" API
- res.send() vs res.json() vs res.end()

ExpressJS - Getting started

Key Takeaways

- ExpressJS builds two handy objects for you: ***request*** and ***response*** that are passed to the “handler function” or the “callback function” or the “middleware”.
- If a route sends a response, the connection is closed and no further processing takes place.

ExpressJS - Getting started

Key Takeaways

- ExpressJS builds two handy objects for you: **request** and **response** that are passed to the “handler function” or the “callback function” or the “middleware”.
- If a route sends a response, the connection is closed and no further processing takes place.

Another way to think about it

- Middleware - a special type of function in express that contains **req** and **res**.
- **req** - contains the variables that were passed from the **client**
- **res** - the way you “return” to the **client**.

Interacting with the Server using curl

- So far, we've been using the browser as our interface to access the Hello World app. But there are other tools we need to use for more specialized interactions.
- Try running: `curl --version` (try on Git Bash if on windows)
- Now, run: `curl "http://localhost:4000/send/html"`

Interacting with the Server using curl

- So far, we've been using the browser as our interface to access the Hello World app. But there are other tools we need to use for more specialized interactions.
- Try running: `curl --version` (try on Git Bash if on windows)
- Now, run: `curl "http://localhost:4000/send/html"`
- Another tool used to make API calls is [Postman](#).
- Learn more about curl: `curl --man`

Handling POST requests

Code Example

- One thing easy to do with curl but difficult using the browser is [handling POST Requests](#)
- Send data via POST with: `curl -d @<filename> <URL>`
- POST and GET are **HTTP verbs** or methods. This will be covered more in future lectures.

Handling POST requests

Key Takeaways

- The `express.urlencoded()` middleware will look for form submission headers and add the data to `request.body`.
 - a. This assumes data was submitted using a traditional submit button
 - b. Or via a POST request sent using an application such as Postman
 - c. NOT using javascript/json.
- `express.json()` is the JSON version of this middleware
 - a. It performs the same functionality and adds submitted data to `request.body`

Exercise - Create a GET handlers in ExpressJS

Instructions

Extend the *hello world* ExpressJS app

- Mild: Add a new GET endpoints that returns the plain text: **"ExpressJS Rulez!!!"**
- Medium: Add a new GET endpoint that returns JSON: **{"hello": "world"}**
- Spicy: Add GET endpoints that returns HTML including the server's current Date inside HTML tags of:
<html><body><h1>THE_CURRENT_TIME_GOES_HERE</h1></body></html>

What we covered

- The **Client-Server Model**
- What is an **API**
- How to make a **server-side application** using Javascript, npm, nodeJS, and Express
- The **request** and **response** objects in Express
- Using **curl** to make HTTP requests
- Introduction to **HTTP verbs: GET and POST**
- **Different response types**: plain text, json, and HTML