Client-Server Architecture: Fullstack Applications

The purpose of this lecture is to be a "closure" of the back-end part of the curriculum.

It should pave the way to to FullStack applications, reinforcing that back-end and front-end are two SEPARATE processes that doesn't share memory nor any resources and only communicate through http.

Machine 1 (computer + operating system) Machine 2 (computer + operating system)

Filesystem (serial data)

Node.js process
(C/C++ runs V8, libev, libeio)

Chrome process
(C/C++ runs V8, DOM, etc.)

HTTP

static files
(images, .html docs,
.js files, etc.)

10111000111010110
01001011010010100
10101010101101010
0110100101011

raw database data
(e.g. BSON, .sql)

10111000111010110
010010110100

FS
I/O

JavaScript (being executed by **V8**) JS

www.mycoolapp.com/somepath

Address bar (makes GET requests)

Express.js (middleware framework)

morgan (logger middleware)

next();

express.static (middleware)

next();

body-parser (middleware)

next();

express.Router (middleware)

next(err);

error handling middleware

DOM (node tree, what gets rendered)

html

body

h1

button

JavaScript (executed by **V8**) JS

FS I/O

FS
I/O

DBMS:
mongod / postgres
process (C/C++)

translate query into binary
read/write and send back
result to requester

Mongoose (ODM) / **Sequelize** (ORM)

schema

model → instances

mongodb / **pg** (database driver)

Db Protocol   Db Protocol

mongo / psql process

> shell
(in terminal)

$ console/shell (in terminal)

> console (in dev tools)