Deconstructing Full-stack Web Applications

Break it apart so you can build it up

E190BO · AI-Amplified Prototyping



Hyperschedule

PAUSE THE VIDEO

How many different computers had to cooperate to make that happen?

Pause and write down a number.

Outline

1. **A few preliminaries:** the basic architecture, Hypertext Transfer Protocol (HTTP), and some command line tools
2. **Deconstructing Hyperschedule:** from the pixels on the screen on down
3. **Building your own:** how we'll build one ourselves

A few preliminaries

Two machines

The client

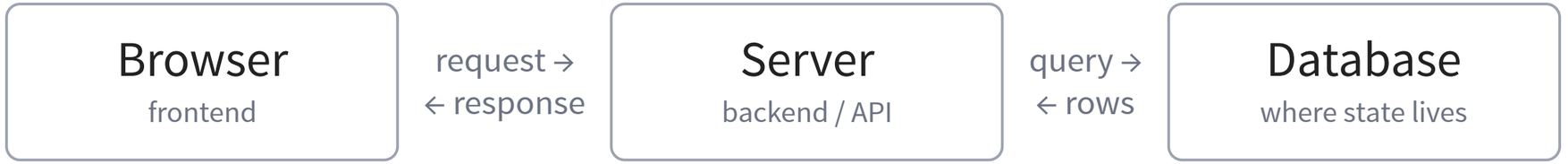
- Your browser, on **your** laptop
- Runs HTML, CSS, and JavaScript that it *downloaded*
- Anyone can read this code

The server

- Someone else's computer, somewhere else
- Runs code you never see
- Holds the data

The question we'll return to throughout this video: *which machine runs this code?*

The basic architecture



- The **frontend** is what you see and click. It runs in the browser.
- The **backend** takes requests and decides what to do with them.
- The **database** remembers things after everyone closes their laptop.

Every web app has these three boxes. Now let’s find them in a real one.

Anatomy of a URL

`https://` `banana.hyperschedule.io` `/v4/sections/FA2026`

Part	Example	Answers the question
scheme	`https://`	how do we talk? (encrypted HTTP)
host	`banana.hyperschedule.io`	*which machine?* DNS turns this name into an address
path	`/v4/sections/FA2026`	*what do you want* from that machine?

HTTP: a request and a response

Request (you → server)

```
GET /v4/term/current HTTP/1.1
Host: banana.hyperschedule.io
User-Agent: curl/8.17.0
Accept: */*
```

Response (server → you)

```
HTTP/1.1 200 OK
Content-Type: application/json
Server: cloudflare
cache-control: public,max-age=600

{"year":2026,"term":"FA"}
```

A **verb** and a **path**, then **headers**, a blank line, and a **body**. It's just text.

Verbs and status codes

Verb	Means
`GET`	read
`POST`	create
`PUT` / `PATCH`	update
`DELETE`	delete

Code	Means	Whose fault
2xx	it worked	nobody's
3xx	look over there	nobody's
4xx	bad request	**yours** (the client)
5xx	server broke	**theirs** (the server)

The ones you'll meet most often: `200` OK, `400` malformed, `401` who are you?, `403` I know who you are, and no, `404` no such thing, `500` the server crashed.

Tool 1: Browser devtools allow you to see how the view is rendered

Open with ⌥⌘**I** (Mac) or **Ctrl+Shift+I** / **F12** (Windows/Linux).

Tab	Shows you	We'll use it for
Elements	the page as a tree	Layer 1
Network	every request the page makes	Layers 2, 3, 5
Application	what's stored in your browser	Layer 2
Console	errors, and a JS prompt	when something breaks

Safari: first turn on *Settings → Advanced → Show features for web developers*.

Tool 2: `curl` allows you to get data over HTTP from the command line

A browser without the browser: send one request and print what comes back.

Command	Does
`curl URL`	print the response body
`curl -s URL`	…silently (no progress bar)
`curl -i URL`	…with the response headers
`curl -I URL`	headers only
`curl -v URL`	the whole conversation, both directions
`curl -X POST URL`	use a different verb

Windows PowerShell: type `curl.exe`, not `curl`. Or skip the problem entirely and use your Codespace (two slides on).

Tool 3: `jq` allows you to view structured JSON data

JSON is the shared language between frontend and backend: objects `{}`, lists `[]`, strings, numbers, `true`/`false`, `null`.

`jq` pretty-prints JSON and pulls pieces out of it:

```
curl -s $B/v4/sections | jq length                # how many items?
curl -s $B/v4/sections | jq '.[0]'                # the first one
curl -s $B/v4/sections | jq '.[0].course.title' # one field of the first one
```

`$B` is shorthand for `https://banana.hyperschedule.io` (run `export B=https://banana.hyperschedule.io`).

You'll use your Codespace to run these commands

- A **Linux computer in the cloud**, accessed in your browser or from VS Code
- Built from a recipe in your repo (`.devcontainer/`), so **everyone gets an identical machine**
- `curl`, `jq`, `git`, Python 3.13, and Claude Code are already installed
- Opening one: your repo → **Code** → **Codespaces** → **Create codespace**

Deconstructing Hyperschedule

The map

The pixels

Logic in your browser

The network

The API server

The database

Below the bottom

Around it all

We’ll start where you touch it and dig until we hit the bottom.

Layer 1 · The pixels

The pixels React components + CSS frontend

Logic in your browser

The network

The API server

The database

Below the bottom

Around it all

▶ LIVE Inspect the page

1. Open hyperschedule.io and open devtools → **Elements**.
2. Hover over elements in the tree and watch the matching region light up.
3. Right-click one course result → **Inspect**. It's just nested `<div>`s.
4. Point out that each region is a *component*, one file each:

```
App
├─ CourseSearch        course-search/CourseSearch.tsx
│  └─ CourseRow × N    course-search/CourseRow.tsx
├─ Schedule            schedule/Schedule.tsx
└─ Sidebar             Sidebar.tsx → SelectedList, MiniMap
```

github.com/hyperschedule/hyperschedule/tree/d41a5e5/frontend/src/components

Layer 2 · Logic in your browser

The pixels	React components + CSS	frontend
Logic in your browser	search runs here · guest schedule in localStorage	frontend
The network		
The API server		
The database		
Below the bottom		

Around it all

▶ LIVE Watch the network while you search

1. Devtools → **Network**. Clear it (🚫).
2. Type `CSCI 81` in the search box, one letter at a time.
3. Point out: **no new requests.** Not one per keystroke.

PAUSE THE VIDEO

Where is the search happening?

Pause and guess.

Search is happening on your laptop

- On page load, the app downloaded **every section for the term, once**: **2,187 sections, 2.4 MB of JSON**. (Reload with Network open and find `sections`.)
- Every keystroke searches that list **in your browser**.

```
export function matchesText(text: string, section: APIv4.Section, ...): number | null {
    if (text === "") return 1; // everything matches with the same score
    const tokens = Array.from(text.toLocaleLowerCase().matchAll(tokensRegex))...
```

frontend/src/lib/search.ts:13

That's why search feels instant. It's also a **choice**, and it would be a bad one on a phone over cellular, with 50,000 items, or with private data.

▶ LIVE Where does a guest schedule live?

1. Devtools → **Application** → **Local Storage** → `https://hyperschedule.io`.
2. Expand the entry and find your schedule, with `CSCI 81` in it.
3. Point out: **no server involved.** This is stored only in this browser.
4. (Optional) Clear it and reload. The schedule is gone.

frontend/src/hooks/store/user.ts:312

Layer 3 · The network

The pixels	React components + CSS	frontend
Logic in your browser	search runs here · guest schedule in localStorage	frontend
The network	hyperschedule.io → GitHub Pages · banana.hyperschedule.io → Cloudflare	network
The API server		
The database		
Below the bottom		

Around it all

▶ LIVE The frontend and backend are served from different machines

```
curl -sI https://hyperschedule.io | grep -i server     # server: GitHub.com
curl -sI $B/v4/term/current       | grep -i server     # server: cloudflare
```

- **`hyperschedule.io`** hosts the app's *files* (HTML, JS, CSS). They're static and served by GitHub Pages.
- **`banana.hyperschedule.io`** is the app's *data*, which is live and sits behind Cloudflare.

The page you downloaded in Layer 1 comes from one place. The data it fetches comes from another.

▶ LIVE You can talk to the API yourself using `curl`

```
export B=https://banana.hyperschedule.io
curl -v --http1.1 $B/v4/term/current       # the raw conversation, from the primer
curl -s $B/v4/sections | jq length         # 2187
curl -s $B/v4/sections | jq '.[0].course.title, .[0].seatsFilled, .[0].seatsTotal'
curl -si $B/v4/sections/FA2026 | head -1   # 200 here you go
curl -si $B/v4/sections/FA2099 | head -1   # 404 no such thing
```

The browser made exactly these requests. It just drew pictures with the answers.

Caching is configured in the header

```
curl -sI $B/v4/sections | grep -i cache-control
```

```
cache-control: public,immutable,s-max-age=15,max-age=30,stale-while-revalidate=60,stale-if-error=86400
```

- `max-age=30`: your browser can reuse this answer for **30 seconds**. That's why a seat count can be slightly stale.
- `stale-if-error=86400`: if the server is down, keep serving the old answer for **a day**. Hyperschedule survives an outage.

Layer 4 · The API server

The pixels	React components + CSS	frontend
Logic in your browser	search runs here · guest schedule in localStorage	frontend
The network	hyperschedule.io → GitHub Pages · banana.hyperschedule.io → Cloudflare	network
The API server	GET /v4/sections · POST /v4/user/section	backend
The database		
Below the bottom		

Around it all

The server is what responds to our requests

```
// backend/src/server.ts
const app = new App()
    .use("/v4", v4App)
    .use("/v3", v3App)
    .use("/auth", authApp);

// backend/src/index.ts
await connectToDb(DB_URL);
app.listen(8080);
```

server.ts

```
courseApp.get("/sections", async (req, reply) => {
    const sections = await getAllSections();
    return reply
        .header("Cache-Control", "…max-age=30…")
        .send(JSON.stringify(sections));
});
```

routes/v4/courses.ts:10 (trimmed)

- **A server is a process that never exits.** `listen(8080)` waits forever.
- **Routing** maps (verb, path) to a function. That function answered your `curl $B/v4/sections`, and it set the cache header you just read.

Layer 5 · The database

The pixels	React components + CSS	frontend
Logic in your browser	search runs here · guest schedule in localStorage	frontend
The network	hyperschedule.io → GitHub Pages · banana.hyperschedule.io → Cloudflare	network
The API server	GET /v4/sections · POST /v4/user/section	backend
The database	MongoDB: `sections`, `users`	backend
Below the bottom		

Around it all

The Network tab enables us to trace what a click does

1. **Log in** (already done before recording), then open devtools → **Network**.
2. Add a course and find the `POST` to `/v4/user/section`.
3. Show its **Payload** tab: the JSON the browser sent.

Hop	Code
Browser sends `POST /v4/user/section`	frontend/src/lib/api.ts:33
Route checks the cookie (401) and the body (400)	backend/src/routes/v4/user.ts:143
Model writes to MongoDB	backend/src/db/models/user.ts:196

Writing data stores documents in MongoDB

A user is **one document**:

```
{ _id: "u~…", school: "HMC",
  schedules: {
    "s~…": {
      name: "Fall plan",
      term: { year: 2026, term: "FA" },
      sections: [
        { section: { department: "CSCI",
                     courseNumber: 81, … },
          attrs: { selected: true } } ] } } }
```

Adding a course pushes onto that nested list:

```
collections.users.findOneAndUpdate(
  { _id: userId, ... },
  { $addToSet: {
      [`schedules.${scheduleId}.sections`]:
        { attrs: { selected: true }, section },
  } },
);
```

`$addToSet` means “add it unless it’s already there.”

MongoDB stores **documents**, not tables. Most apps, including the ones we’ll build, use SQL tables. The job is the same; the shape is different.

PAUSE THE VIDEO

A guest’s schedule lives in your browser. A signed-in user’s lives in MongoDB.

Which one survives switching laptops? Why?

Layers 6 & 7 · What the browser can’t see

The pixels	React components + CSS	frontend
Logic in your browser	search runs here · guest schedule in localStorage	frontend
The network	hyperschedule.io → GitHub Pages · banana.hyperschedule.io → Cloudflare	network
The API server	GET /v4/sections · POST /v4/user/section	backend
The database	MongoDB: `sections`, `users`	backend
Below the bottom	the registrar, polled every 60 s	data source

Around it all

Docker · .env · CI/CD

ops

The data comes from a database. The app is built using Docker

Where the data comes from

- Hyperschedule doesn't own it. A **fetcher** polls the colleges' registrar every **60 s** for seat counts and rewrites `sections`.
- The registrar URLs are deliberately kept out of the repo:

> *"We promised the school to not disclose how exactly we are contacting their server."*

hmc-api/fetcher/endpoints.ts

The scaffolding

- `docker-compose.yml`: four services (frontend, backend, db, nginx). **The whole diagram in one file.**
- Same code, different machine:
  - `.env.development` → `http://localhost:8080`
  - `.env.production` → `https://banana.hyperschedule.io`
- Every push is checked automatically, and merging deploys the site. **No human uploads files.**

Building from the bottom up

The whole thing

The pixels	React components + CSS	frontend
Logic in your browser	search runs here · guest schedule in localStorage	frontend
The network	hyperschedule.io → GitHub Pages · banana.hyperschedule.io → Cloudflare	network
The API server	GET /v4/sections · POST /v4/user/section	backend
The database	MongoDB: `sections`, `users`	backend
Below the bottom	the registrar, polled every 60 s	data source

Around it all

Docker · .env · CI/CD

ops

Your number from the start? **At least seven computers:** your laptop, a DNS resolver, GitHub's CDN, Cloudflare, Hyperschedule's server, MongoDB, and the registrar. An eighth, GitHub Actions, built the page you downloaded.

Read top-down. Build bottom-up.

We just *read* an app from the pixels down. We'll *build* one from the bottom up:

- A page has nothing to show until an API exists.
- An API has nothing to serve until data exists.
- And we can test each layer before the next one exists. We just talked to an API with `curl` and no UI at all.

Same boxes, our tools

Box	Hyperschedule	What we'll use
Database	MongoDB	SQLite → Postgres
API server	TypeScript + tinyhttp	Python + FastAPI
Frontend	React + Vite	plain HTML/CSS/JS
Hosting / CDN / env	GitHub Pages, Cloudflare, `.env.*`	Railway
How you build it fast	a team of students, over years	you + an agent
A new kind of client	—	an MCP server over your API

What we'll do in class

- Before class, set up your **Codespace**.
- **Part 1:** redo this video yourself, from the command line. It ends with making Hyperschedule's API return **200, 400, 401, and 404**.
- **Part 2:** take apart **Wordle** using the same strategy.