Intro to Backend Development

Lecture 1 · Routes

Anton Matchev
Aayush Agnihotri

Logistics

Enrollment

- This is CS 1998 Section 603, a 2-credit S/U course
- If you have passed the **pretest AND filled out the Google form**:
  - Enroll directly on Student Center - course number is **11847**
  - Make sure you are on our Ed Discussion – if not, email courses@cornellappdev.com

Structure

- Lectures on M/W 7:30 - 8:20pm in Olin 255
- We rotate between lecture days and demo days
- At the end of every class, we will have a Kahoot to record attendance
- Recordings of lectures will be uploaded to our YouTube channel

Assignments

- Weekly assignments
  - Released on Mondays
  - Due the following Monday (except for this week!) at 11:59pm
  - **6** total slip days
  - Can only use up to **3** slip days on a single assignment
  - -1 penalty per day after slip days run out

Grading

Item	Weight
PA1 - Reddit	10
PA2 - Venmo (Basic)	10
PA3 - Venmo (Full)	10
PA4 - CMS	10
PA5 - Dockerize CMS	10
PA6 - Deploy CMS	10
Final Project	30
Weekly Surveys	5
Attendance	5
Total	**100**
Passing Score	**70**

Attendance is mandatory, but...

Thank you Shungo and Kate #201

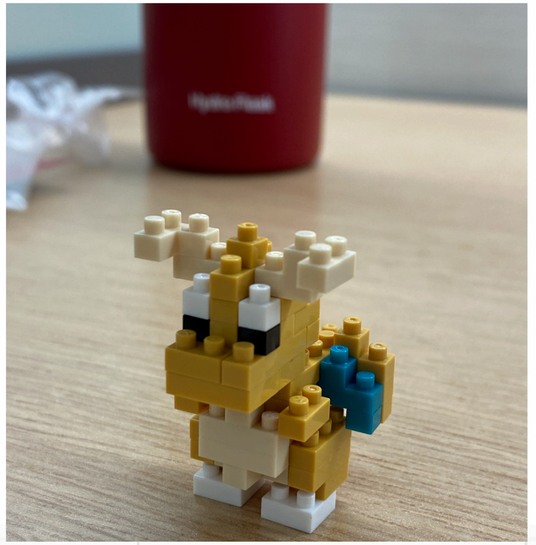
James Kim
5 months ago in General

193
VIEWS

9

Thank you Shungo and Kate.

It is beautiful.



Textbook

- Found at **https://backend-course.cornellappdev.com/**
- SUPER Useful!
  - Syllabus
  - Assignment information

Academic Dishonesty

- ZERO tolerance
- Follow Cornell's Academic Integrity guidelines
  - Cite what you use if it's not yours
  - Talking amongst yourselves is fine, copying code is not
  - If we find out about an AI violation, we will automatically drop you

Academic Dishonesty

- Our faculty advisor:
  Walker White

Course Overview

1. Routes
2. Databases
3. Relational Databases
4. Abstractions

Course Overview

1. Routes
2. Databases
3. Relational Databases
4. Abstractions
5. Containerization & DevOps
6. Deployment & Services

Course Overview

1. Routes
2. Databases
3. Relational Databases
4. Abstractions
5. Containerization & DevOps
6. Deployment & Services
7. Hack Challenge
8. Images

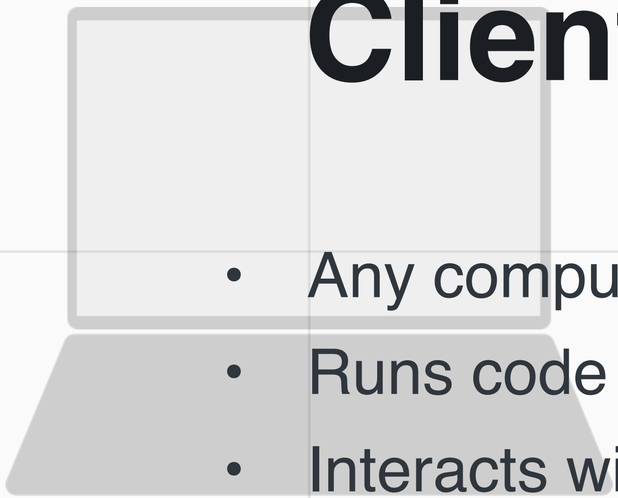
Instructors

Tony Matchev
c/o 2026

Aayush Agnihotri
c/o 2026

Let’s Begin!

Client-Server Model

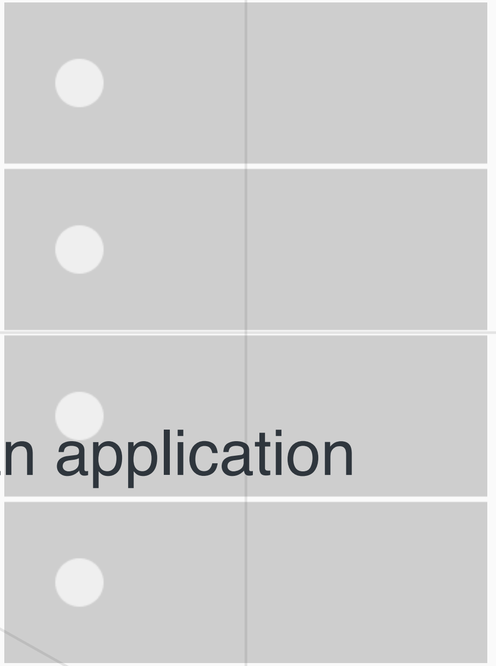


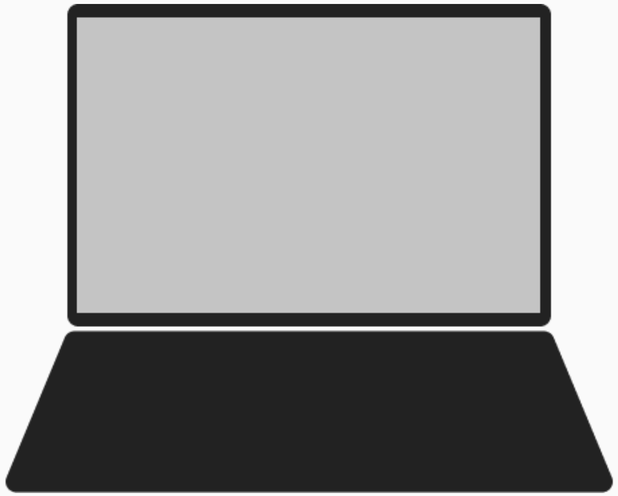
Client

- Any computer (i.e. phone, laptop, tablet, etc.)
- Runs code locally
- Interacts with the **frontend** of an application
  - Display content
  - Handle user interactions (i.e. clicking a button)

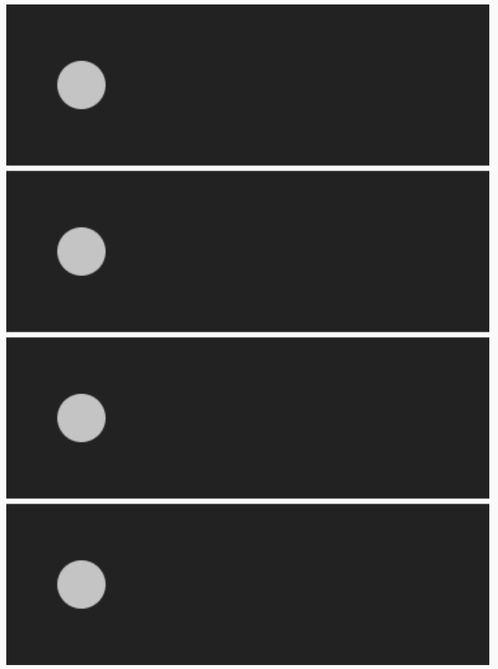
Server

- Also just a computer
- Runs **backend** code that handles the data of an application
- Centralizes access to information
- Communicates with clients





Client



Server

Client-Server Communication

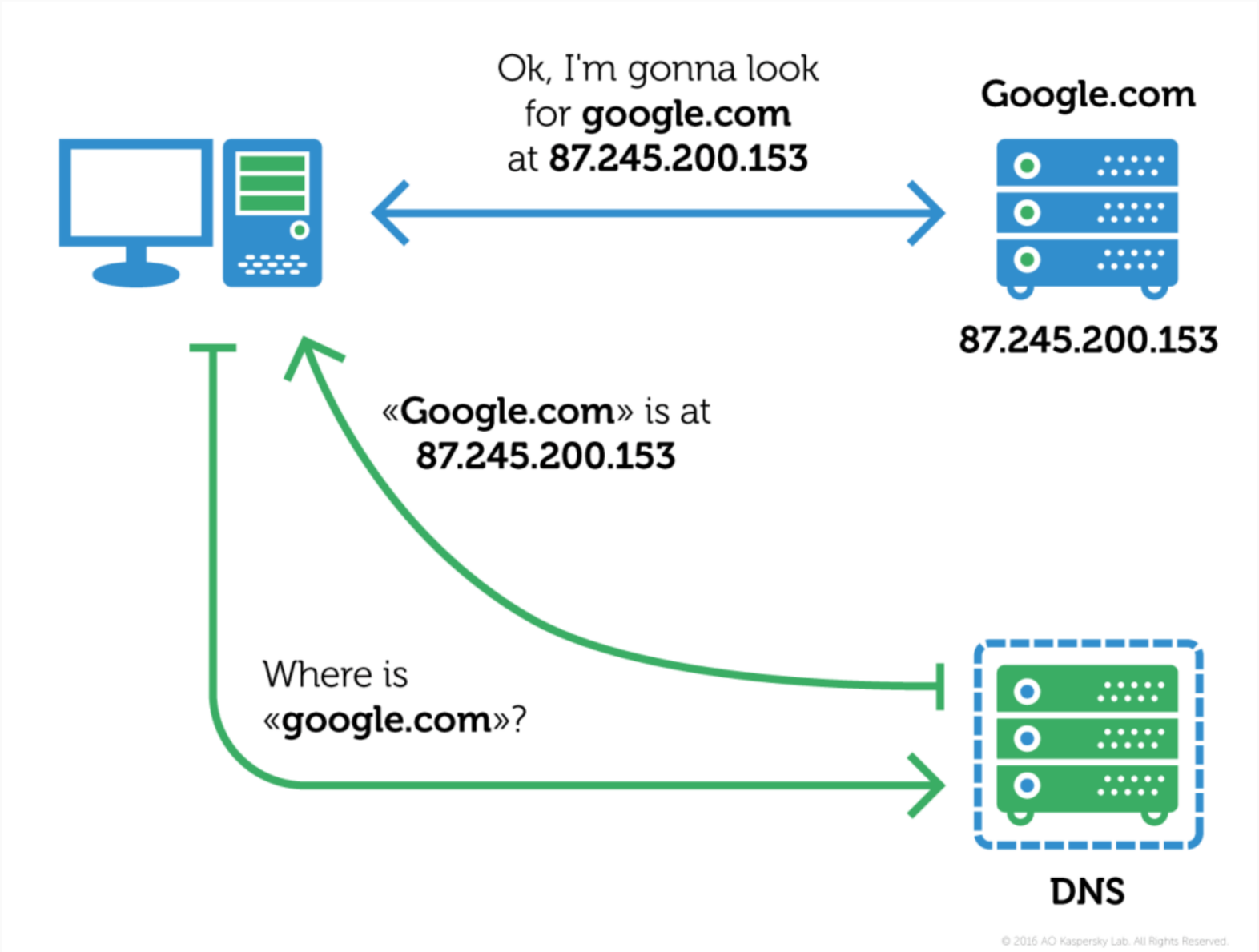
http://www.google.com

http://www.google.com

- HyperText Transfer Protocol
- Defines message formatting and transmission standards
- Used everywhere

http://www.google.com

- www.google.com is a **domain name**
- Domain != server
- Domain names need to be translated into an IP address


Ok, I'm gonna look
for google.com
at 87.245.200.153
Google.com
87.245.200.153
«Google.com» is at
87.245.200.153
Where is
«google.com»?
DNS
© 2016 AO Kaspersky Lab. All Rights Reserved.

Request

- Initiated by **client**
- Network call over the internet
- **URL** to indicate request destination
- **Methods** to indicate operation purpose
- **Body** that contains additional data from client

Request URL

- Indicates request destination (also called “endpoint”)
- Defines a specific **route path** to hit (think of routes like functions)
- Can integrate data into the URL
  - Information encoded within the path
  - Query parameters

Request Methods

- 4 main archetypes of operations:
  - **C**reate
  - **R**etrieve
  - **U**pdate
  - **D**elete

Request Methods

POST = information transmission

GET = information retrieval

PUT = information revision

DELETE = information removal

- And many more!
- REST(ful) APIs typically conforms to these methods

Request Body

- Standardized formats (i.e. plain text, JSON, HTML, etc.)
- Contains information necessary for desired operation
  - Username & password for login
  - Image content for media upload
- Not the only piece of data attached to request, but most useful

Example Requests

GET http://www.google.com/search?q=query

GET http://www.google.com/search?q=query

- GET request

GET http://www.google.com/search?q=query

- GET request
- Google domain

GET http://www.google.com/search?q=query

- GET request
- Google domain
- /search route path

GET http://www.google.com/search?q=query

- GET request
- Google domain
- /search route path
- Query parameters
  - “key1=value1&key2=value2…"

POST http://www.google.com/login

```
{“username”: “js123”, “password”: “password”}
```

POST http://www.google.com/login

```
{“username”: “js123”, “password”: “password”}
```

- POST request

POST http://www.google.com/login

```
{“username”: “js123”, “password”: “password”}
```

- POST request
- /login route path

POST http://www.google.com/login

```
{“username”: “js123”, “password”: “password”}
```

- POST request
- /login route path
- Request body in JSON format (must be double-quotes!)

Response

- Triggered by client **request**
- Returns confirmation of requested operation
- **Response codes** used to indicate success at abstract level
- Sends meaningful data in a **body** just like a request

Response Codes

- Indicates status of requested resource
- Sent accompanying the response body
- 200 -> OK
- 404 -> Not Found
- 500 -> Internal Server Error

Example Responses

GET http://www.google.com/search?q=query

200

Status Code

```
{
  “count”: 100,
  “result”: [{…},…]
}
```

Body

POST http://www.google.com/login

200

Status Code

```
{
  “success”: true,
  “data”: {…}
}
```

Body

POST http://www.google.com/login

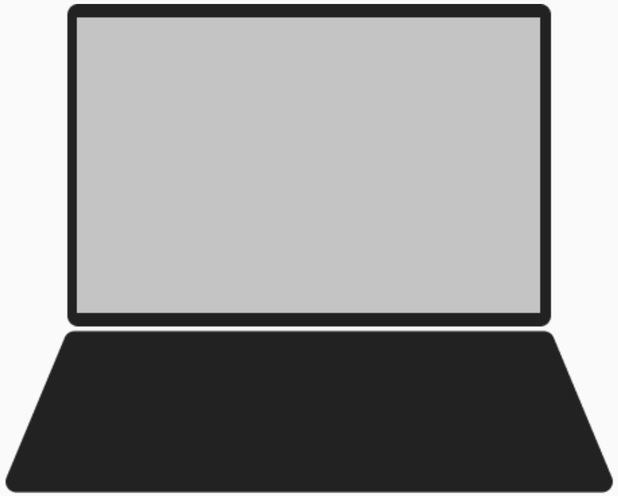
401

Status Code

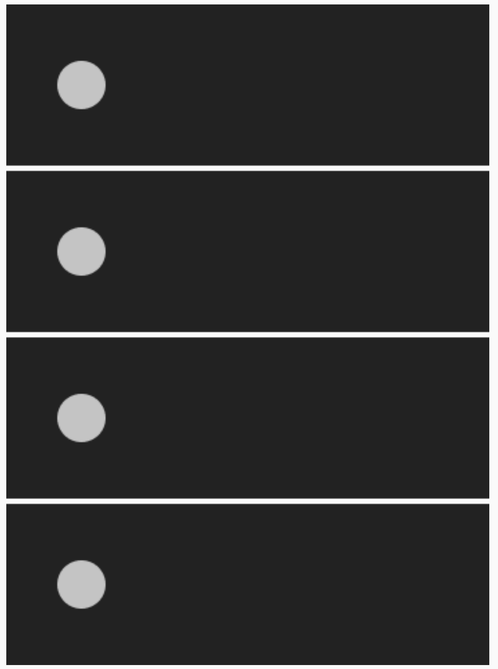
```
{
  “success”: false,
  “error”: “…”
}
```

Body

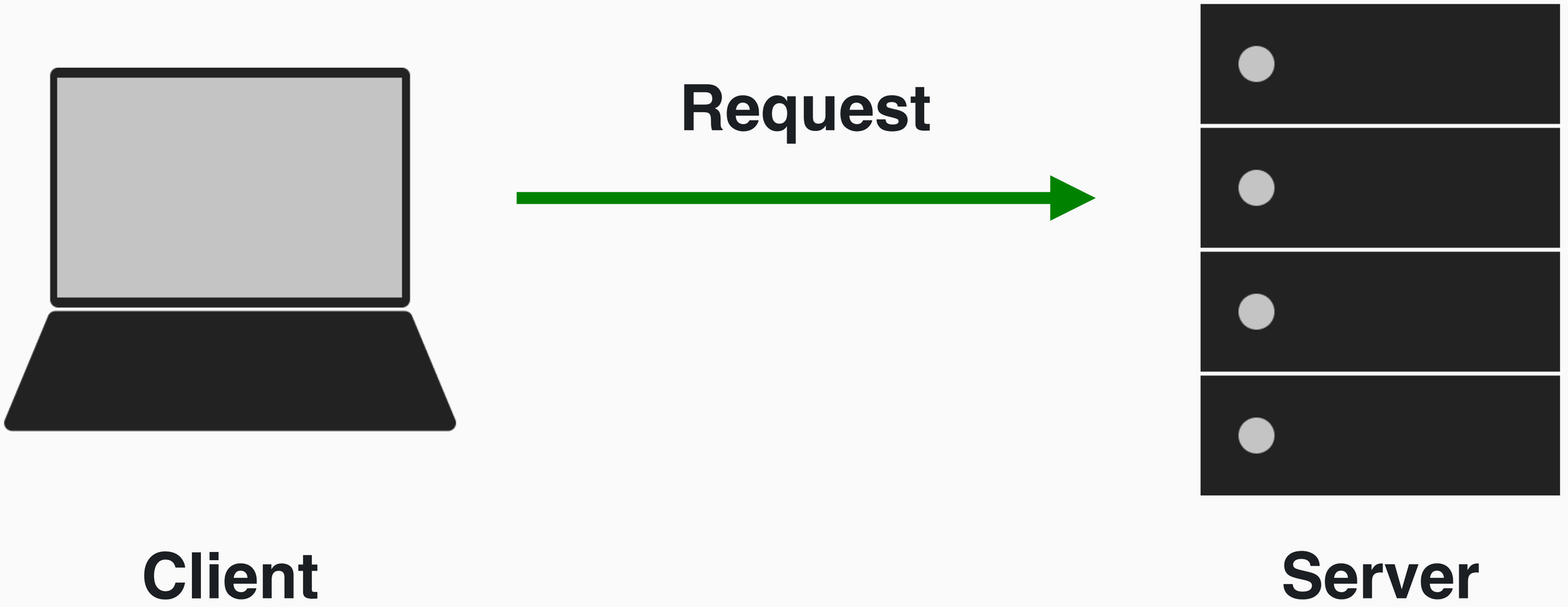
Summary



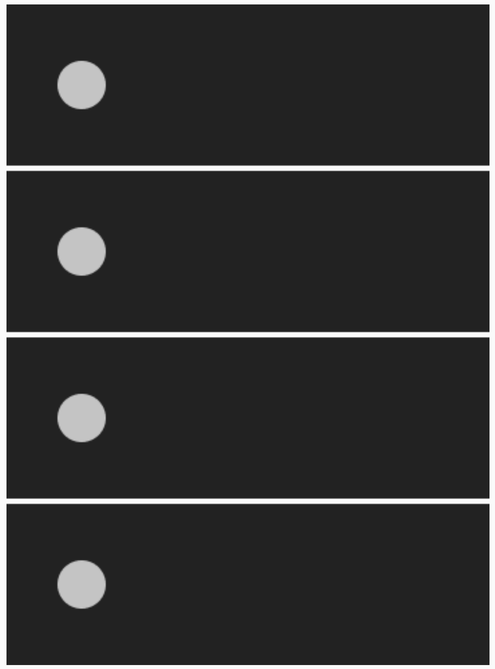
Client



Server


Request
Client
Server

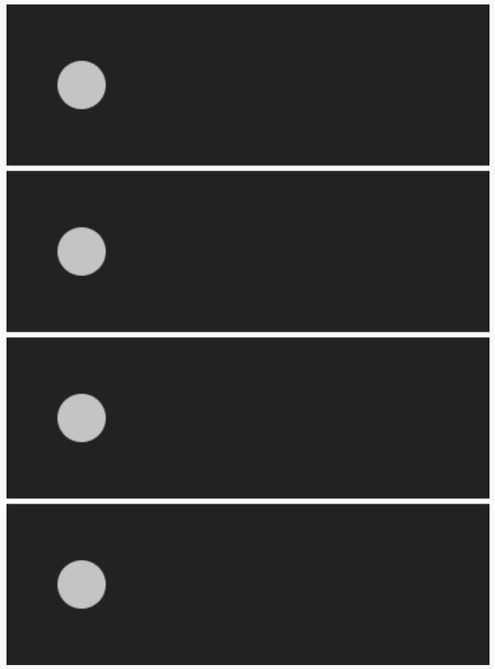
Request



Server

- Route triggered
- Data operation
- Return response

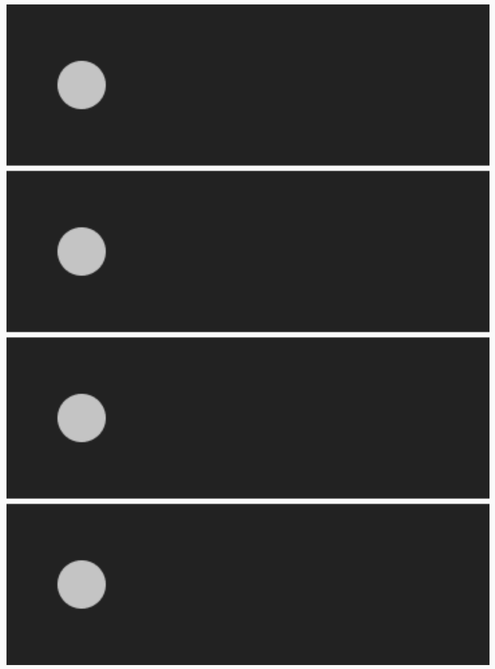
Request



Server

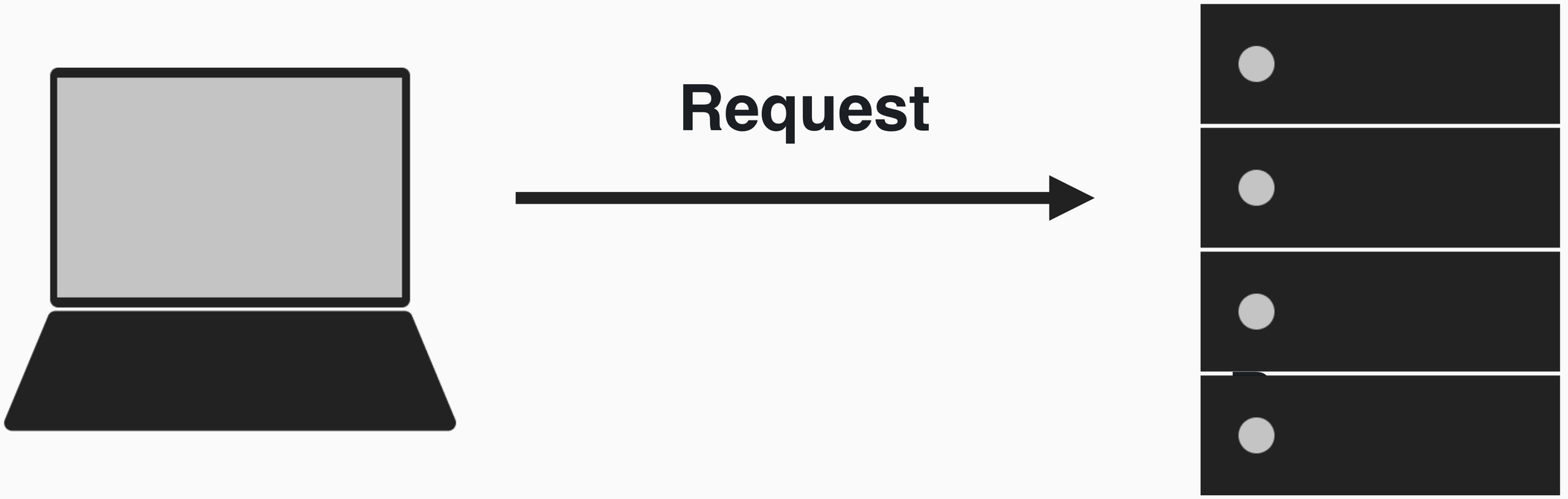
- Route triggered
- Data operation
- Return response

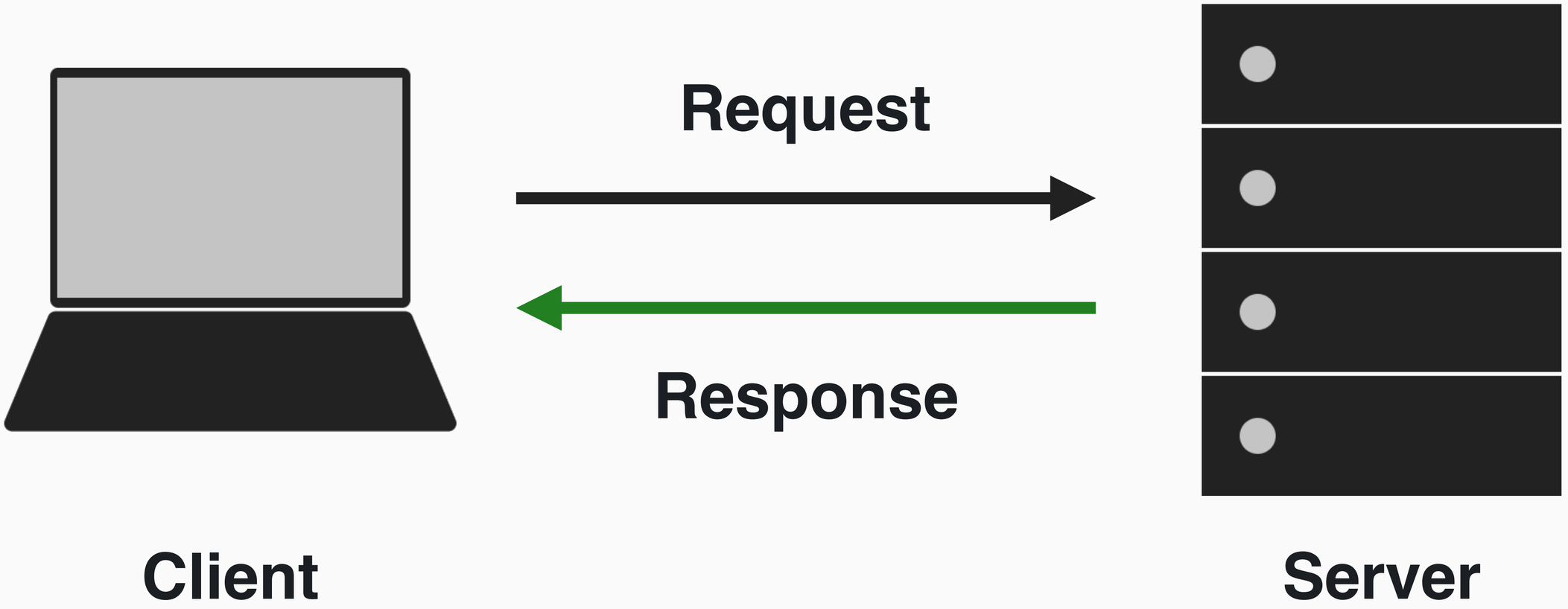
Request



Server

Route triggered
Data operation
Return response


Request
Client
Server


Request
Response
Client
Server

Questions?

Pre-Demo Knowledge Check

Documentation

- Documentation is super important for you and others who use your code to be able to understand its functionalities

Documentation

- Documentation is super important for you and others who use your code to be able to understand its functionalities

```
def get_task(task_id):

    # returns the task with the id `task_id`

    ...
```

Documentation

- That being said, avoid commenting code where self-explanatory or not necessary

Documentation

- That being said, avoid commenting code where self-explanatory or not necessary

```
const items = [...];

const data_loaded = false; // checks if data has been loaded

const user = {...};
```

Documentation

Debugging

- The process of removing errors from our code
- We will be using Postman to debug our backend code and will demonstrate shortly!

Virtual Environments

- Creates an isolated environment for Python projects
- Virtual environments contain packages of specified versions
- Useful when working on multiple projects
  - Say one project requires Flask 2.0.0 and another requires Flask 1.0.2
- Don’t forget to activate/deactivate!!

Action Items

- Join Ed Discussion by SATURDAY 11:59 pm – link on CMS
- If you are not added to CMS by the end of the weekend **(FILL OUT GOOGLE FORM)**, email **cornellappdevcourses@gmail.com**
- Good luck, y’all got this!

Demo