Building & Testing REST APIs

CSCC01 • Summer 2024

Topics Covered

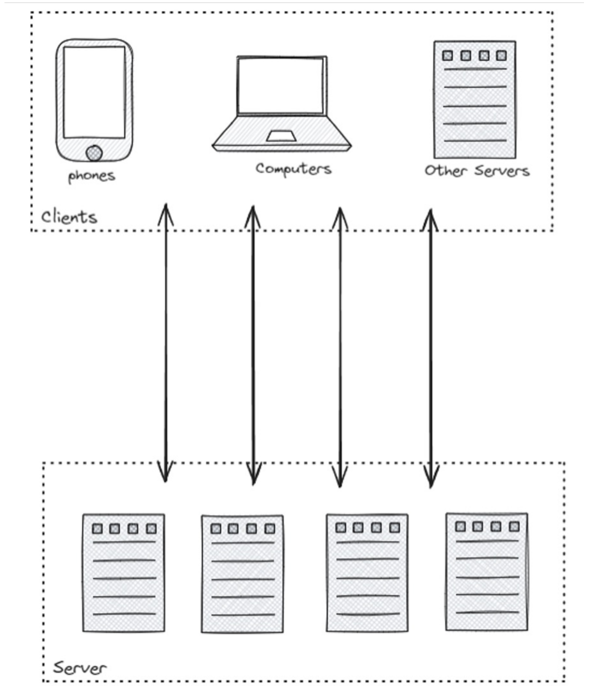
- Client-Server communication

- HTTP

- Quick Intro to Express

- HTTP Examples + Testing your APIs

- Restful APIs



Review: Client-Server Communication

- The **client-server** model is an architecture pattern where a **client** requests a resources or a service from a **server**

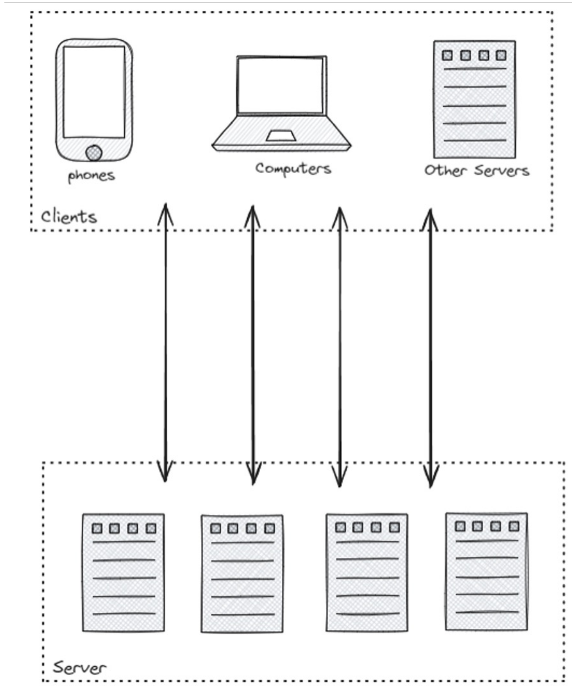
- **Today's Tutorial:** How can we facilitate the communication between two programs?

Communication Protocols

Many different means of communication.

To list a few:

- GraphQL
- WebSockets
- gRPC
- Message Queues
- **HTTP ← What we're covering today**



What is HTTP?

- **HTTP** stands for hypertext transfer protocol which is built on TCP/IP
- Follows a request-response pattern
- Client makes a request to server → server processes request and sends back response → client receives response

Components of HTTP

	Request	Response
Responsibility	Created by client, sent to server	Created by server sent to client
Main Components	● URL ● Method (GET, POST, PATCH, etc) ● Body (optional) ● Headers	● Status Code ● Headers ● Body

Anatomy of a Request URL

The url/address of the resource we want to operate on. Some components of a URL:
(E.g. https://localhost:3000/api/department/CSC/users?name=porom&job=unemployed)

- The scheme
  - **http** or **https**
- The server/domain address
  - In our example it’s “**localhost**”
- The port
  - Defaults to 80 if not provided
  - In our example it’s “**3000**”
- The path
  - The series of segments separated by a slash. Points to the name of a resource on the server
  - In our example it’s “**api/department/CSC/users**”

Anatomy of a Request URL

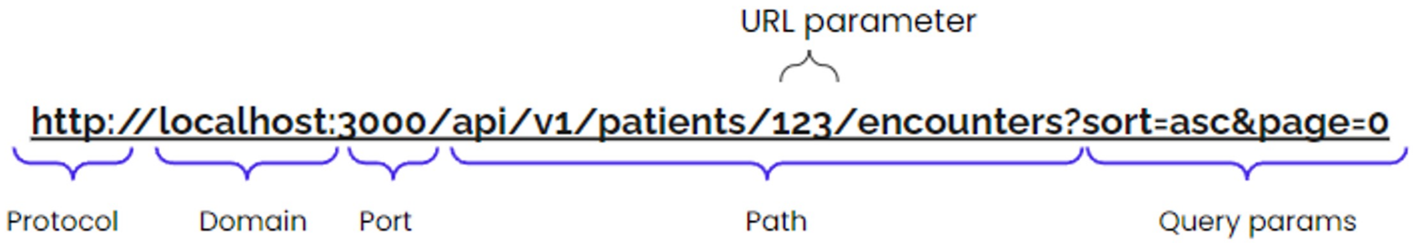
The url/address of the resource we want to operate on. Some components of a URL:
(E.g. https://localhost:3000/api/department/CSC/users?name=porom&job=unemployed)

- The path parameters
  - A dynamic variable in a URL path.
  - Can be defined in code with /:variableName
  - not immediately apparent just by looking at the URL
  - In our example it's "**CSC**", (/api/department/:deptId/users)

- The query
  - An optional URL input at the end, starting with a question mark (?), consists of key-value pairs separated by ampersands (&) or semicolons (;)
  - In our example it's **name=porom**, and **job=unemployed**
  - Usually used for filtering/sorting the results

Another Sample

(Courtesy of Cho Yin Yong, instructor for CSCC09 at UTSC! Thanks Cho)


URL parameter
http://localhost:3000/api/v1/patients/123/encounters?sort=asc&page=0
Protocol
Domain
Port
Path
Query params

Protocol: `http` (alternative is https)
Domain: `localhost`
Port: `3000`
Path: `/api/v1/patients/123/encounters`
Query Parameters: `{“sort”: “asc”, “page”: “0”}`
URL Parameters: `123` (patient_id)

Request
Body

Purpose:
Need a way to send (potentially large) amounts of data to our backend.

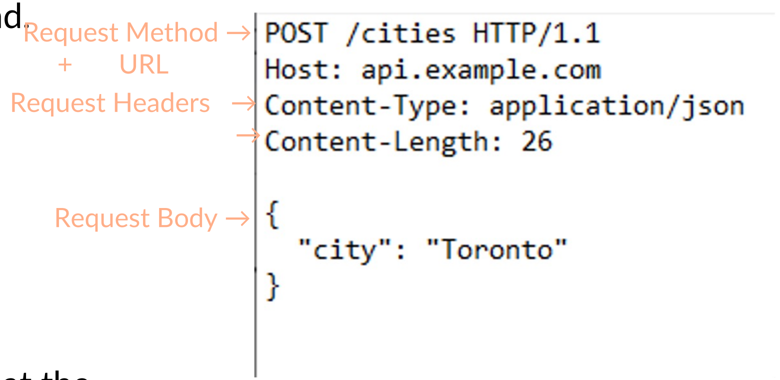
Can't pass it in as query params since there's a limit to URL size :(

Solution:
Include a "body" with your request.

You must specify the "Content-Type" so the server knows what format the body is in.

Most likely you'd only be using "application/json" format.

(Raw text of actual HTTP request)



```
POST /cities HTTP/1.1
Host: api.example.com
Content-Type: application/json
Content-Length: 26

{
   "city": "Toronto"
}
```

Request Headers

- Stores the metadata related to a request in the form of key-value pairs, examples include:
  - Type/format of data in request body (Content-Type)
  - Type of machine the request is coming from (User-Agent)
  - Authentication credentials/tokens for the request (Authorization)
  - Any custom headers you include

- The server can use the header to understand how it should parse/authorize the request

Request Method

The type of action you want to perform.

Method	Description	Inputs
GET	**Request** a resource	Query, Path Params
POST	**Create** a new resource	Query, Path Params, Body
PATCH	**Update** a part of an existing resource	Query, Path Params, Body
PUT	**Update** an entire existing resource	Query, Path Params, Body
DELETE	**Delete** a resource	Query, Path Params

Response Status Code

An integer describing the result of the request

Status code categories, and examples

(1xx series exempt from this list)

2XX - Success
- 200 - OK
- 201 - Resource created

3XX - Redirect
- 301 - Resource has been moved to a different URL

4XX - Client Error (Bad request)
- 404 - Not found
- 422 - Unprocessable content (Correct Syntax, server understands request but can't process it)
- 400 - Bad request (Incorrect Syntax, server doesn't understand)
- 401 - Unauthorized
- 418 - I'm a teapot, server refuses to brew coffee (real status code)

5XX - Server Error
- 500 - Internal Server Error
- 508 - Infinite loop detected
- 507 - Insufficient storage
- Not a good idea to be too specific with server error status, **why?**

Response

Headers

- Similar to request header, the response header contains metadata about the response:
    - Format of the response body
    - The server which generated this response
    - Timestamps
    - Any custom headers created by the server

Body

- Similar to request body, the response body contains data returned by the server
- Can be in JSON, plain text, HTML, etc

Demo
Stock management API

Now that we got the formalities out of the way, let’s build an actual Web API.

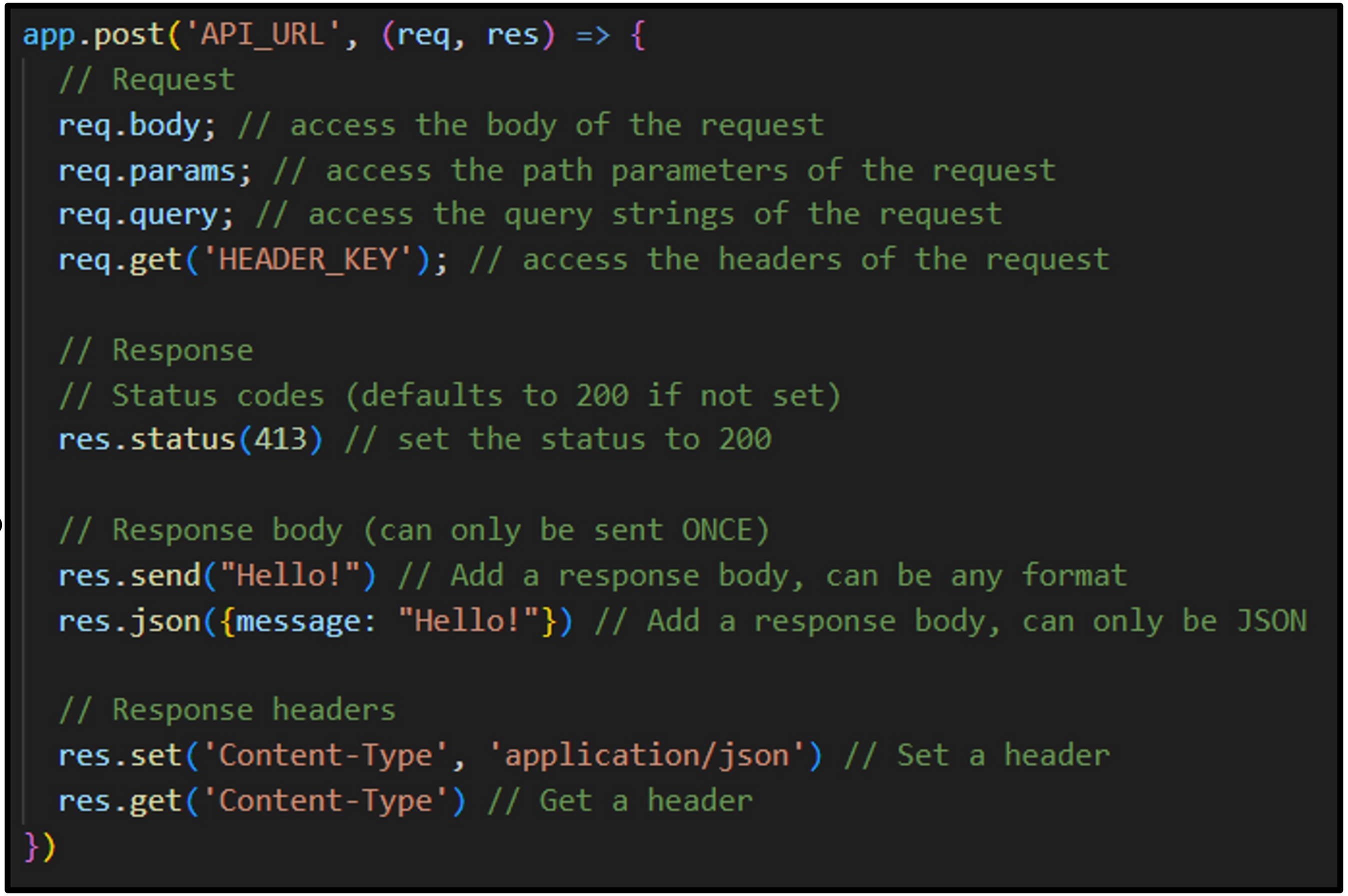
Download and Extract the tutorial demo zip file from Quercus.

Open in VS code, and run ‘npm run tutorial’.

Follow along with the tutorial.

Express.js

- A web application framework for node.js
- The anatomy of an HTTP endpoint in express:



```
app.post('API_URL', (req, res) => {
  // Request
  req.body; // access the body of the request
  req.params; // access the path parameters of the request
  req.query; // access the query strings of the request
  req.get('HEADER_KEY'); // access the headers of the request

  // Response
  // Status codes (defaults to 200 if not set)
  res.status(413) // set the status to 200

  // Response body (can only be sent ONCE)
  res.send("Hello!") // Add a response body, can be any format
  res.json({message: "Hello!"}) // Add a response body, can only be JSON

  // Response headers
  res.set('Content-Type', 'application/json') // Set a header
  res.get('Content-Type') // Get a header
})
```

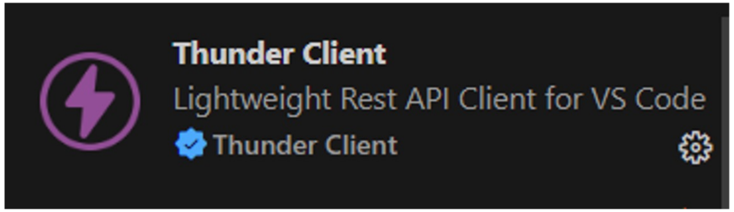
Testing your APIs

Many means of testing HTTP APIs:

- cURL (command line)
- Hopscotch
- Insomnia
- Postman (industry favourite, but has some recent controversy)
- Thunder Client VSCode extension (We’ll use this since it’s the most lightweight)

Thunder Client Setup

- Launch VSCode
- Navigate to extensions
- Search thunder client & Install



Thunder Client

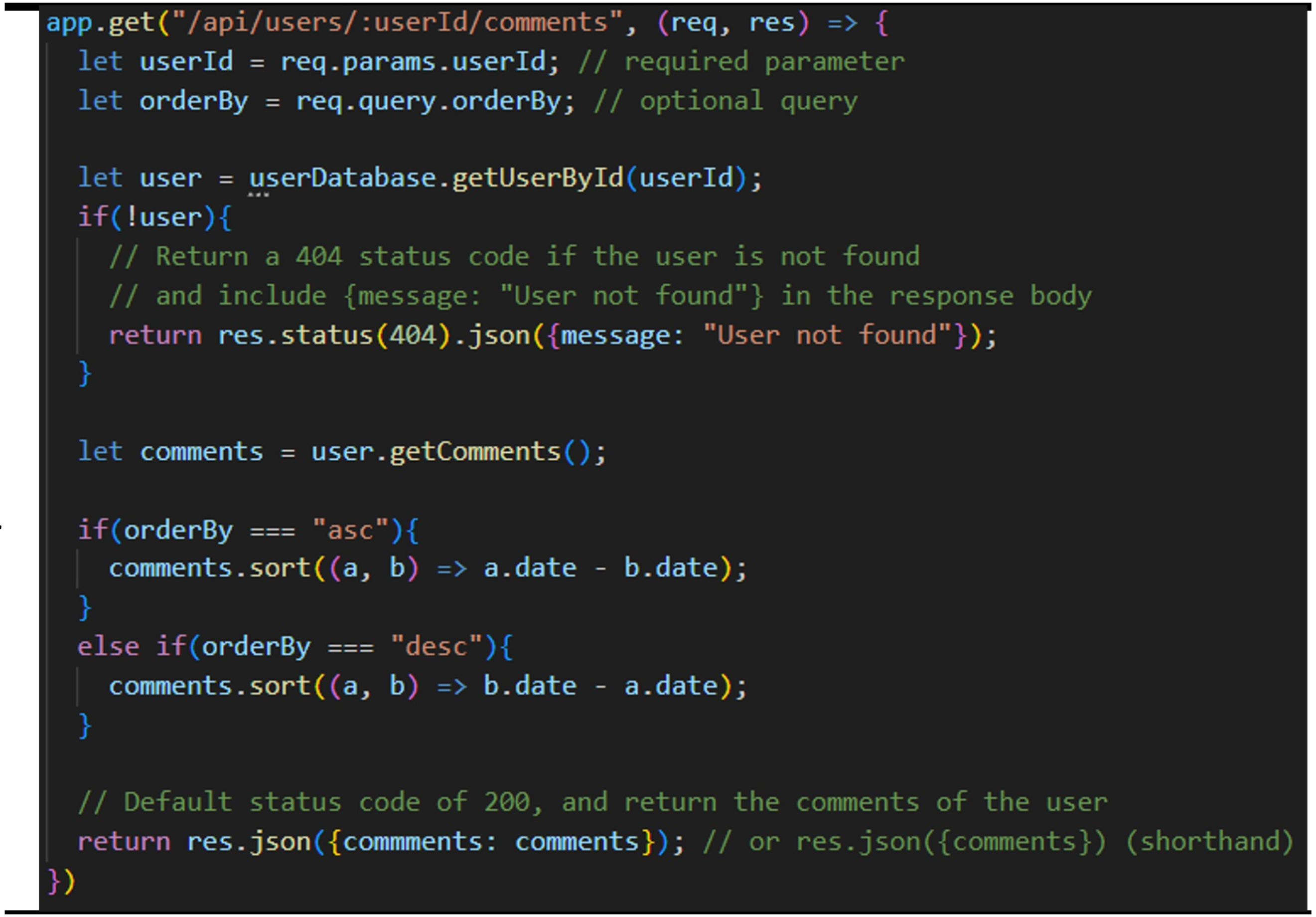
Lightweight Rest API Client for VS Code

Thunder Client

- Will be available on your sidebar:

Example

API to get comments for a user



```javascript
app.get("/api/users/:userId/comments", (req, res) => {
  let userId = req.params.userId; // required parameter
  let orderBy = req.query.orderBy; // optional query

  let user = userDatabase.getUserById(userId);
  if(!user){
    // Return a 404 status code if the user is not found
    // and include {message: "User not found"} in the response body
    return res.status(404).json({message: "User not found"});
  }

  let comments = user.getComments();

  if(orderBy === "asc"){
    comments.sort((a, b) => a.date - b.date);
  }
  else if(orderBy === "desc"){
    comments.sort((a, b) => b.date - a.date);
  }

  // Default status code of 200, and return the comments of the user
  return res.json({commments: comments}); // or res.json({comments}) (shorthand)
})
```

Demo
Create a Product

API Specification:

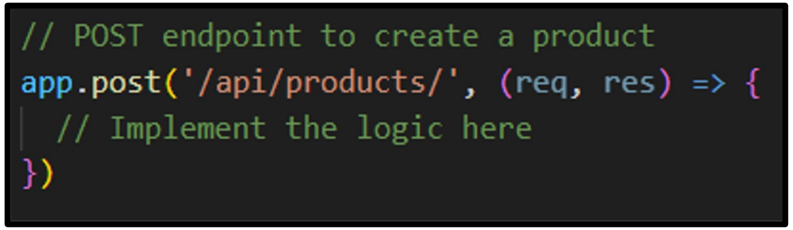
Description: Create a new product
Method: POST
Path: /api/products/
Request Body:
- name
- price
- stock

(Success) Response Body:
- product (newly created)
  - name
  - stock
  - id
  - price

What if name, and price isn't passed in the body?

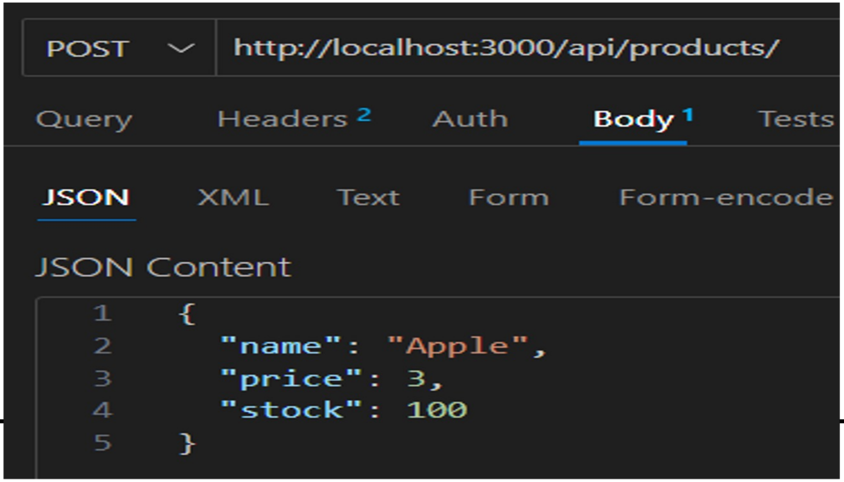
Instructions:

- Find the product endpoint, and implement the logic



```
// POST endpoint to create a product
app.post('/api/products/', (req, res) => {
  // Implement the logic here
})
```

- Test on Thunder Client



```
{
    "name": "Apple",
    "price": 3,
    "stock": 100
}
```

Demo

Get all products

API Specification:

Description: Get all products

Method: GET

Path: /api/products/

Optional Request Queries:

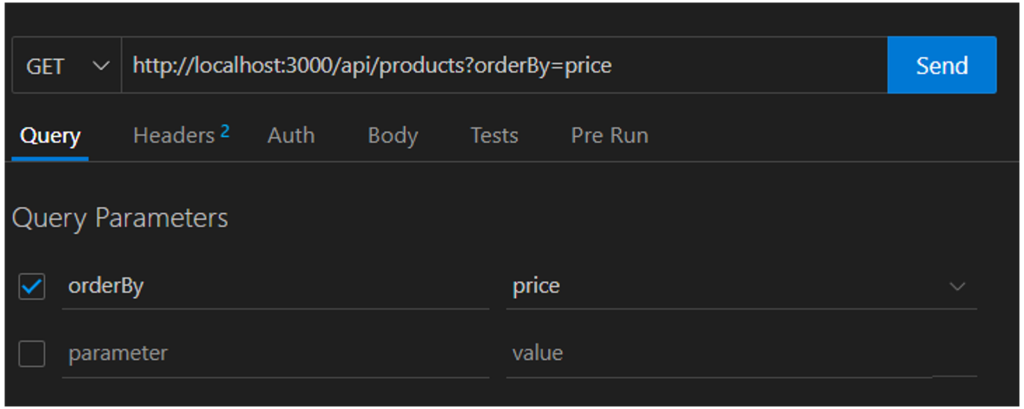
- orderBy (price or stock)
  - Order by price or stock, least to greatest

(Success) Response Body:

- products (array of products found)

Instructions:

- Find the product endpoint, and implement the logic
- Test on Thunder Client



http://localhost:3000/api/products?orderBy=price

orderBy: price

Challenge
Get a single product by id

API Specification:

Description: Retrieve a single product by it's id.
Method: GET
Path: /api/products/:productId
Path Parameters:

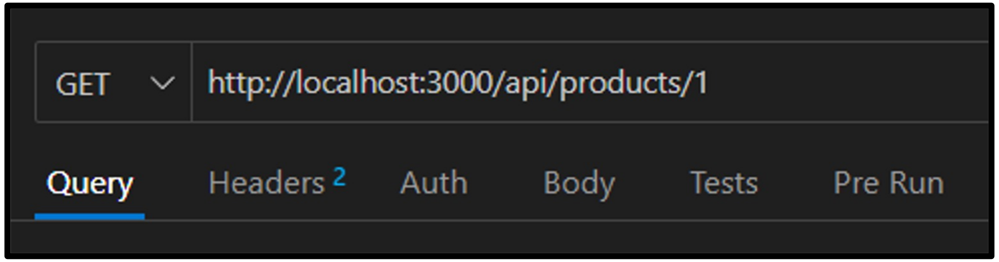
- productId

(Success) Response Body:

- product:
    - name
    - stock
    - id
    - price

Instructions:

- Find the product endpoint, and implement the logic
- Test on thunder client


GET http://localhost:3000/api/products/1

What about when productId doesn't exist?

Demo
Purchase Product

API Specification:

Description: Purchase a product by removing it's stock, and returning updated product, and the total cost of transaction.

Method: PATCH
Path: /api/products/:productId/purchase
Request Parameters:
- productId

Request Body:
- amount (# of units to purchase)

Instructions:

- Find the product endpoint, and implement the logic
- Test on Thunder Client

(Success) Response Body:
- product
  - name
  - id
  - price
  - stock (after purchase)
- cost
  - The cost of transaction

What about if the amount is more than in stock?

Challenge
Update a Product

API Specification:

Description: Update an entire product by it's id, and return the updated product.
Method: PUT
Path: /api/products/:productId
Request Parameters:
- productId

Request Body:
- name
- price
- stock

Instructions:

- Find the product endpoint, and implement the logic
- Test on Thunder Client

(Success) Response Body:
- product
  - name
  - id
  - price
  - stock

Challenge
Delete a Product

API Specification:

Description: Delete a product by it’s id
Method: DELETE
Path: /api/products/:productId
Request Parameters:
- productId

(Success) Response Body:
- The deleted product
    - name
    - id
    - price
    - stock

Instructions:

- Find the product endpoint, and implement the logic
- Test on Thunder Client

REST

Stands for **Representational State Transfer.**

Attributes of a web API which promote scalability, regardless of what communication protocol you’re using (although HTTP was designed with REST methodologies in mind).

The following criteria of a RESTful system:

- Follows client-server architecture
- Stateless
  - The client should provide all details necessary for a server to process a request.
  - No state should be stored on the server (but can be stored in a database)

REST

- Uniform interface
    - Use URI (Uniform Resource Identifier) to identify a resource
    - HTTP implements this using URLs
    - Manipulate/request resources through these URLs

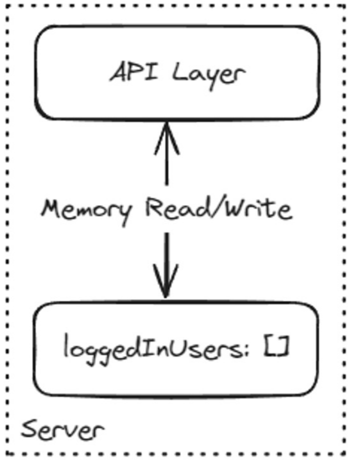
- Cacheability
    - Resources should be cacheable (on client or server), and responses should include information about whether a resource is cacheable (Cache-Control in HTTP)

- Layered System Architecture
    - Don’t assume client, and server directly connect to each other, a request or response could be going through a number of intermediaries.
    - Example: Load balancers, API gateways, Middlewares, etc

REST: Note about Statelessness

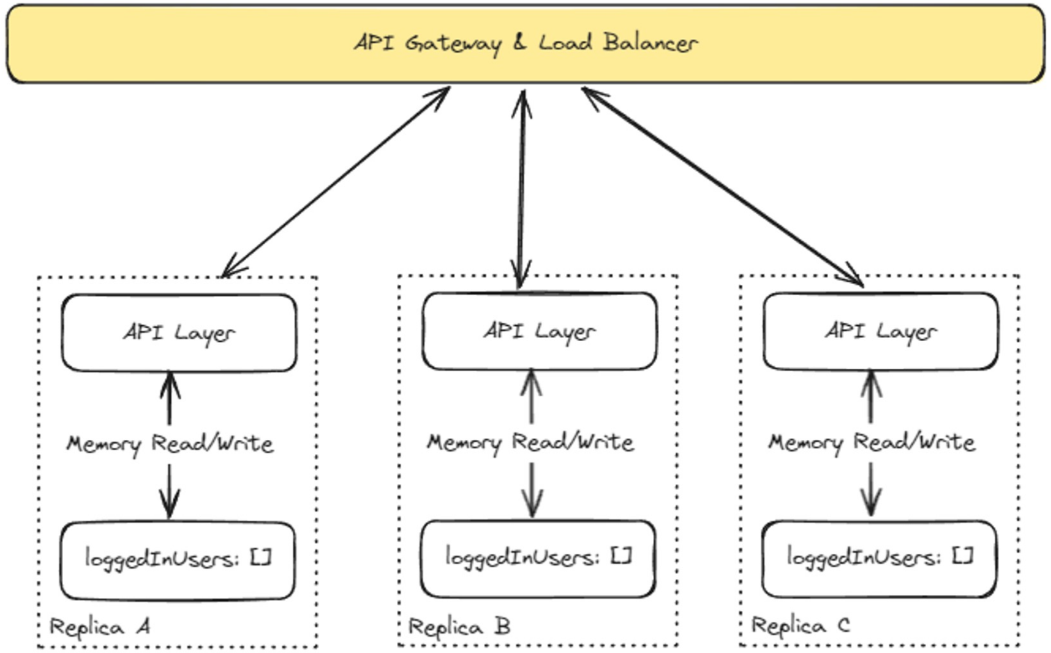
REST states that the server should be stateless. Let’s observe an example where a server is stateful, and some of its pitfalls

Consider a client/server architecture, where a list of currently signed in users is stored on the server in-memory (i.e. we are storing the currently logged in users as state on our server)



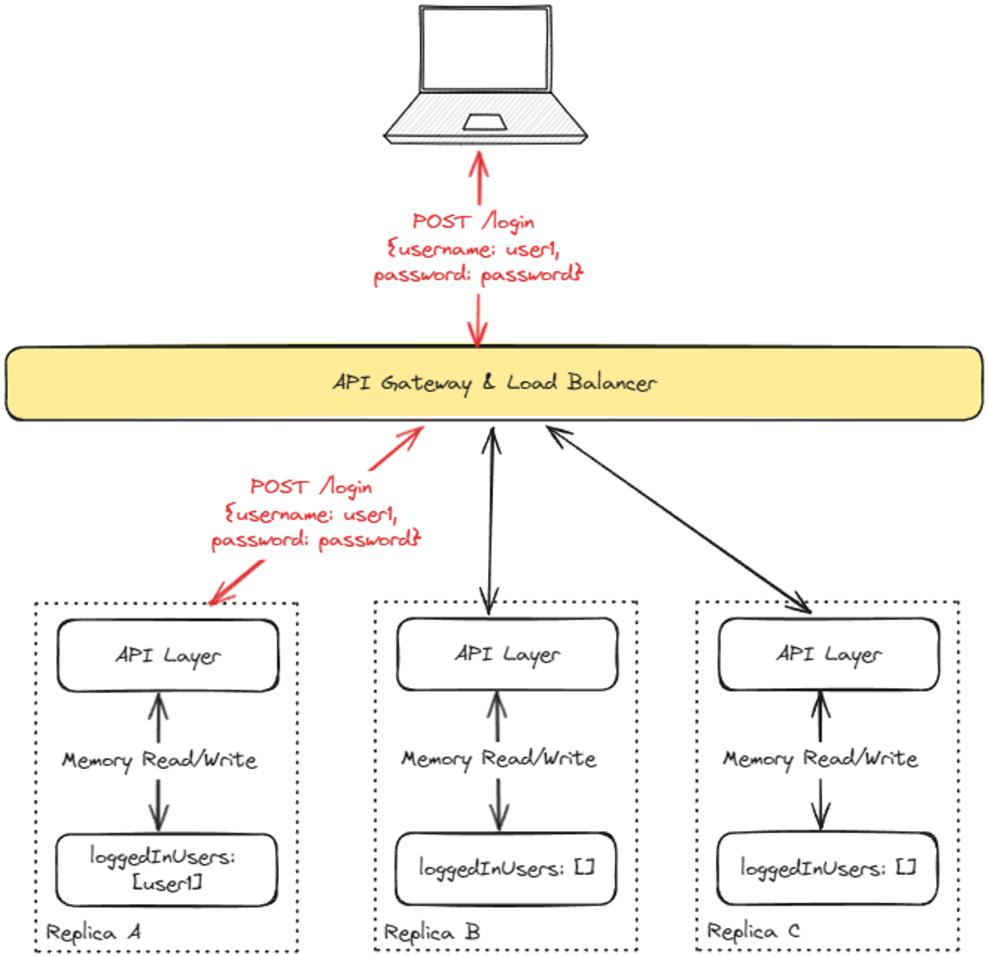
REST: Note about Statelessness

Consider a production environment, where we horizontally scale our server, and use an API gateway to load balance across all servers



REST:
Note about Statelessness

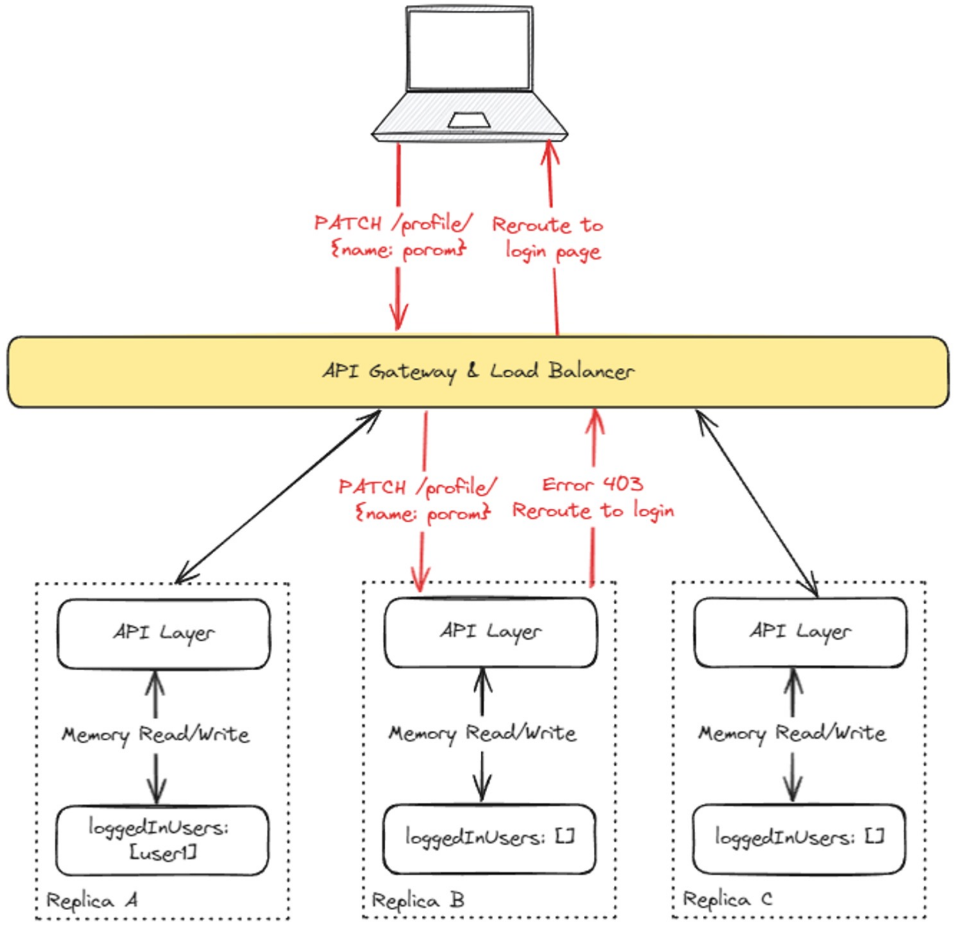
Consider a request where a user runs a login request, which the load balancer routes to Replica A



REST:
Note about Statelessness

Now the user requests to edit their profile. Our load balancer routes it to Replica B (Replica A has too much traffic at the moment)

We are met with an error 403, because our user is only logged in on replica A!?



REST:

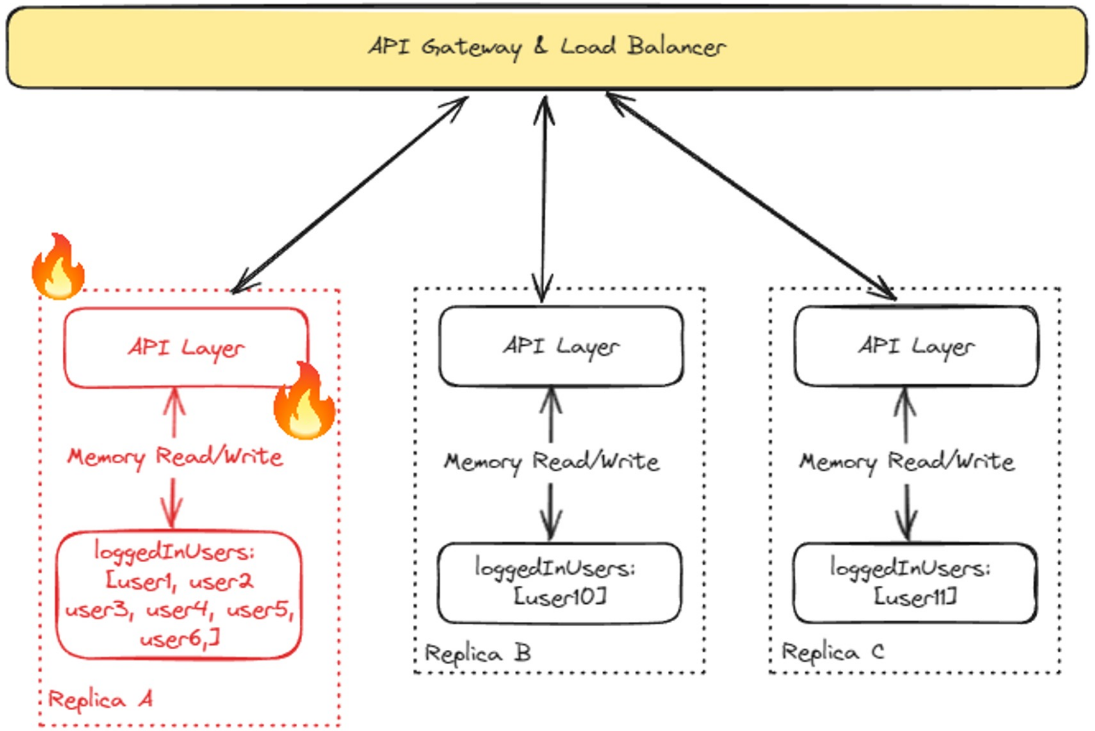
Note about Statelessness

Can no longer freely send request to any replica.

Requests can only be routed to a server that a user is authenticated on.

Defeats the purpose of our load balancer, and affects the scalability of our application :(

How can we become stateless?



REST:
Note about Statelessness

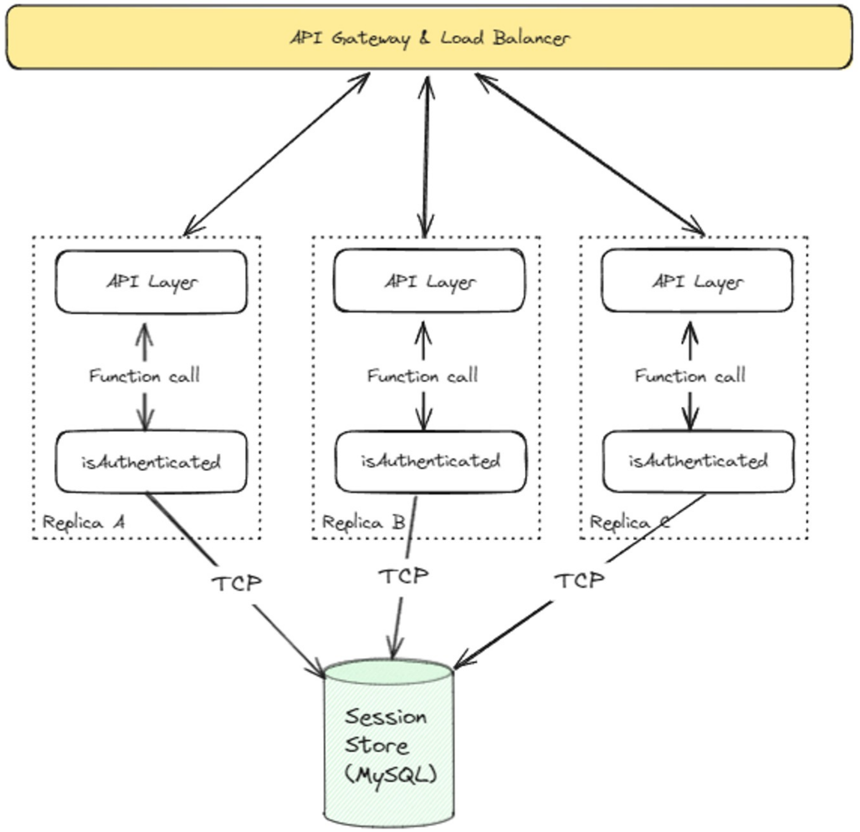
One option is to make use a **database** to store sessions.

- Simple to implement

But

- Slows down each request, since we need to query the database to authenticate
- Single point of failure, if our session database goes down, users can't authenticate into the system

Solution? **Signed Tokens**



Fin.

If you want to learn more about building Web APIs take CSCC09!